Goal-Oriented Control of Self-Organizing Behavior in Autonomous Robots

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
„Doctor rerum naturalium"
der Georg–August–Universität zu Göttingen

vorgelegt von

Georg Martius

aus Leipzig

Göttingen 2010

Prof. Dr. Theo Geisel (Referent)
Max-Planck-Institut für Dynamik und Selbstorganisation und
Abteilung für Nichtlineare Dynamik, Georg–August–Universität Göttingen

Prof. Dr. Ralf Der (Referent)
MPI für Mathematik in den Naturwissenschaften, Leipzig

Dr. J. Michael Herrmann
Intitute of Perception, Action and Behaviour, University of Edinburgh

Tag der Mündlichen Prüfung: 07. September 2009

Contents

Nomenclature

Symbol	Description
a	column vector, lower case Latin letters
a_i	element of vector a in row i
ξ, η	column noise vector, lower case Greek letters
ϵ	constant factor like learning rates, also lower case Greek letters
A	matrix, capital letters
A^+	pseudo-inverse of matrix A, see Section 4.1.3
A_{ij}	element of matrix A in row i and column j
λ_i^X	i-th eigenvalue of matrix X
$\mathbb{I}$	identity matrix
a^{-1}	componentwise inverse of a vector: $(a^{-1})_i = a_i^{-1}$
$\frac{a}{b}$	componentwise division: $\left(\frac{a}{b}\right)_i = \frac{a_i}{b_i}$
$a \circ b$	vector multiplication componentwise: $c = a \circ b$ means $c_i = a_i b_i$
$A \circ b$	row-wise multiplication of vector and matrix: $C = A \circ b = b \circ A$ means $C_{ij} = A_{ij} b_i$
$f(a)$	function f componentwise applied to a vector. If the argument is clear in the context then we write only f. Further we denote $f(a)_i = f(a_i) = f_i$
$F(a)$	usual function (takes a vector as argument)
F'_a	partial derivative of function F with respect to a $(F'_a)_{ij} = \frac{\partial F(a_i)}{\partial a_j}$
E, Ξ	error function (written without parameters)
$\overline{a}_t$	sliding time average of a_r, with $r \in (t-\tau, t]$ (applied componentwise on vectors)
$\langle a \rangle$	mean value of time series a_t (applied componentwise on vectors)

List of Symbols

Symbol	Description	Equation
x	sensor values	3.2
y	motor values	3.2
z	membrane potential	3.10
$W(\cdot)$	world function	3.3
α	hardware constant in toy world	3.7
$K(\cdot)$	controller function	3.2, 4.1
$g(\cdot)$	tanh activation function (componentwise)	3.30, 4.18
C	controller weight matrix (c in 1-D)	3.2, 4.18
h	controller bias vector	3.2, 4.18
$M(\cdot)$	world model function	3.16, 4.2
A	model weight matrix (a in 1-D)	3.26, 4.19
S	second model weight matrix	4.91
b	model bias vector	3.26, 4.19
L	Jacobian matrix in sensor space	3.23, 4.8
J	Jacobian matrix in motor space	4.15,4.34
R	Linear response matrix (r in 1-D)	3.10, 4.34
ς	additive sensor noise	3.7
ξ	prediction error (noisy)	3.17
η	prediction error at motor outputs	4.14
v	postdiction error	3.24, 4.11
ν	postdiction error in motor space	4.16
E	TLE (time-loop error)	3.25, 4.12
E^{x}	Some other error function labeled with x	4.70
γ_{x}	scaling of additional error x	4.71
ϵ	learning rate	3.19
$\psi(\cdot)$	dynamical system in sensor space	3.17, 4.4
$\phi(\cdot)$	dynamical system in motor space	4.13
$\Upsilon_{[a,b]}(\cdot)$	Clipping function to the interval $[a, b]$	4.47
v	translational velocity	4.105
ω	angular velocity	5.24, 6.37
$F_i(\cdot)$	prediction function of expert i	6.1
W^i	weight matrix of expert i	6.3
r	Number of experts	6.1
Ξ^i	prediction error of expert i	6.2
$\breve{\Xi}^i$	minimum of prediction error of expert i	6.17, 6.22
$\tilde{\Xi}^i$	penalized prediction error of expert i	6.18
p	penalty factor for suboptimality	6.18
ϱ	In Chapter 6: $\mathcal{Q}$-learning exploration	6.36
γ	In Chapter 6: $\mathcal{Q}$-learning discount factor	6.24
λ	In Chapter 6: $\mathcal{Q}$-learning eligibility trace decay	6.31
δ	small constant	
τ	time constant	
ζ	noisy quantity	

Chapter 1

Introduction

> People think of these eureka moments
> and my feeling is that they tend to be little things,
> a little realisation and then a little realisation built on that.
>
> *Roger Penrose*

Research interest in autonomous robotics has increased tremendously in the last decades. The reason is not only that robots are such an exciting topic but also because our society has many applications for intelligent mobile assisting devices, e. g. for the care of elderly people, for emergency and rescue operations or as a plain household aid. It is widely recognized that such complex robots need to be highly adaptive and self-learning machines in order to cope with the ever-changing environment and complexity of the real world [94]. The present thesis follows this line of thought by first considering the self-organized development of sensorimotor coordination in autonomous robots. In the second part these highly adaptive systems are shaped to achieve goal-oriented behaviors and finally to perform a given task – from playful to purposeful behavior.

1.1 From Classical Artificial Intelligence to Embodied Systems

Artificial intelligence (AI) is a research field that aims at understanding and synthesis of intelligence. Back in the 1950s, intelligence was thought to be the result of a symbol-crunching computer program located somewhere in our brain [101]. Since then the classical approach has developed into a large field of research with many branches and has produced impressive results in applied computer science and engineering. For example, the victory of IBM's chess-playing supercomputer (Deep Blue) over Garry Kasparov, the world chess champion in 1997. In such formal or computational domains, like game playing and logical

reasoning the traditional approach to intelligence seems to be adequate. However, it became evident that natural forms of intelligence observed in animals or humans on a daily basis cannot be explained with intelligence as a mere computational process, neglecting the body-environment interactions [124]. One of the most influential trends in AI and robotics started with Brooks in the 1980s when he promoted the so-called behavior-based approach [24, 25]. The important aspect of this approach is a tight coupling between sensation and action without the need for a complex internal representation. Influences from cognitive science led to the concept of *embodiment*, which states that intelligence requires a body [25, 28, 31, 122, 167]. From a more practical perspective this means that the robot with its controller, body, and environment must be treated as a unity. This went so far that the term morphological computation [115, 122], was coined, referring to the seemingly computational processes that are performed by the physical components of a robot. A prominent example is Tad McGeer's passive dynamic biped walker [95] that walks down a slope without any active control. Another example that illustrates nicely how the environment can shape the behavior of a system controlled in a reactive manner is the salamander. A salamander performs different motion patterns when walking on the ground or swimming in the water with the same low-level sensorimotor mapping [69, 70]. The new understanding of intelligence has led to a paradigm shift from knowledge-based systems to embodied reactive systems [33, 67]. Physical and information-theoretical implications of embodied adaptive behavior became important and the research focuses on systems acting in the real, physical world. Dynamical systems theory was increasingly applied to understand the lower level mechanisms underlying intelligent behavior [66]. Higher forms of intelligence are now approached by starting from lower level ones. Consequently, action and cognition are not supposed to be a result of something directly built into a robot but rather the result of emergence and development [48, 85, 110].

1.2 State of the Art in Autonomous Robot Control

Before we review the most prominent approaches to autonomous robot control that aim at the emergence of animal-like and intelligent behavior, let us first put them in coarse relations. Evolutionary robotics, analogously to biological evolution, optimizes robots over many "generations" to perform some task. However, the individual robots are mostly fixed during their lifetime. Somewhat complementary is reinforcement learning, which is used to obtain goal-oriented behavior in the lifetime of a robot. Developmental robotics is more concerned with intrinsic motivation for learning and behavior in general and thus focuses on the emergence of coordinated behavior without specific goals. In this matter information theoretic quantities have recently been shown to play an important role. Finally, there is the concept of homeokinesis that gives rise to the development of coordinated embodied behavior within short time scales. Let us now go into detail.

Perhaps the most prominent approach to autonomous robot control is *evolutionary robotics*. Stefano Nolfi gives the following accurate formulation in [104]:

> Evolutionary robotics is a new technique for the automatic creation of autonomous robots. Inspired by the Darwinian principle of selective reproduction of the fittest, it views robots as autonomous artificial organisms that develop their own skills in close interaction with the environment and without human intervention.

With genetic algorithms a fitness function, encoding some desired goal, is optimized in a high-dimensional configuration space. Evolutionary robotics allows for the evolution of a control program and/or the morphology of a robot to perform a desired behavior. This is achieved by generating a set of possible solutions (individuals), which are then iteratively recombined, mutated and selected until a satisfying solution is found. Evolutionary design was successfully used in many robotic applications [78, 104, 105, 114, 146], but, up to now it has not been possible to achieve an open-ended process, where more and more complex structures/behaviors emerge. Suggestions on how to achieve such an artificial open-ended evolution were given in [23, 102]. Raising computational power and further investigations of the role of morphogenesis give hope for emergence of increasingly complex systems in the following years. However, we are interested in the development of a robot during its lifetime, which is not addressed by evolutionary robotics.

The online learning of goal-oriented autonomous behavior is most often achieved with *reinforcement learning* [161]. Reinforcement learning (RL) is concerned with how an agent is to take actions in an environment to maximize a long-term reward. It is rooted in the concept of classical conditioning [116], which says that a reward or punishment is associated with an earlier presented conditional stimulus, such that later this stimulus is sufficient to predict the reward or punishment. A generalization of this concept is the temporal difference learning that is used to predict future rewards and to solve the reinforcement learning task [176]. Interestingly, it was found that the firing rates of dopamine neurons[1] of mammals show a convincing correspondence to the reward prediction error of the temporal difference learning theory [144]. RL algorithms attempt to find a policy that maps the perceived states of the world to the actions that maximize the long-term reward. A major milestone was achieved by Richard S. Sutton when he introduced the temporal difference learning algorithm called TD(λ) [158], which was even proven to converge to an optimal solution [35]. Since then many different approaches to the RL problem have been proposed, e. g. $\mathcal{Q}$-learning [169], adaptive actor-critics algorithms [79], policy gradient methods [12] and ISO-learning [77, 128]. $\mathcal{Q}$-learning, for instance, is proven to generate an optimal policy (action selection) in fully observable Markovian systems. These approaches have yielded some remarkable empirical successes in learning to play games, including checkers [139], backgammon [163], and chess [13], but also to control robots, for instance to play soccer [27, 156], perform navigation tasks [54], control a fast biped walking [87], and to control humanoid robots [120]. It is important to note that RL works best on discrete

[1] The dopamine neurons regulate the release of the neurotransmitter dopamine that regulates the strength of synaptic connections for neurons that use dopamine as neurotransmitters, e. g. the dopaminergic neurons chiefly found in the midbrain.

state and action spaces. In real-world applications, however, we find high-dimensional continuous spaces that make it necessary to use heuristics and sophisticated methods to overcome the curse of dimensionality. RL can also be applied to continuous state and action spaces using function approximators as proposed by Kenji Doya [52]. Nevertheless, it is still based on trial-and-error, which can be an exhaustive process. Without a few clues along the way, a learner can get lost exploring a huge space of poor solutions dominated by negative reinforcement [81]. Additionally, RL commonly considers behaviors as sequences of actions rather than as complete sensorimotor couplings.

A tight sensorimotor coupling is important for exploiting the feedback of the physical interaction, i. e. to make use of the embodiment. In this context the framework of dynamical systems, known from mathematics and physics, started to get increasing attention in the last two decades [15, 73, 112]. It is a powerful method to analyze [66] and construct [39, 73] robot controllers, as it allows one to formulate the evolution of a certain state of the system, e. g. the sensor values, in a quantitative manner and enables analytical and qualitative predictions. Dynamical system theory also led to the application of chaos control and coupled chaotic oscillators to robotics [80, 125, 153].

Already Alan Turing expressed the importance of developmental aspects of human intelligence: "Instead of trying to produce a program to simulate the adult mind, why not rather try to produce one which simulates the child's." [165]. This is now addressed in the fields of developmental robotics and active learning, which are concerned with open-ended learning and the source of motivation. Research in developmental psychology states that humans perform intrinsically motivated activities to experience a particular feeling of competence and self-determination [37]. Intrinsically motivated activities are also prominent at the level of motor control. When considering mammals, for example, we observe the development of complex sensorimotor coordination starting from the early phase of postnatal epigenesis, which cannot be fully described by the genetic code [56]. Especially interesting is the emergence of playful behavior, observed in many young mammals. For example, a kitten first learns to walk from more or less random movements and then starts to play with everything it can find. The reason why nature produces playful behavior is because it enhances motor skills and the self-model of the animal that become vital in its later life. Long ago it was realized that intrinsically motivated exploratory activities have a fundamentally different dynamics than goal oriented behaviors: they are not homeostatic, the general tendency to explore is never saturated, and it is not a response to a perturbation or deficit of any non-nervous-system tissue [171]. There have been a couple of efforts to obtain an intrinsically motivated system. Jürgen Schmidhuber proposed in 1991 a method to introduce curiosity in a control system. He proposed to use the learning progress (compression progress in his terms) of an internal model as a reward for a reinforcement learning system [143]. The learning progress is indeed compatible with our intuitive understanding of curiosity and boredom – easily predictable things get boring quickly but also inherently unpredictable things, like the noisy display of a TV-set out of reception do not receive our attention for more than a moment. Most other approaches to developmental robotics also use the combination of RL and some intrinsic reward [65, 88]. Pierre-Yves Oudeyer and

Frédéric Kaplan, instead, estimate the predictability of a situation with a meta-predictor. They have been able to show playing behavior in a Sony Aibo robot [172], however with very limited amount of discrete actions [110, 111]. Formulating the learning progress in a probabilistic manner J. Michael Herrmann was able to obtain behavior for the sake of information gain, resulting in a vivid interaction among agents [61] similar to the evolved turn-taking robots by Hiroyuki Iizuka and Takashi Ikegami [68].

In general, information theory, as one of the most universal concepts in natural sciences, has gained increasing attention in the field of robotics. Starting from the definition of information entropy by Claude E. Shannon [145] it now offers a formal language to analyze and compare seemingly different systems quantitatively. The application to robotic systems is rooted in the observation that many biological information processing systems are optimal with respect to some quantity [10, 21]. Recent developments have shown that these optimization criteria can be expressed in an information theoretical way, using quantities like mutual information between certain channels or predictive information in the stream of stimuli [22, 133]. Max Lungarella and Olaf Sporns demonstrated that maximizing the information structure in the sensory stream can result in coordinated behavior [150]. In another study they analyzed how the interaction with the environment and the morphology of a situated robot shapes the information structure in the sensorimotor loop [86]. A more general discussions about embodiment and information theoretic perspectives can be found in [123, 127]. However, it is not clear how to directly obtain an on-line control strategy from information theoretic quantities. On the one hand a long sampling is required and on the other hand it is difficult to obtain the dependence of the quantities on the control parameters. Thus, the proposed methods so far use evolutionary algorithms to optimize the quantities and obtain therefore only static controllers. The information theoretic approach is not to be confused with the field of probabilistic robotics, pioneered by Sebastian Thrun and Wolfram Burgard, which achieved remarkable results in the context of localization, mapping and navigation of autonomous robots [164].

Since we are interested in the emergence of behavior in robotic systems the phenomena of self-organization is of special interest for this work. Self-organizing processes bring about the emergence of structure or function in many complex systems ranging from the fields of physics, chemistry, computer science, economics and biological systems [58, 174]. As the word *self-organization* suggests, it is about the evolution of a system in an *organized* form that comes about from internal drives, thus from it*self*. A broader definition is given by Hermann Haken: "A system is self-organizing if it acquires a spatial, temporal or functional structure without specific interference from the outside." [59]. It is not surprising that artificial intelligence and robotics aim at systems that exhibit self-organizing processes in one way or another. For example, swarm robotics [14, 17] uses mechanisms inspired by the pattern formation of flocking birds [107, 132]. Self-organization is also found in learning systems, for instance the structure of self-organizing maps [76] is not externally specified but rather emerges in the course of learning.

A general principle that tries to explain the functionality of complex self-organizing biological systems was introduced already in 1932 by Walter B. Cannon [26] and later by W. Ross Ashby [5]. It is called *homeostasis* and asserts that a biological system acts to maintain physiological variables at certain levels. As we know, many aspects of animal and human body-functions can be indeed explained by the homeostasis of e. g. the blood pressure, the body temperature, sugar levels and so forth. However, it cannot alone account for the generation of coordinated behavior [171]. Inspired by homeostasis and the power of self-organizing processes Ralf Der proposed a novel concept called *homeokinesis* [39, 44] at the end of the last century. This principle aims at the self-organized generation of body and environment related sensorimotor coordination without a specific goal – ergo intrinsically motivated playful behavior. The name homeokinesis reflects that a kinetic quantity or dynamic regime is to be kept in a certain range. In fact, this range is chosen to be at the edge-of-chaos. The main edge-of-chaos hypothesis asserts that biological systems, performing complex computation for survival, operate near a phase transition between ordered and chaotic behavior [97]. This shows a close similarity to the concept of self-organized criticality [9] known from physics. In the application to robots this can be intuitively phrased as "active but predictive behavior" and leads to the spontaneous self-organization of many behavioral patterns [42, 48]. Since we are convinced that playful and self-exploratory behavior is one of the important prerequisites for the development of complex goal-oriented behaviors in a self-learning autonomous robot, we use the homeokinetic approach as the basis for our research.

1.3 Research Questions and Thesis Overview

The following three major questions are investigated in this thesis:

- How can the homeokinetic controller be extended to operate on more complex systems,
- how can the self-organized robot control be guided to specific behaviors, and
- how can autonomous robots develop a pool of motoric skills in a self-driven way?

We are aiming at control algorithms that can be ultimately used in real world applications, which implies continuous sensor and motor value streams and realistic environments. Since the real world is much more complex than any analytically tractable model it is necessary to check the hypotheses and algorithms on real robots or in realistic simulations. The construction of a real robot is without a doubt the most rigorous and also entertaining way of validation, but it is also very time consuming and costly. That is why we created only one of those. For virtual world experiments the author developed an efficient and physically realistic simulation tool [91], which allows for versatile experiments and is now used by many researchers. The robot simulator and robotic platforms are presented in **Chapter 2**.

In order to tackle the main questions it is important to first understand the dynamics, the type of behaviors generated, and the limitations of the homeokinetic control. For that

Chapter 3 introduces the concept of homeokinesis and highlights its essential properties in the one-dimensional case. The multidimensional system is considered in **Chapter 4**, where we present a uniform mathematical description of the system in different domains. In the application to various robotic platforms we find highly embodied (body and environment related) and coordinated behaviors [43, 46, 63]. The analysis also led us to the re-examination of basic constitutes of the system. Even though the initial homeokinetic controller achieves a vivid interaction for many robotic systems, we find plenty of room and necessity for improvement also on the basic level. For example, the introduction of suitable regularization measures yields a more robust control algorithm and the extension of the adaptive internal world model will turn out to be essential for the control of more complex robotic systems.

To achieve specific desired behaviors, we studied how to influence and shape the self-organization process. In general terms, this approach is called *guided self-organization* [130, 136]. As the name suggests, the core idea is to combine goal-oriented design with self-organized development to obtain a system which unites benefits of both. This is especially important in high-dimensional systems where the self-organized search for useful behaviors can take a very long time and it is not guaranteed that all possible behaviors are visited in finite time. The novel idea of guided self-organization can be applied to many problems in modern physics, e. g. in nano material science [30], and it is especially suitable for developmental robotics. So far, there was only one international workshop in 2008 on this topic [130], which postulated first conceptual ideas. Already before that we presented the first guided self-organized robot in 2007 [90]. To our knowledge, the first application to swarm robotics was also recently presented in [136].

The main challenge is how to balance guidance and self-organization such that the system achieves the imposed goals without losing its flexibility and ability to re-organize itself. To deal with this we propose two novel and general methods to shape the emergence of behaviors in **Chapter 5**. The first one deals with the integration of additional energy functions into the original formulation of the homeokinetic controller using an appropriate metric for the gradient descent. Furthermore we propose a mechanism to specify symmetries of the physical system or of the desired behavior as soft-constraints. This produces efficient self-exploration in high dimensional systems and allows one to achieve specific behaviors. The second method uses an online reward signal to modulate the speed of search in the behavior space, which has proven to be very effective in some applications [90].

One of the major achievements of the homeokinetic controller is its ability to find coherent behaviors even for very complex and high-dimensional systems. Unfortunately, in its present formulation all obtained behaviors are forgotten as soon as new behaviors are exhibited, such that it cannot be used in many applications. A first step into the integration of a long term memory was proposed by Frank Hesse [62], where behavioral changes are stored in order to anticipate unpredictable situations. We present in **Chapter 6** a generic solution to the long standing problem of fading memory. We developed an additional system that extracts and stores the most successful behaviors from the lively interacting

robot [89]. This novel combination empowers a robot to acquire a repertoire of behaviors without human intervention. The acquired behavioral primitives are then used to generate higher order behaviors and to solve practical tasks, which we demonstrate in two applications.

In the **last chapter** we conclude our results and discuss perspectives for further research.

The research in the present thesis is reported in the "we" form, independently of whether the results have been obtained in collaboration or solely by the author. The following section presents the most important results obtained by the author at a glance.

1.4 List of Most Important Results

- Complex virtual robots and environments can be set up in a short amount of time and can be efficiently simulated with the developed robot simulator LpzRobots [91] (Section 2.1).
- Efficient usage of many sensors through the definition of sensorimotor dynamics in motor space (Section 4.1.2).
- Emergence of body- and environment-related behaviors from scratch is found in a new set of robotic systems and the systematic exploration of the behavior space was demonstrated (Sections 4.3 and 4.4) [43, 46].
- Proof that the homeokinetic controller prevents a deprivation of the adaptive world model by exploring all action subspaces. (Section 4.5) [45].
- Foundation for the incorporation of goals into the original homeokinetic learning dynamics by appropriate integration of additional error functions (Section 4.7.1).
- Enhanced world model that resolves the ambiguity between self-induced and environmentally-induced sensations. Thus, the controller can cope with action-independent dynamics in the environment (Sections 4.8 and 4.8.5).
- Guided self-organization via direct teaching and cross-motor teaching leads to the development of specific behaviors. Cross-motor teaching empowers fast coordinated behavior in high-dimensional systems with little given information while maintaining adaptability (Sections 5.1.1 and 5.2).
- Shaping of the self-organizing behavior via online reward signals brings about a preference for desired behavior (Section 5.3) [90].
- Acquisition of behavioral primitives with competing experts. Each expert reproduces a certain behavior that was before generated by the homeokinetic controller. This is the first time a generic long-term memory was integrated into the system (Sections 6.1.4 and 6.1.5) [89].
- Task solving through the combination of behavioral primitives with reinforcement learning, demonstrated with an obstacle avoidance task (Section 6.2.3).

Chapter 2

Robot Simulation Environment and Robotic Devices

> After three days without programming,
> life becomes meaningless.
>
> *Master Programmer in "Tao of Programming"*
> *by Geoffrey James*

This chapter introduces the workbench used for the robotic experiments in this work. First the developed robot simulator is introduced. We highlight its most important features and present a novel collision model. Subsequently, the collection of virtual and real robotic platforms that are used in the present thesis are shortly described. These sections may be used as a reference for the forthcoming chapters. The hasty reader might skip this chapter and return to the description of the robots as required.

2.1 Robot Simulator LpzRobots

Realistic computer simulations are very important not only for experimental scientists but also for theoretical studies. They allow one to quickly check hypotheses and algorithms and verify generalizations and approximations that have been done in the course of analytical derivations. This is especially fitting for robotics, where the hardware is normally error-prone and requires rather intense maintenance. However, many argue that robot experiments must be performed with real robots only. This harsh opinion is rooted in the fact that software controllers tested in simulations often have not been able to reproduce the same results in reality. Nevertheless, computer simulations are a valid test-bed and provide at least a good starting point for the development of controllers [55]. The gap between reality and simulation is also shrinking because we can nowadays perform physi-

cally realistic computer simulations. Moreover, control algorithms for autonomous robots should be adaptive enough to cope with the reality gap.

Despite the fact that there are many robotic simulators available, none of them were suited to our needs. In the initial phase we verified several simulators, such as Player/Gazebo [109] and Webots [32]. For example, the Gazebo simulator does not support custom materials and Webots is an expensive proprietary simulator that cannot be customized easily. This led us to the development of a new simulator called LpzRobots. The design and the major part of the implementation was conducted by the author. Further credits go to Frank Güttler, Frank Hesse, Ralf Der and Marcel Kretschmann.

This chapter is particularly aimed at those who plan to use the simulator, those interested in physical robot simulations in general and those who want to have a look at the back-stage of our virtual world of self-organizing creatures. In the next section we focus on the overall structure of the simulator. Afterwards we will describe the user interaction (Section 2.1.2) and show how to create virtual worlds (Section 2.1.3). In Section 2.1.4 we present an new method to handle material properties in the simulation. It follows a description of the matrix library (Section 2.1.5) and a list of highlighted features (Section 2.1.6). A comprehensive documentation with technical details and the source code is available on the project website [91].

2.1.1 Structure

Let us now consider the major design choices and the overall structure of the simulator. The heart of the LpzRobots simulator is the main loop that performs a time discrete physical simulation and determines the information flow. All important parts of the environment and the robots can be specified in terms of rigid bodies with geometric shapes and physical properties. The latter include the inertia tensor as well as surface properties. To be able to observe the simulations a graphical representation is optionally rendered.

The main reason to develop and use a robot simulator is to test control algorithms. Ideally, the control algorithms are quickly usable in other simulation environments and to control real robots. Therefore the interface between controller and simulator must be generic and the controllers should reside in a separate module instead of being tied into the simulator. For the development of our algorithms it is important to be able to observe the evolution of internal parameters online and to change some control parameters like learning rates during the runtime[1]. For that reason, a framework for controllers called SelfOrg was developed independently from the simulator. It allows for quick controller development and a flexible connection of robotic systems and controllers, which will be subject of the next section. Since the software is written in C++ we used the concepts of object-oriented programming. The knowledge of C++ is not obligatory to understand the main points

[1]The modification of parameters is only necessary during the test phase. In the later robot experiments the parameters are not changed manually, except states otherwise.

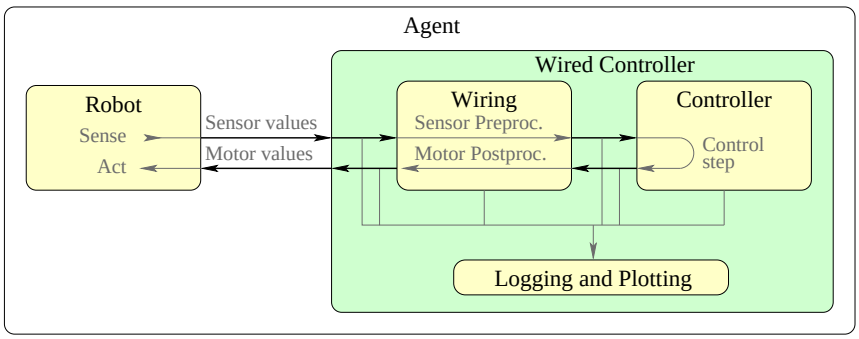


Figure 2.1: Core architecture of an agent with a wired controller and a robot. The arrows denote the information flow during one simulation step.

here. Nevertheless, some terms shall be briefly mentioned such as *class* that refers to an object type, *interfaces* which is an abstract class to specify only the signature, and *subclass* or *inheritance* for the mechanism to define a more specific class based on an existing one.

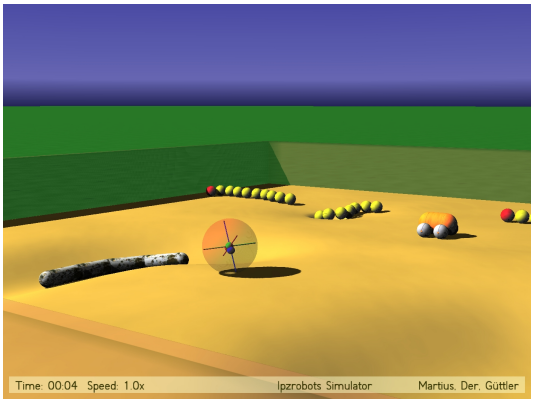
The SelfOrg Framework

The SelfOrg framework is designed for connecting a controller to any system, be it a real robot, a simple academic program, or our full-fledged robot simulator. The most important part is the `wired controller`, consisting of a `controller` and a `wiring` paired with some utilities to log, plot and configure the system. The wiring allows for the preprocessing of sensor and motor values, making the connection to different systems very easy. The wired controller might be directly integrated into another program, e. g. into a real robot control program. Alternatively it might be used within an `agent` together with the representation of a `robot`. Figure 2.1 depicts the information flow within an agent and its structure. Since all parts are specified using clear interfaces, a high reusability and interchangeability is achieved.

Simulation Class and the Main Loop

Let us now come the actual simulator. The central element is the `simulation` class with contains the main loop. In order to write a simulation, the user defines a subclass of this class and overloads typically only one function, the `start` routine to specify the environment, the obstacles, the agents, and specific parameters. Given that, the simulator enters the main loop and performs iteratively physical simulation steps using the "Open Dynamics Engine" (ODE) [149]. A control step (Fig. 2.1) is performed every n-th iteration (specified by the parameter `controlinterval`). This allows for a selection of the update rate independently of the step size of the physical simulator. The update of the graphical display, which is done using the graphics library "Open Scene Graph" [108], is executed

(a)



(b)

```
Welcome to the virtual ODE - robot simulator of the Robot Group Leipzig
--------------------------------------------------------------------------------
Press Ctrl-C on the console for a commandline interface.
Press h      on the graphics window for help).

[Simulation Environment]
 cameraspeed=          100.000000
...
 simstepsize=            0.010000 *
 gravity=               -9.810000 *
 realtimefactor=         1.000000 *
 drawinterval=          20.000000 *
 randomseed=          1219862012.000000 *
######
Type: ? for help or press TAB
>
?      help   list   load   ls     quit   set    show   store  view
> ls
Agents ---------------(for store and load)
ID: Name
  0: SchlangeLong
  1: Nimm4_0
...
  6: Sphere1
Objects --------------(for set and show)
ID: Name
  0: Simulation Environment
  1: InvertMotorNStep
  2: SchlangeLong
...
```

Figure 2.2: User interface of the LpzRobots simulator. **(a)** Graphical simulation window; **(b)** Terminal with console interface.

every k-th iteration, where k is calculated to achieve a proper frame rate of e. g. 25 fps. In order to obtain a smooth and continuous simulation the internal time of the simulation is synchronized with the real time. Of course, different factors are supported to speed up or slow down. The `simulation` class also has a variety of additional functions which can be overloaded to have sufficient control over the simulation process, define custom keystrokes, and trigger specific events.

2.1.2 User Interaction

The user interaction with the simulator is threefold. Operations concerning the display, e. g. camera position, display style or video recording are accessible through the graphical window. The camera can be manipulated with the mouse in combination with the mouse buttons in different modes, which are shortly introduced in Section 2.1.6. A transparent head-up display shows the simulation time and the simulation speed with respect to real time. All available keystrokes can be displayed on demand at a help screen. The second way of interacting with the simulator is via a console on the terminal window. It allows one to set parameters and to store and load controllers. The console features a history, auto-completion and many more characteristics of a UNIX shell. The interface was intentionally uncoupled from the graphics in order to be usable in real robot experiments or non-graphical applications. Both interfaces are depicted in Figure 2.2. Finally, the user can display internal parameters online, such as sensor values, network synapses and so forth with different custom tools like our `neuronviz` and `guilogger` as displayed in Figure 2.3.

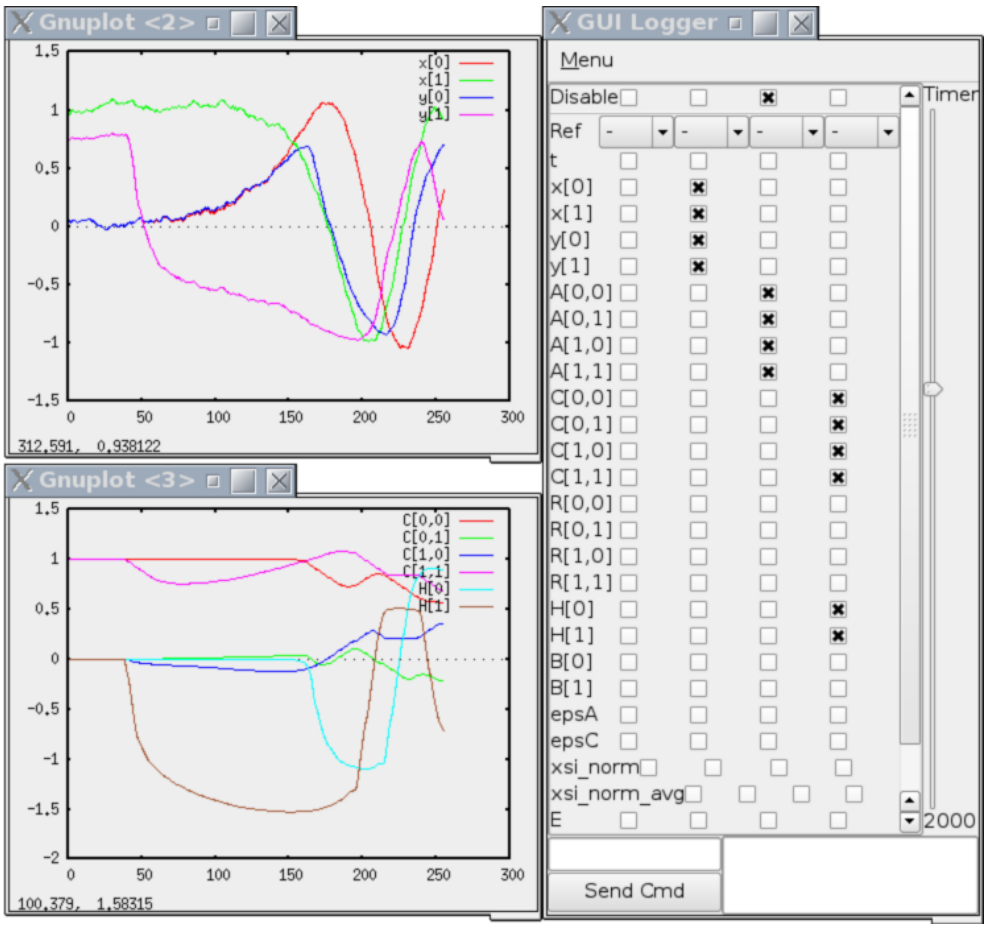


Figure 2.3: GuiLogger window with two controlled Gnuplot windows. In the main window (**right**) sets of channels are selected. Their temporal evolution is shown in the subwindows (**left**), here sensor values and motor values, and synaptic weight of the controller.

2.1.3 Creating the Virtual World

This section will give a brief overview of how to create virtual worlds in LpzRobots and discuss the major design choices. The usual problem in software engineering is to find the right level of abstraction. Unfortunately, object-oriented programmers often tends to stack one abstraction layer on top of another and opt for beauty and compactness at the price of flexibility.

While designing the simulator we had to combine physical, geometrical and graphical representations of the objects in the virtual world into one structure. This structure is called `primitive` and can have all of these properties. However, there are cases where no physical body or geometric representations are required or wanted. For example, static objects in the world, like walls or the floor, do not need a mass and impulse because they are considered to be unmovable. Likewise a massive weight inside of a robot, e. g. for balancing, does not need a geometric shape for collision detection. For that reason, we kept the abstraction layer thin and allowed for many customizations.

To build objects in the simulator one constructs primitives like spheres, boxes, capsules, cylinders and combines shapes. They can be pairwise connected by joints, which in turn can have motors attached to them. Eventually, the building blocks of the simulation must be positioned correctly. There are two ways to do that, either with a quaternion and a translation vector or with a 4×4 matrix containing both at once. We chose the latter, because it is much simpler in concatenation and application of transformations. A special case of homogeneous coordinates is used, which uses four dimensional vectors containing (x, y, z, w) where x, y, z code the space coordinates and w is 0 for a orientation vector and 1 for a position in space. The transformation matrices contain a 3×3 rotation matrix and a translation vector, which is only applied to position vectors.

$$\begin{pmatrix} x & y & z & w \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix} = \begin{pmatrix} r_{11}x + r_{21}y + r_{31}z + t_x w \\ r_{12}x + r_{22}y + r_{32}z + t_y w \\ r_{13}x + r_{23}y + r_{33}z + t_z w \\ w \end{pmatrix}^\top \tag{2.1}$$

This allows one to concatenate transformations via simple matrix multiplication. For complex objects like a multi-segment arm one can recursively add one segment after another by only multiplying relative transformations with the transformation matrix of the previous segment. In pseudo code we may write:

$m \leftarrow$globalPose
createSegmentAt(m)
for all $l \in$ localTransformations **do**
  $m \leftarrow l * m$
  createSegmentAt(m)
end for

2.1.4 Collision Detection and Surface Properties

One of the most important parts in the rigid body simulation is the detection and treatment of collisions. The Open Dynamics Engine (ODE) [149], which we use for the rigid body physics simulation, offers routines to check for collisions and proposes a number of so-called contact points. The simulator can create contact joints at such points to mimic surface interactions such as friction, bouncing and slip. In the following we will have a closer look at the developed strategy for efficient collisions detection and the realization of material and surface properties.

Efficient Collision Detection

To make collision detection practically computable also in larger systems ODE uses so called *collision spaces*, that group a number of preferably close geometric bodies together. Thus, robots, for example, usually have their own collision space. Collision detection is first performed on the level of collision spaces using their bounding boxes, i. e. checking for the intersection of the smallest cubes containing all bodies of a collision space. Only in the case of an intersection of the bounding boxes the geometric bodies of the two collision spaces must be pairwise tested. Additional collision tests within each space have to be performed as well.

Since not all collision spaces are supposed to treat internal collisions, there is a list of spaces to be checked. Further and more importantly not all geometric bodies are supposed to collide with each other. For instance, bodies connected with joints should typically not collide since they intersect by construction. To exclude pairs of bodies we introduced a hash set which is checked for each potential collision. The efficiency is ensured, because the hash set has a complexity in $O(1)$ for element search as long as only few hash-collisions occur. This is achieved by using the hash code $2b_1 + b_2$ where b_i are the memory pointers of the colliding body objects.

Material and Surface Properties

In order to model complex scenarios the collision treatment of the ODE needs to be augmented. Normally collisions are treated in a global callback function where the two colliding geometric bodies are given. In order to distinguish between different physical interactions, each geometric body carries a `substance`[2] description. The interaction parameters are then obtained through the combination of the two `substance`s. We consider four different parameters k_p, k_d, μ, and *slip* to describe the interaction. Here k_p and k_d denote the spring constant and damping constant respectively, μ is the Coulomb friction constant, and *slip*

[2]The name *substance* was chosen due to the fact that the possibly better fitting term, *material*, is already used by the graphics renderer to describe visual surface properties.

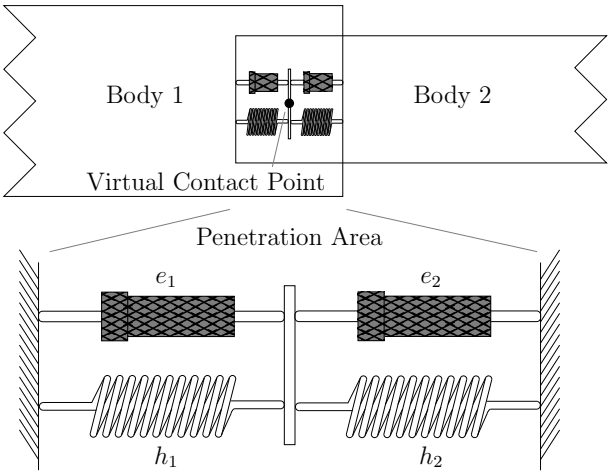


Figure 2.4: Material-interaction model with two spring-damper elements.

is the force dependent slip (FDS) parameter. FDS is used to model non-coulomb friction that occurs for example when the wheel of a car starts to slide sideways. More formally, the two contacting surfaces slide past each other with a velocity proportional to the force that is being applied tangentially to the surface. This differs from normal (Coulomb) frictional effects since it does not cause a constant acceleration, but rather leads to a steady velocity.

Our design for the substance parameters are roughness (r), slip (s), hardness (h), and elasticity (e). The Coulomb friction parameter μ is obtained by a multiplication of the roughness from both substances. This results in a high friction for two rough materials but in low friction, if one of the materials is very smooth (e. g. ice). The *slip* parameter is the sum of both slip parameters. The spring and damping constants are calculated using the schema of two spring-damper elements serially connected as illustrated in Fig. 2.4.

The spring constant of each collision side is given by the hardness h_1 and h_2. The spring constant k_p of the combined system is given by

$$\frac{1}{k_p} = \frac{1}{h_1} + \frac{1}{h_2}\,. \tag{2.2}$$

The damping constant k_d is derived from the elasticity e of the combined spring-damper system, but is more difficult to compute. Considering the damping in the form of energy loss we can write the energy or work done by each spring as: $W_i = F \cdot x_i = \frac{F^2}{h_i}$ using $F = h_i x_i$. The energy loss through damping is $W_i^D = W_i(1 - e_i)$. The final damping is now:

$$\begin{aligned} k_d = (1 - e) &= \frac{W_1^D + W_2^D}{W_1 + W_2} \\ &= \frac{F^2(1 - e_1)/h_1 + F^2(1 - e_2)/h_2}{F^2/h_1 + F^2/h_2} \end{aligned}$$

Table 2.1: Substance parameters and resulting interaction parameters.

Parameter	Range	Interaction Parameter	
roughness (r)	$[0, \infty)$	$\mu = r_1 \cdot r_2$	
slip (s)	$[0, \infty)$	$slip = s_1 + s_2$	
hardness (h)	$(0, \infty)$	$k_p = \frac{h_1 \cdot h_2}{h_1 + h_2}$	(2.2)
elasticity (e)	$[0, 1]$	$k_d = \frac{h_2(1-e_1)+h_1(1-e_2)}{h_1+h_2}$	(2.3)

$$= \frac{h_2(1-e_1) + h_1(1-e_2)}{h_1 + h_2} \tag{2.3}$$

Table 2.1 summarizes the parameters and their dependencies. Now the parameters k_p and k_d need to be converted to the parameters used by the ODE, which is described in the manual [149] and is given by $\texttt{ERP} = \frac{\Delta t k_p}{\Delta t k_p + k_d}$, and $\texttt{CFM} = \frac{1}{\Delta t k_p + k_d}$.

Special Cases

The above described approach is perfectly suitable for typical rigid body interactions, however, some special cases cannot be modeled. For instance, infrared distance sensors can be implemented with ray-objects. Their sensor value is obtained from collision routines. This is supported by an optional callback function of `substance` class which can overwrite the default collision treatment. Another example of a special case is when a material has different properties in different directions, like the skin of a snake. A similar situation occurs when a conveyer belt is to be modeled. The uniform motion of the surface can be implemented by custom collisions, instead of modeling the complicated belt structure.

2.1.5 Matrix Library

The author developed a small matrix library that is particularly suitable for the development of our controllers but is nevertheless of general nature. It is part of the homeokinetic controller package but can also be used and downloaded independently. Most available matrix libraries focus on the optimization for large and often sparse matrices but lack a concise syntax. Ideally one wants to write mathematical formulas in a one-to-one fashion in the program code. This requires a compact syntax and convenient operators. Another design criteria was the simplicity required when operating on simple embedded systems like the Atmega chips [6] used for our real robot experiments.

The main features of the matrix library are automatic memory management, operator overloading, and save operations. Vectors are treated as matrices with only one column or one row. For each operation there is a copy version and an *in situ* version. The latter can be used for optimization purposes, but require careful thinking. The copy operations work like their mathematical counterparts. Their operands are not changed and the result

Table 2.2: Matrix operations of the matrix library (excerpt).

function name	operator	meaning	description
`C(A)`	`C = A`	$C = A$	copy operation
`C.add(A,B)`	`C = A + B`	$C = A + B$	addition
`C.sub(A,B)`	`C = A - B`	$C = A - B$	subtraction
`C.mult(A,B)`	`C = A * B`	$C = A \cdot B$	multiplication
`C.mult(A,f)`	`C = A * f`	$C_{ij} = A_{ij} \cdot f$	scalar multiplication $f \in \mathbb{R}$
`C.exp(A,i)`	`C = A^i`	$C = A^i$	exponent $i \in (-1, 0, 1, 2, T)$
`C=A.multrowwise(b)`	`C = A & b`	$C = A \circ b$	row-wise multiplication
`C=A.map(g)`		$C_{ij} = g(A_{ij})$	function application
`C=A.mapP(arg,g)`		$C_{ij} = g(\text{arg}, A_{ij})$	func. appl. with argument
`C=A.above(B)`		$C = \begin{pmatrix} A \\ B \end{pmatrix}$	vertical concatenation

is a new matrix. All operations perform range checks and insure dimension compatibility which can be globally switched off for performance reasons. Beside the usual arithmetic operations for matrices like $+, -, *$ we introduce multiplication with a scalar, the exponent operator and many more useful operations as listed in Tab. 2.2. Special attention should be given to the exponent operator which, depending on the argument, is either the inversion (-1), the identity matrix (0), the matrix itself (1), the square (2), or the transposed (T). The latter is implemented by defining a symbol $T = 255$, which is therefore just a number and can be treated accordingly. Another interesting operation is the function application operator called `map` following the style of functional programming. It applies a function to all elements of a matrix componentwise. For the exponent and the row-wise multiplication we reuse the operators `^` and `&` that are originally used for bit-wise operations. Somewhat counterintuitive is their operator precedence (order of evaluation) which is lower than for the other arithmetic operations. Since the precedence cannot be changed in C++ more parentheses must be used.

The following illustrative code example shows the generation of a random matrix

```
Matrix W(20,5); // creates a (20 times 5) matrix initialized with 0
W = W.map(random_minusone_to_one)*0.01 // assign random value
```

where `double random_minusone_to_one(double)` returns a random number between -1 and 1. The program code for the equation $y = \tanh(Cx + h)$ is for instance

```
Matrix y      = (C * x + h).map(tanh);
```

which looks very similar to the original mathematical formulation and is therefore quickly written and more easily understood.

The performance of the code is still high even though new matrices are created for intermediate results. We use high performance memory operation like `memcpy` and `memzero` to

speed up initialization and copy procedures. Additionally, all range checks can be excluded at compile time after the code was successfully tested.

2.1.6 Highlights

Apart from the above mentioned features the simulator has much more interesting properties, some of which will be listed in this section.

Simulation speed: The speed of the simulation can be controlled by a single factor, determining the ratio between simulation time and real time. The simulation is synchronized with the clock so that a homogeneous time flow is achieved independent of the currently available CPU resources, provided that they are sufficient. This synchronization can also be turned off in order to run at maximal speed. Depending on the complexity of the simulation the speedup is up to 300 times and is usually above five on a typical desktop machine (about 2.8 GHz Pentium IV).

Camera Control: The user can operate the camera in different modes and control its movements with the mouse. The modes include a static camera that can be swayed and tilted up and down and moved in all directions. Another mode is the *following* mode that automatically moves the position of the camera relative to a selected robot. The *TV* mode points the view of the camera towards a selected robot. All camera movements are additionally smoothed to avoid unnatural jumps. These modes were mostly implemented by Frank Güttler and details can be found in his master thesis [57].

Video Recording: To capture the simulation in a movie, the user can at any time start and stop the recording of a video. This might slow down the simulation speed, however the video is kept at the right rate.

Multi Threading: Since computers nowadays have several processing cores it is important to parallelize the execution of code. The simulator can optionally run the graphics, physics, and controllers in separate threads. To be able to reproduce results, the handling of random numbers was modified to cope with the parallel processing.

Playgrounds: In order to create an appropriate environment for robots, there are several obstacles and rectangular and round arenas available. Additionally, a customized wall configuration can be created with the common 2D-CAD program `xfig` [148] and imported into the simulator, see Figure 2.5.

Terrains: For more challenging environments, a terrain using a height-map bitmap can be created, see Figure 2.2.

Many Examples: In the current version the simulator bundle contains about 15 sample simulations and 25 robots.

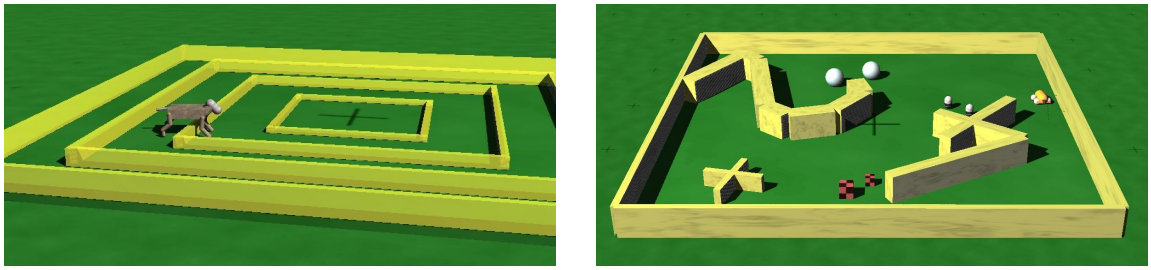


Figure 2.5: Different Environments with obstacles.

2.1.7 Summary

This chapter provided a brief overview of the developed robot simulator, featuring much more than could be highlighted here. Complex virtual robots and environments can be set up in a short amount of time and efficient simulations are possible. We developed a material interaction model that allows the user to specify intuitive material properties and therewith more realistic simulations. The simulator has grown into a large software project with more than 25 000 physical lines of code, not counting the controllers and simulations.

2.2 The *Zoo* of Robotic Creatures

In this section we describe the robots used in multiple places throughout this work. The collection of robots is useful to study different aspects of the homeokinetic principle. Simple robots are required to have a straightforward testbed for new algorithms and more complicated robots allow for the demonstration of various properties and limitations of the homeokinetic control. The following sections serve as a reference and can also be read when required.

Let us define the most important variables describing the sensors and motors of a robot. Both sensor and motor values are real numbers. For a robot with n sensors and m motors we denote the sensor values by the vector $x \in \mathbb{R}^n$ and the motor values by $y \in \mathbb{R}^m$. The latter are defined to be in the interval $[-1, 1]$. The sensor values are not particularly constrained but often lie in the same range. The next section starts with a simple driving robot and then we proceed to more complicated robotic systems.

2.2.1 TwoWheeled Robot

The TwoWheeled robot is a simulated driving robot with a capsule shaped body and – as the name suggests – two wheels, see Fig. 2.6. The wheels are driven by motors, which

Figure 2.6: Pictures of the simulated TwoWheeled robot. The robot has two actuated wheels and two wheel velocity sensors. **(a)** Screenshot taken from a simulation; **(b)** Wire-frame view with the axis of wheel rotation.

can rotate in both directions. The motor signals determine the desired rotational velocity, which is reached within one simulation time step if the required torque does not exceed the maximal configured motor torque. The robot has two sensors measuring the actual wheel velocities. Thus, we have $n = m = 2$. The sensors enable the controller to know when an obstacle was hit, because the wheels would not rotate in this case. A demonstration of the inertial and slip effects are given later in Section 4.8.4. Typical behaviors of this robot are straight driving, when both motors perform the same action, curved driving, when the motors rotate with different velocities, and finally turning at the point, when both motors operate with opposite sign.

2.2.2 FourWheeled Robot

Very similar to the TwoWheeled robot is the FourWheeled robot, which has, not surprisingly, four wheels instead of two. The motors and sensors work in the same way as the ones of the TwoWheeled robot (Section 2.2.1), only that there are four of each kind ($n = m = 4$). Additionally, the robot is equipped with six infrared (IR) sensors as depicted in Fig. 2.7. In reality these sensors actively sense by emitting a beam of light and measure the strength of the reflection. Thus, they measure the distance to a reflective obstacle. In the simulations they simply measure the distance to obstacles directly, so that no differentiation among the materials exists. If no obstacles are within the range of the sensor then its value is 0 and if an obstacle is very close then the sensor value is 1. In between a linear characteristics is used. The operation of the robot is a bit more complicated compared to the TwoWheeled robot, because the wheels on one side have to be driven in a coordinated fashion. The reaction of the body to different combinations of motor actions are given in Section 6.1.4. For normal driving the wheels on the same side are controlled to have the same velocity.

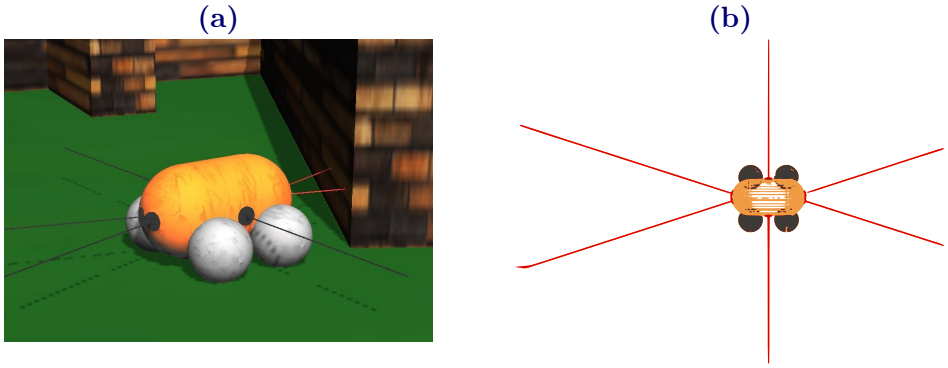


Figure 2.7: Pictures of the simulated FourWheeled robot. The robot has four actuated wheels and four wheel velocity sensors. **(a)** Screenshot taken from a simulation. The IR sensors are drawn for illustration in black if they measure no obstacle and in red of they do; **(b)** Wire-frame view with indicated IR sensor ranges, which are longer at the front (**left**).

2.2.3 Rocking Stamper

The Rocking Stamper is the only real robot constructed by the author. It consists of a bowl-like trunk with a pole mounted on it that is driven by two motors ($m = 2$) in orthogonal directions, see Fig. 2.8 and 2.9. The robot is equipped with two infrared (IR) sensors ($n = 2$) mounted at the front end of the trunk looking down and slightly sideways. They measure the distance to the ground or to the wall, which changes depending on

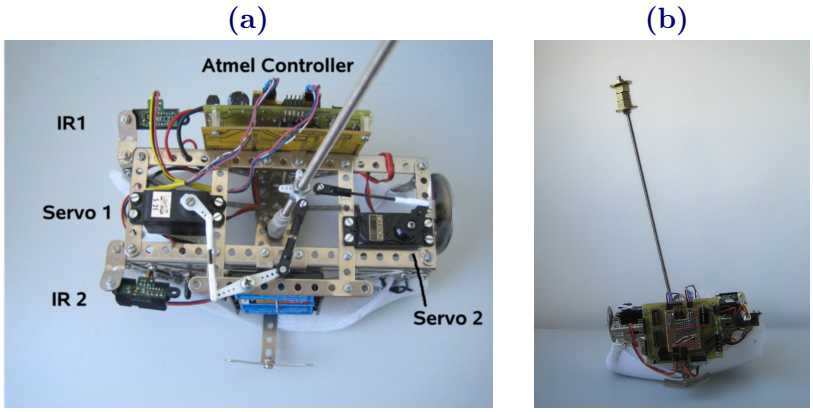


Figure 2.8: The Rocking Stamper, a pole driven bowl-shaped robot. **(a)** Close view from the top; **(b)** Side view.

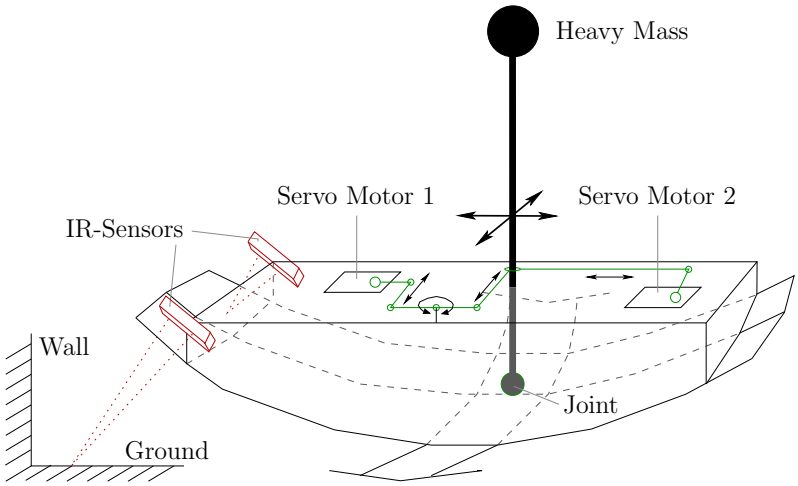


Figure 2.9: Schematic diagram of the Rocking Stamper. The pole is moved relative to the trunk by the two servo motors. This causes the trunk to tilt so that the IR sensors, which are mounted at the front, measure a different distance to the ground/wall.

the pose of the trunk in a nonlinear fashion. The motor commands dictate the nominal positions of the servo motors, which determine the angles of the pole relative to the trunk, as illustrated in Fig. 2.9.

The robot is equipped with an embedded controller board holding an Atmega32 processor and a battery pack, thus the robot can operate autonomously. However, in the current implementation the on-board hardware is only used to establish a connection to a workstation which runs the actual control algorithm. From a performance point of view it is possible to run the following control algorithms on a embedded processor, especially for a two-dimensional system. However, it is easier to test and record data if the control algorithm runs on a workstation.

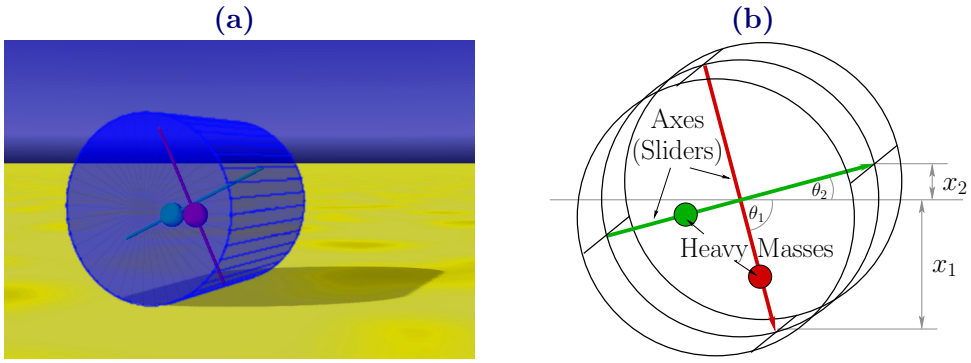


Figure 2.10: Barrel robot. **(a)** Screenshot from a simulation. The red and green masses are moved by actuators along the axes; **(b)** Schematic view of the robot. The sensor values are $x_i = \sin(\theta_i)$. In the illustrated configuration $x_1 < 0$.

The robot can basically rock in two dimensions. The swing in the direction front-back is more easily achieved because the body has a smoother curvature compared to the sideways direction. By suitable combination of both rocking dimensions the robot can perform a locomotion behavior. Another mode of operation occurs if the pole is moved quickly around the center position, such that the massive weight stays almost at a fixed position, while the trunk oscillates.

2.2.4 Barrel Robot

The Barrel robot has a cylindrically shaped body. Inside are two masses that are moved by motors along orthogonal axes, as displayed in Fig. 2.10. The induced change of the center of gravity causes the robot to roll in one direction or the other. The heavy masses have the same weight as the hull. This implies a large inertia, so that a few revolutions are required for acceleration and deceleration. Since the robot cannot turn, it can only move along a line. The two motor values control the nominal positions of the masses along the axes. A value of zero stands for a centered position and -1 and 1 correspond to the outer positions. The position control is done with a PID[3]-controller implementation.

The robot has two axis-orientation sensors which simulate a gravitation sensor ($n = 2$). For each of the two axes the projection of the axis direction onto the z-component of the world-coordinate system is measured. The sensor values can also be defined as $x_i = \sin(\theta_i)$, where θ_i is the angle of the i-th internal axis to the ground plane, see Fig. 2.10(b). If the axis is horizontal the corresponding sensor value is zero, and if the axis is vertical we find a sensor value of -1 or 1. For a normal rolling mode the sensor values perform a harmonic oscillation.

[3]PID stands for proportional-integral-derivative. This is a generic feedback control mechanism to perform a set-point control.

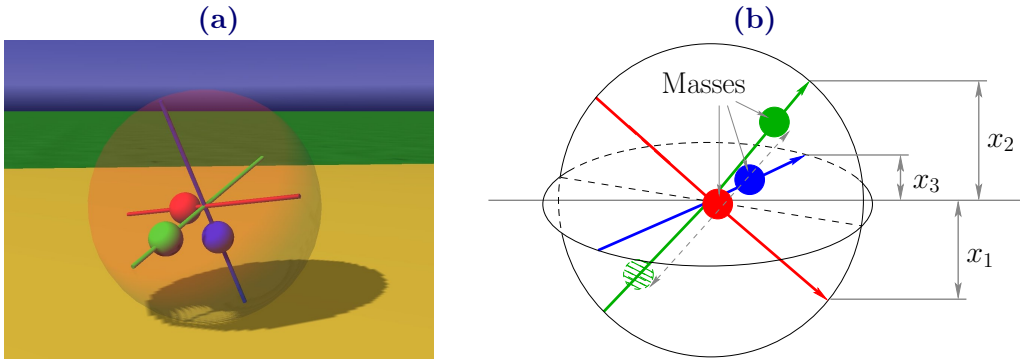


Figure 2.11: Spherical robot with axis-orientation sensors. **(a)** Screenshot from a simulation. The red, green, and blue masses are moved by actuator along the axes; **(b)** Schematic view of the robot with axis-orientation sensors (x_i).

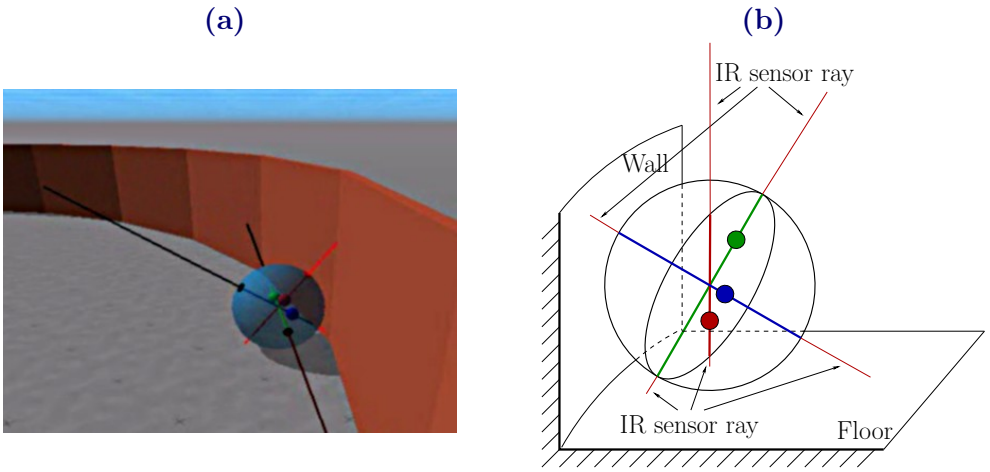


Figure 2.12: Spherical robot with infrared sensors. **(a)** Screenshot from a simulation. The IR sensors are drawn in black if they measure no obstacle and in red of they do; **(b)** Schematic view of the robot with IR sensors.

2.2.5 Spherical Robot

The Spherical robot is a 3D version of the Barrel robot (Section 2.2.4). It has a ball shaped body and is equipped with three internal masses whose positions are controlled by motors ($m = 3$), see Fig. 2.11(a). If not differently stated, then each mass has the same weight as hull alone. The position control of the movable masses is done via servo motors. The motor values define the nominal positions of the masses along the axes. A value of zero stands for a centered position and -1 and 1 correspond to the outer positions.

We use this robot in two different sensor configurations. The first one uses axis-orientation sensors ($n = 3$), see Fig. 2.11. These are the same sensors used with the Barrel robot.

For each axis the projection of its direction onto the z-component of the world-coordinate system is measured. The second setup uses infrared (IR) sensors as depicted in Fig. 2.12(b). In total six sensors are used that extend the internal axes towards the outside of the robot ($n = 6$). Each sensor measures the distance to the ground or to another object. The range of the sensors is the 5 fold of the hull's radius.

2.2.6 Short Circuit

The Short Circuit is not really a robot but rather a dummy construction used to test the algorithms. The motor values are simply mapped to sensor values in a 1-to-1 fashion. More precisely, for an n-dimensional Short Circuit we define $x_i = y_i$, for $i = 1, \ldots, n$. From a controller point of view this construction behaves like a perfect robot when proprioceptive sensors[4] are expected, because the control commands are virtually 100% executed.

2.2.7 Planar Snake Robot

The planar Snake robot is a high-dimensional virtual creature. It consists of a number of segments that are pairwise connected by joints, see Fig. 2.13. Each joint has one degree of freedom that is actuated by a servo motor. A motor value of zero corresponds to a straight configuration and -1 and 1 are associated with the fully deflected positions at $\pm 90°$ off center. The robot is underactuated, meaning the power of the motors is not sufficient to move the joints independently of each other to any position. This effect is especially strong at the joints in the middle of the robot. In order to change the joint angle, both halves of the body have to be moved, which causes too much inertial momentum and friction. Additionally to the motors, each joint is equipped with a position sensor that measures the angle of deflection, see Fig. 2.13(b). These sensors provide a measure of the actual joint configuration, which allows the controller to determine the effects of its actions.

2.2.8 Armband Robot

Let us now consider the Armband robot. The name comes from the German word *Armband* meaning a wristband or bracelet of a watch. This robot consists of a sequence of n flat segments placed in a ring-like configuration, where subsequent segments are connected by hinge joints (in total n). The resulting body has the appearance of a bracelet or chain, see Fig. 2.14(a),(b). Each joint is driven by a servo motor and has a joint-position sensor. The joints have a center position, which is such that the robot is in a perfectly circular configuration, see Fig. 2.14(b) (angle of $2\pi/n$ with respect to a straight positioning). The motor values and sensor values are given in terms of joint angle deviations from the center,

[4] Proprioceptive sensors measure joint angles or positions of body parts.

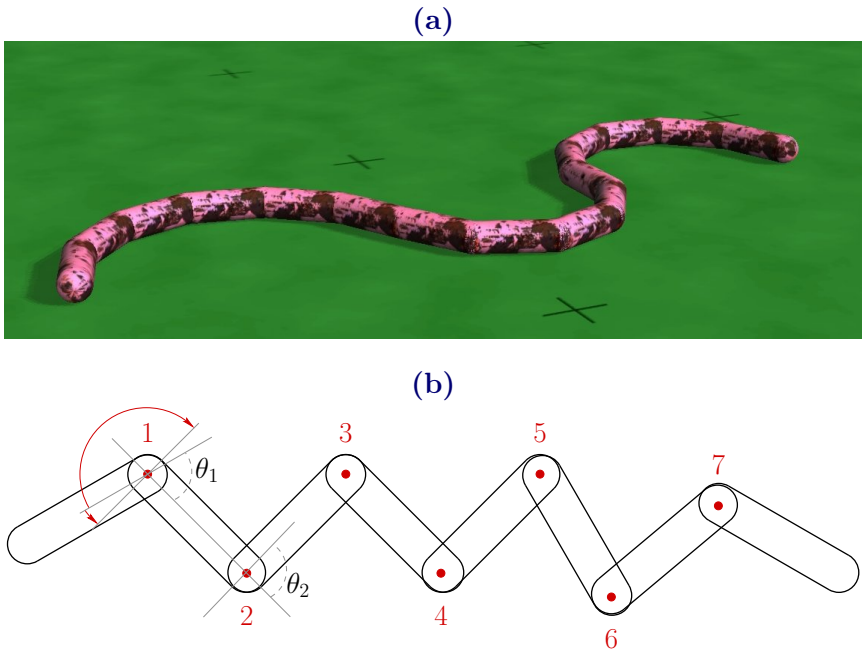


Figure 2.13: Planar Snake robot. The robot has $n + 1$ segments connected by n joints, each equipped with a servo motor and a joint-position sensor. **(a)** Screenshot of a simulation ($n = 15$); **(b)** Schematic diagram with 8 segments ($n = 7$) . The red points mark the positions of the servo motors. Each joint has a position sensor value $x_i = \theta_i$, which is zero for an extended joint. The maximally deflected positions are at $\theta = \pm\pi/2$ (90°).

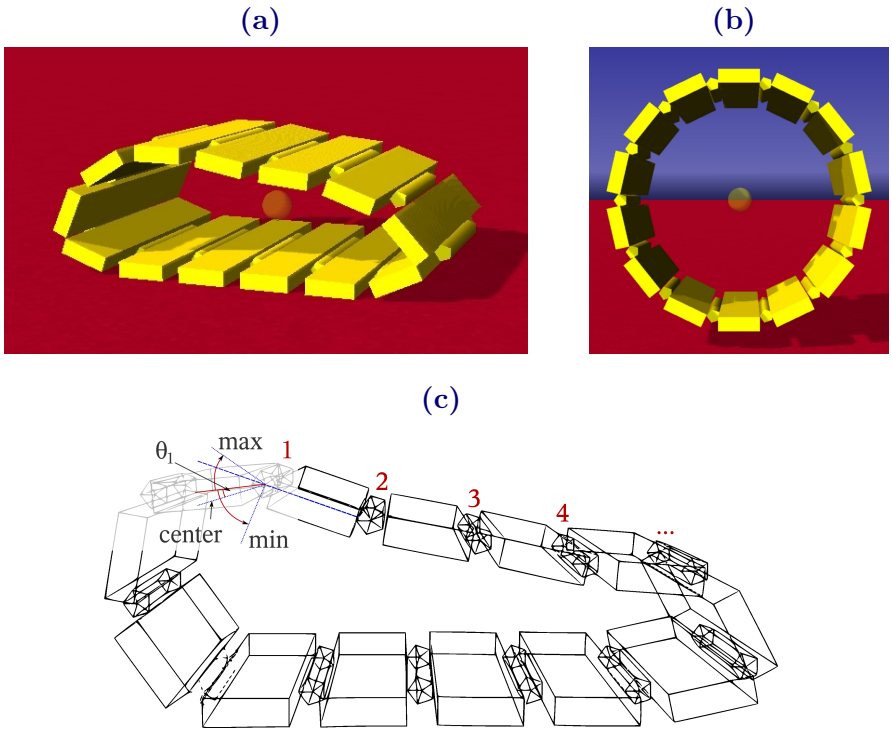


Figure 2.14: Armband robot. **(a,b)** Screenshots from the simulation for $n = 13$. The transparent sphere in the center marks the center of mass of the robot; **(b)** Configuration where all sensors are zero (all joint are at center position); **(c)** Schematic view of the robot. The prismatic structures are hinge joints actuated by servo motors. All joints are equal. The motor values and sensor values are defined in terms of joint deflection angles (θ_i) from the center position (see **(b)**). The values are scaled to the interval $[-1, 1]$. The joint limits are: max= $4/3 \cdot 2\pi/n$, and min=$\pi/3$.

as displayed in Fig. 2.14(c). Note that the joints are highly coupled through the ring configuration. Therefore, an independent movement of a single joint is not possible. Instead it has to be accompanied by a movement of the neighboring joints and of distant joints.

2.2.9 Summary

In this section we introduced a set of robots that differ substantially in shape and mode of operation. The number of degrees of freedom (DoF) ranged from two DoF of the TwoWheeled robot to 16 DoF of the Snake robot.

Chapter 3

Homeokinesis for Robot Control

> Simplicity is not the goal.
> It is the by-product of a good idea
> and modest expectations.
>
> *Paul Rand*

In this chapter we will introduce the concept of homeokinesis [39, 44] and formulate it in mathematical terms. First, we explain the motivation for this approach and describe it in general terms. After that we briefly consider the phenomena of self-organization, as it plays an essential role in the concept. We then proceed with the formulation of the sensorimotor dynamics in the closed-loop setup. For the purpose of illustration we limit ourselves in this chapter to one-dimensional systems. The multi-dimensional case will be the subject of the next chapter. Before we derive the learning rules, we investigate the basic properties of the non-linear control system in terms of fixed point stability and bifurcation behavior. We demonstrate the system dynamics and its adaptive nature in a synthetic example at the end of this chapter.

3.1 Introduction

The control of a robot can be much easier if the embodiment, namely the dynamics of interaction of the body with the environment, is exploited. This requires that the robotic device is controlled in a reactive manner, where motor actions are direct consequences of sensory simulations, instead of the control with a mere sequence of actions [122]. Such a setup is called closed-loop control and allows for the exploitation of the peculiarities of the body-environment interaction. Even with a fixed mapping of stimuli to actions complex behavior can emerge, see [124]. Considering the curiosity and playful behavior of young mammals, there must be an intrinsic motivation to constantly explore one's environment and also the capabilites of one's own body. In animals it is believed that playful behavior

is exhibited because it enhances motor skills. The motivation of homeokinesis is to propose a principle that accounts for the generation of playful and coordinated behavior without specific goals and ultimately achieve a robot that can actually learn something by itself. In general terms the idea of homeokinesis is that the intrinsic motivation or drive to do something is rooted in the requirement to keep the sensorimotor dynamics at the edge-of-chaos – between inactivity and hyperactivity or, as we will see later, between sensitivity and predictability. Since we are aiming at a control strategy for autonomous systems, a fully internal representation of the sensorimotor dynamics is required, as we will present below. Ralf Der proposed not only the abstract concept of homeokinesis [39] but also provided a constructive definition using an energy function, namely the time-loop error (TLE) [44], that allows for an on-line parameter update, as presented below. In this way the dynamical system describing the sensorimotor dynamics is regulated by the internal learning dynamics to a point where spontaneous symmetry breaking sets in and entirely self-driven dynamics emerges. Let us now get an understanding of the widely used term self-organization before we proceed to the formalization of the homeokinetic robot control.

3.2 Self-organization

Self-organization is used to describe the ability of certain non-equilibrium systems to acquire an organized form, e. g. develop specific structures or patterns, in the absence of external control or manipulation. Self-organization phenomena are observed in many complex systems in the fields of physics, chemistry, computer science, economics, and biology [58, 174]. Even though there is a general agreement in what is self-organized and what is not based on visual inspection, there is little agreement on the precise meaning of the word [126]. The field of synergetics provides a general approach to the self-organization of patterns by the study of order parameters and phase transitions in open physical systems far from the thermodynamic equilibrium. Open systems exchange energy or information with the outside world, which is necessary to realize an entropy export, i. e. to generate order within the system. However, when considering general dynamical systems we do not need to model them in an energy consistent way.

Let us consider some examples to get an idea of the effects of self-organization. One of the famous examples in physics is the spontaneous magnetization in ferromagnetic materials [29]. These materials consist of a field of spin states. At high temperatures their orientation is random due to thermal fluctuations, such that the material shows no global magnetization in the absence of an external magnetic field. Below a critical temperature, the Curie temperature, the interaction between neighboring spins becomes import and through spontaneous breaking of the rotational symmetry locally aligned clusters emerge. The size and formation of clusters is self-organizing. The clusters become larger with decreasing temperature such that eventually all spins are aligned.

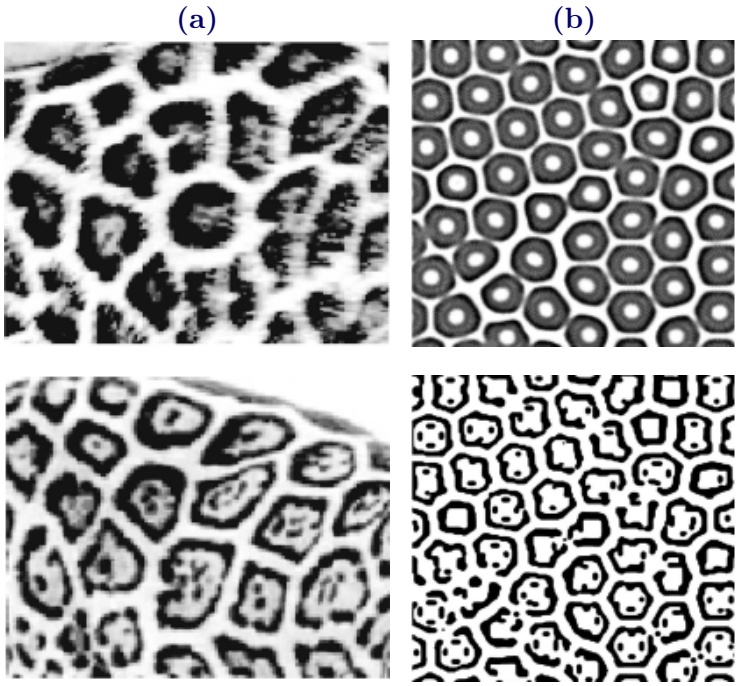


Figure 3.1: Self-organized pattern formation. **(a)** Coat pattern of a jaguar at about 3 month **(top)** and in adulthood **(bottom)**; **(b)** Results of a 2D reaction diffusion system with changed parameters to induce the transient from the upper to the lower pattern (images from [83]).

Another example for self-organization is a reaction-diffusion system. The coupling of local autocatalytic reactions and diffusion can lead to the formation of stationary spatio-temporal patterns as proposed by Alan Turing [166]. Such processes are postulated to cause for the pigmentation of animal coats during morphogenesis. The fur of a jaguar and two different results of a reaction-diffusion system illustrate this similarity in Fig. 3.1.

Another interesting example is how ants find their way to food sources. Experiments showed that only local interaction is required to make the ant colony choose the shortest available food source [38]. Each ant leaves a pheromone trace on its way, whether it searches for food or it returns to the nest. At crossings where several paths intersect they usually choose the direction that is most strongly marked. Ants from nearby food sources will return more quickly, such that those paths are more intensively marked - note the positive feedback. This in turn increases the probability of more ants following the shorter paths, such that the optimal path to the food sources self-organizes.

The above considered systems are composed of many small elements which locally interact with each other and lead to the emergence of a global pattern. However, self-organization can also be found in learning systems. For example, the structure in self-organizing maps [76] is not externally specified but rather emerges in the course of learning. Also here we find a mechanism of positive feedback, namely in the learning rule. For a given

stimulus, not only the best fitting neuron is identified and optimizes its receptive field, but more importantly the neurons in its local neighborhood shift their receptive fields towards the presented stimulus. This leads to a similarity between neighboring neurons such that a more and more structural representation of the set of presented stimuli takes place which eventually results in a topological map.

In all four systems we can identify a control parameter that changes the system behavior qualitatively between an unorganized and self-organized state. The parameter in the magnetization example is the temperature. When decreasing the temperature, a phase transition from the non-magnetized to the magnetized state is provoked and the temperature regulates how large the regions of common magnetization are. In the reaction-diffusion system there is the diffusion constant and the local reaction rate. In the case of the ants the amount of pheromones an ant elicits can be considered as a control parameter. Since the pheromones evaporate over the course of time, a too-weak marking of the tracks does not lead to sufficient self-amplification such that no common path to the nearest food place would emerge. In the example of the self-organizing map, the size of the neighborhood can be considered as a control parameter, which changes the degree of structure in the map. In all examples the control parameter changes the amount of feedback in the system.

The positive feedback gives rise to a bifurcating dynamics of the stable states of the system. To get a more precise grasp of the matter let us consider a dynamical system with the one-dimensional state x:

$$\dot{x} = F(x, \mu)\,, \tag{3.1}$$

where $F : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a smooth function, μ is the control parameter, and $\dot{x}$ denotes the time derivative as usual. This defines a particular dynamical system for a fixed μ, which has fixed points, i. e. points $\{x^* \mid F(x^*) = 0\}$. The fixed points are characterized by their stability which is quantified by the Jacobian (derivative) $\frac{\partial F}{\partial x}(x^*)$. In the vicinity of a *stable* fixed point ($\frac{\partial F}{\partial x}(x^*) < 0$) the trajectories lead to the fixed point. The contrary is true if $\frac{\partial F}{\partial x}(x^*) > 0$ and thus x^* is an *unstable* fixed point. Small deviations from the x^* lead to a further divergence from the fixed point, see e. g. [157]. Changing the control parameter μ can change the position of the fixed point. At certain values μ_c the amount and the quality of the fixed points change, which are called *bifurcation* points as illustrated in Fig. 3.2 for a simple pitch-fork bifurcation. The displayed system at $\mu < \mu_c$ has a single stable fixed point ($x^* = 0$) where the system will eventually end up. If the control parameter is varied from $\mu < \mu_c$ to $\mu > \mu_c$ then small fluctuations, e. g. due to noise, will lead the system state either to the positive or negative stable fixed point as indicated by the arrows in Fig. 3.2.

Besides the self-amplification there is also a need for a limiting force, e. g. a non-linearity, because otherwise the system state would diverge. Consider for example $\dot{x} = \mu x$, which has for $\mu < 0$ a stable fixed point $x^* = 0$, but for $\mu > 0$ the state x diverges to $\pm\infty$. A limiting force is commonly achieved by a non-linearity in the system as we will see later. Note that in natural systems this occurs automatically because of limited resources. In summary, a non-linear system with bifurcating fixed points can exhibit spontaneous

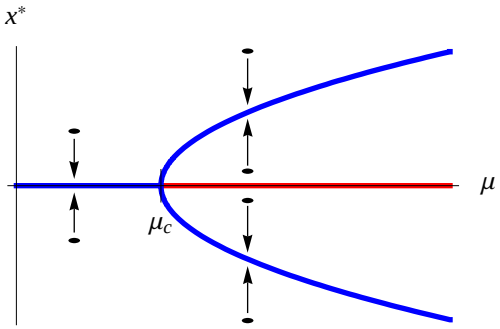


Figure 3.2: Illustration of a pitch-fork bifurcation. Above a critical value μ_C of the parameter μ the stable fixed point $x^* = 0$ is unstable (**red** line) and two stable branches appear (**blue** line).

symmetry breaking and can show emergent phenomena when interacting with many other such systems or with its environment. A single ant does not search for the closest food source, but the interaction of many ants with the environment do. The same applies to the other considered examples, in that only the concerted action of many individual components leads to the global structuring. Note that the underlying dynamical system describing the individual components does not change.

When considering adaptive and learning systems the change of organization is indispensable. The self-organizing map mentioned above is such a learning system and the organization of the system (as a whole) changes indeed. The term self-organizing is also used in conjunction with the self-regulation of the control parameters of a dynamical system to a critical point called *self-organized criticality* (SOC). SOC was introduced by Bak, Tang and Wiesenfeld [9] and is a general concept to explain the observation of spatial and/or temporal scale-invariant characteristic of many natural systems [100]. For example the size of earthquakes, the size of homogeneously magnetized regions, the frequency of words in a text, and the size of neural burst activities in the brain [16, 82] all exhibit a power law statistics. These scale-invariant phenomenon are mostly observed at phase transitions and require a precise tuning of the control parameters. A self-organized critical system self-regulates the control parameters to the critical point. The term self-organization is used in a different meaning here and we will refer to such systems as self-regulating. Nevertheless, the bridge to our understanding of self-organization is that a system at the critical point often exhibits self-organization phenomena.

To conclude, self-organization can occur on a system state level, forming global patterns from local interactions if the control parameters are adjusted appropriately. In high dimensional systems with many control parameters it can be very useful if the dynamical system self-regulates to this working regime. In the forthcoming robot control we will deal with a self-regulating and self-organizing system. The configuration of control parameters will self-regulate under the influence of the interaction of the robot with the external world

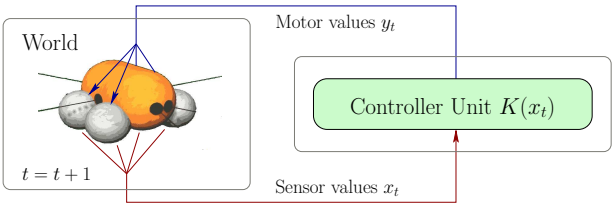


Figure 3.3: Illustration of a robot in the sensorimotor loop.

such that behavioral patterns self-organize. In order to arrive at the point where we can see that let us now specify our setup and the control system in detail.

3.3 Sensorimotor Loop Setup

In this work we consider systems, e. g. robots, in a closed sensorimotor loop with discrete time. This means that at each instance of time $t = 1, 2, \ldots$ we obtain sensor values denoted by $x_t \in \mathbb{R}^n$ which are processed by a control unit that in turn emits motor values denoted by $y_t \in \mathbb{R}^m$. These motor values are executed by the actuators, which possibly effects the environment and the hardware itself. New sensor values are read, now at time $t + 1$, and the loop starts from the beginning. The important point is that the actions are a function of the sensations. More precisely we denote the control function with $K : \mathbb{R}^n \to \mathbb{R}^m$

$$y_t = K(x_t) \tag{3.2}$$

that maps the sensor values to motor values. We formally write the sensation of the robot at time t as

$$x_{t+1} = W(y_t, \ldots, y_1, x_t, \ldots, x_1, t)\,, \tag{3.3}$$

where W is a usually unknown function representing the world. W maps all past motor values and sensor values to sensor values of the next time step. Figure 3.3 shows an illustration of the sensorimotor loop with a wheeled robot.

3.4 Dynamical Systems Formulation of the Sensorimotor Loop

In this section we want to formulate the sensorimotor loop as a dynamical system and illustrate some basic properties of a controller with a single neuron. Let us assume for the moment that the world is known and is Markovian, i. e. without dependence on past values. We can write

$$x_{t+1} = W(y_t, x_t, t)\,. \tag{3.4}$$

Based on the definitions (3.2), (3.4) we can express the sensorimotor loop in a closed form as

$$x_{t+1} = W(K(x_t), x_t, t)\,. \tag{3.5}$$

Since we want to restrict ourselves in this chapter to the one-dimensional case we have i. e. $m = n = 1$. Let the controller K be given by the following function

$$y_t = K(x_t) = \tanh\left(cx_t + h\right)\,, \tag{3.6}$$

that represents a rate-based neuron with hyperbolic tangent activation function, the synaptic connection strength and the bias h. In general, the controller is a one-layer neural network with a weight matrix. Let us use a simplified world that is defined as

$$x_{t+1} = W(y_t, x_t, t) = \alpha y_t + \varsigma_t\,, \tag{3.7}$$

where ς_t is a zero mean noise process. In this example α is a hardware constant. The full system equation reads

$$x_{t+1} = \alpha \tanh\left(cx_t + h\right) + \varsigma_t\,. \tag{3.8}$$

This equation is very similar to the description of a single neuron with an excitatory self-connection in an open-loop setup. The dynamics of such a neuron was investigated in detail in [112, 113, 131]. Nevertheless, let us now find the fixed points of the dynamics and analyze their parameter dependence. For this, we rewrite the equation (3.8) in terms of the membrane potential

$$z_t = cx_t + h\,. \tag{3.9}$$

By neglecting the noise we obtain

$$z_{t+1} = r\tanh(z_t) + h \qquad \text{with } r = c\alpha\ . \tag{3.10}$$

The fixed points are most easily determined graphically, as displayed in Fig. 3.4. However, an analytical solution is also possible using the series expansion of tanh. We used the expansion up to 12th order i. e. $\tanh(z) = z - \frac{z^3}{3} + \frac{2z^5}{15} - \frac{17z^7}{315} + \frac{62z^9}{2835} - \frac{1382z^{11}}{155925} + O\left(z^{13}\right)$. Note that this series converges slowly and the expansion to the 12th order is only accurate for $z \in [-1.5, 1.5]$. It is also important to stop the series at a term with negative coefficient, because otherwise the approximation erroneously produces more fixed points.

The solution of Eq. (3.10) is plotted as a bifurcation diagram for the parameters r and h in Fig. 3.5 which shows a cusp bifurcation [2]. A cusp bifurcation usually appears in systems that are topologically equivalent to

$$\dot{z} = \beta_1 + \beta_2 z - z^3\,. \tag{3.11}$$

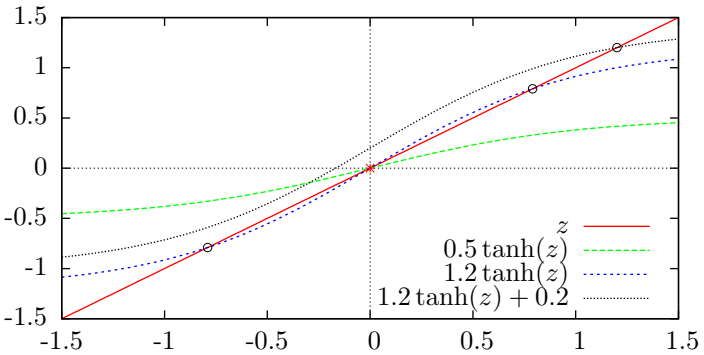


Figure 3.4: Graphical solution for the fixed points of $z = r\tanh(z)+h$, Eq. (3.10) for different values of r and h. For $r = 0.5$ (**green** line) we find only the fixed point at $z = 0$ (stable). For $r = 1.2$ and $h = 0$ (**blue** line) there are two stable fixed points at $z \pm 0.79$ and the fixed point at 0 becomes unstable. If additionally $h = 0.2$ (**black** line), only one fixed point remains at $z \approx 1.2$.

Writing Eq. (3.10) in terms of a differential equation

$$\dot{z} = r\tanh(z) + h - z \tag{3.12}$$

and using only the first two terms of the series expansion of hyperbolic tangent:

$$\tanh(z) = z - \frac{z^3}{3} + O(z^5), \tag{3.13}$$

we obtain

$$\dot{z} = r\left(z - \frac{z^3}{3}\right) - z + h, \tag{3.14}$$

which is equivalent to Eq. (3.11) for $\beta_1 = 3h$ and $\beta_2 = 3(r-1)$. It is important to note that Eq. (3.14) has the same qualitative bifurcation behavior than Eq. (3.12).

Thus, we can draw upon the results obtained for such systems and give the equation for the locations of the saddle-nodes for the approximated system as

$$h = \pm\frac{2}{3}(-1+r)^{3/2}, \; r > 1. \tag{3.15}$$

The saddle-nodes are also seen in Fig. 3.5 at the line where the red and green surface meet.

For better illustration Fig. 3.6 shows different sections of the fixed point structure. The positions of the saddle notes, Eq. (3.15), show the typical cusp wedge, see Fig. 3.6(d). For $r < 1$ the system has only one stable fixed point. We call the system subcritical, since the fixed point is at small z and thus little activity occurs in the sensorimotor loop. For $h = 0$ the system shows a pitchfork bifurcation point at $r = 1$, see Fig. 3.6(a). For $h \neq 0$ two

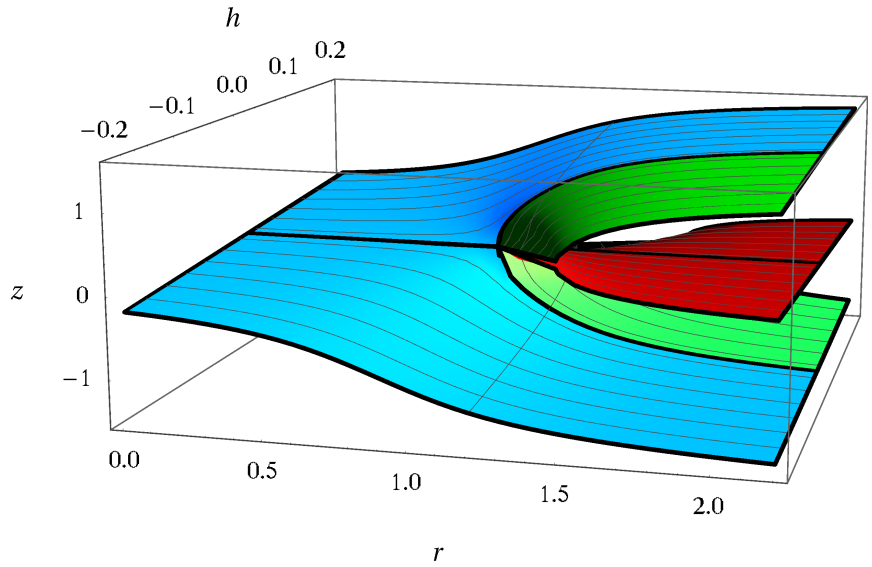


Figure 3.5: Bifurcation diagram for $z = r\tanh(z) + h$, Eq. (3.10), as a surface. Colors: **blue, green** are stable fixed points and **red** is unstable.

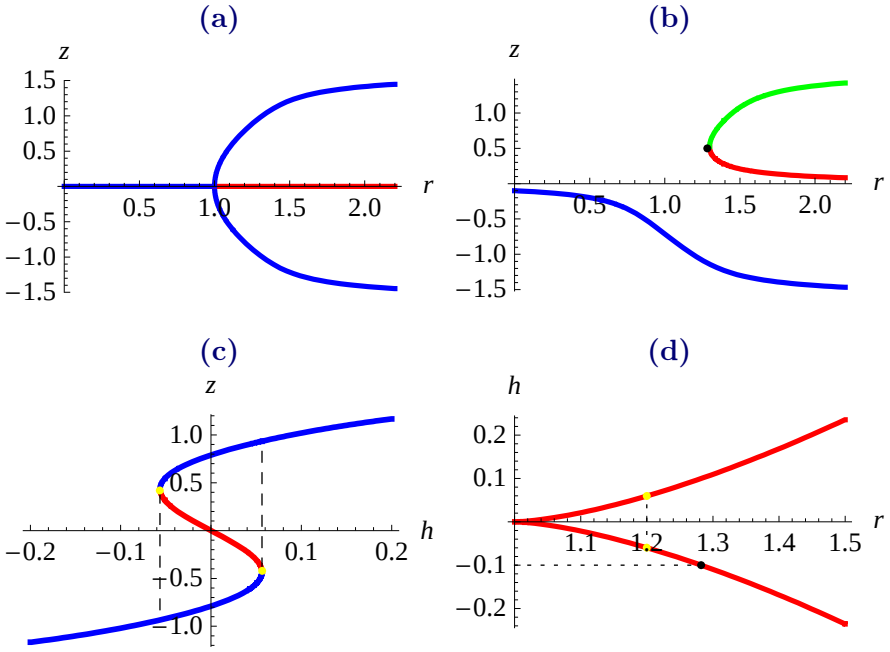


Figure 3.6: Bifurcation diagrams for $z = r\tanh(z) + h$, Eq. (3.10). **(a)** Section at $h = 0$; **(b)** Catastrophic bifurcation at $h = -0.1$; **(c)** The hysteresis in dependence of h in the supercritical regime at $r = 1.2$, also indicated in Fig. 3.5. **(d)** Typical cusp wedge showing the saddle nodes in dependence of r and h, cf. Eq. (3.15). The colored points are correspondingly marked in **(b,c)**. Colors: **blue, green** are stable fixed points and **red** marks unstable fixed points.

branches emerge, see Fig. 3.6(b), and the bifurcation becomes catastrophic. Catastrophic bifurcation means that the system state can undergo a drastic transition for small changes in parameter values. To illustrate the consequences, Fig. 3.6(c) displays the bifurcation diagram for the supercritical parameter $r = 1.2$ in dependence of h and shows a clear hysteresis effect. This means that the system resides in its fixed point when the parameter h is slowly decreased or increased until the fixed point disappears and the state jumps to the fixed point with the opposite sign (dashed lines). When the parameter h is changed in the other direction the same behavior is observed. Hence, for one parameter configuration the system can be in two possible states and h can be used to force a transition.

To summarize, given a simple closed-loop system with a single nonlinear neuron we find non-trivial fixed points and hysteresis depending on the control parameters. This type of dynamics is found in many systems, e. g. in statistical mechanical models of magnets, see [157]. For robot control it seems suitable to have r slightly above the bifurcation point in order to have two stable fixed points which can be changed either by changing h or by external influence. It is also important that the basins of attraction are large enough to avoid noise induced switches. In the case of multiple sensors and motors we get, of course, a much larger number of attractors and limit-cycles, which will be demonstrated by application to robotic hardware later in this work.

3.5 Homeokinetic Principle and Time Loop Error

Given the properties of the control system described in the previous section, the challenge is how to adapt the parameters r and h in order to obtain a certain type of behavior. As discussed in the introduction of this chapter the homeokinesis describes a general principle to self-regulate the parameters so that coordinated behavior emerges. We are not concerned about a specific behavior but rather about a class of behaviors which are similarly explorative, sensitive, and highly body and environment related. Thus far the description of the system involves the unknown function W, cf. Eq. (3.5), and thereby one cannot in general determine r. In order to obtain an entirely internal description of the sensorimotor loop we introduce a so-called *world model*. The name already reflects the purpose: to be a model for the processes that happen in the world perceivable by the sensors. In other words, it is about predicting future sensor values based on present motor values. We make the simplified ansatz and write formally $M : \mathbb{R}^m \to \mathbb{R}^n$,

$$\tilde{x}_{t+1} = M(y_t) \tag{3.16}$$

where $\tilde{x}$ is the predicted sensor value. The difference between the true sensor values and the predicted ones is captured by the modeling error $\xi_{t+1} = x_{t+1} - \tilde{x}_{t+1}$ and we can write

$$x_{t+1} = \psi(x_t) + \xi_{t+1}\,, \tag{3.17}$$

with

$$\psi(x_t) = M(K(x_t)) \tag{3.18}$$

where $\psi(x_t)$ is the internal representation of the sensorimotor loop. The modeling error ξ captures the stochastic component of the sensorimotor loop which we expressed by ς in the simple toy world, cf. Eq. (3.7), e. g. sensory noise and other influences that are not captured by the internal model.

Now, we adapt the system parameter by using the closed dynamical description. In machine learning the adaptation of parameters is often described by a gradient flow on an energy surface also called *error function*, which will be denoted by E in the following. To decrease the error function we use the gradient descent. Thus, any parameter of the system, say p, is adapted by

$$\Delta p = -\epsilon \frac{\partial E}{\partial p}, \tag{3.19}$$

where ϵ is a learning rate. This adaptation will decrease the error at least to a local minimum, assuming the learning rate was chosen sufficiently small. Hence, the main difficulty in designing a controller is to construct an appropriate error function. One suggestive concept is homeostasis. It was introduced by Cannon (1939) and later by Ashby [5] as a general principle to explain the functionality of complex self-organizing biological systems by the maintenance of physiological variables in certain intervals, e. g. the blood pressure, the body temperature and so forth. In an abstract way we may similarly request a minimal prediction error. In this case the error function reads

$$E^{\text{pred}} = |\xi|^2, \tag{3.20}$$

where ξ is the prediction error, cf. Eq. (3.17). Without knowing the details about the concise system, we can already analyze the coarse properties of the resulting behavior by considering the minimum of E^{pred}. The system with minimal E^{pred} will prefer highly predictable situations. These occur for instance when the robot does not move at all, because the world model will quickly learn to predict the static sensor values. There is no need for the robot to move because the system is stabilized over time and fluctuations are suppressed. In the above considered setup there is another solution for infinitely large r, which corresponds to a strong motor output, which is stable against perturbation. Even though the robot would move in this case it is completely insensitive.

This behavior can also be explained by considering the stability to perturbations. The system minimizing E^{pred} damps perturbations with progressing time. This leads to the thought of inverting the time, because then we have a stabilization backwards in time, which is also a destabilization forward in time. In terms of an error function the so-called *time-loop error* (TLE) [44] or *postdiction error* was introduced, which we will derive in the following. To achieve the stabilization backward in time we introduce the quantity v that

describes the necessary change to the sensor inputs to compensate for the prediction error. Hence, v_t is defined such that

$$\psi(x_t + v_t) = \tilde{x}_{t+1} + \xi_{t+1} \tag{3.21}$$

and $\hat{x}_t = x_t + v_t$ is called the virtual input. To calculate v it is necessary to invert the sensorimotor loop function ψ. In practice this can be done with a Taylor expansion to the first order

$$\psi(x_t + v_t) = \tilde{x}_{t+1} + \psi'_x(x_t)v_t + O(v_t^2)\,, \tag{3.22}$$

where ψ'_x denotes the derivative of ψ with respect to x. In the multi-dimensional case the derivative ψ'_x is called Jacobian matrix and will be denoted by L. In the one-dimensional case it is just a scalar number which is given by

$$L_t = \frac{\partial \psi(x_t)}{\partial x_t}\,. \tag{3.23}$$

Using that, we obtain v in a linearized way as

$$v_t = \frac{\xi_{t+1}}{L_t} \tag{3.24}$$

and the time-loop error (TLE) as

$$E_t = |v_t|^2 = \left|\frac{\xi_t}{L_t}\right|^2\,. \tag{3.25}$$

The minimization of the TLE might be phrased as the requirement to minimize the necessary sensor value change that would have reduced the prediction error. The interesting part of this error function is that it is composed of two terms, namely the prediction error and the inverse Jacobian. The latter can be explained more intuitively as a measure of sensitivity. Sensitivity describes how changes in sensory inputs influence the state of the system over time and hence it is expressed by the derivative of the loop function. Since the system is in a closed-loop setup, sensitivity leads to activity (non-zero motor values). The modeling error ξ is a measure of predictability because it captures the misfit between true and predicted sensory information. Figure 3.7 depicts the information flow and the setup of the homeokinetic control system.

Many self-organizing systems have two opposing forces that, in balance, bring the system into the working regime, see Section 3.2. Here, the two forces are:

- the requirement for sensitivity, which results in a driving force and
- the nonlinearities of the controller together with the prediction error as a counteracting force.

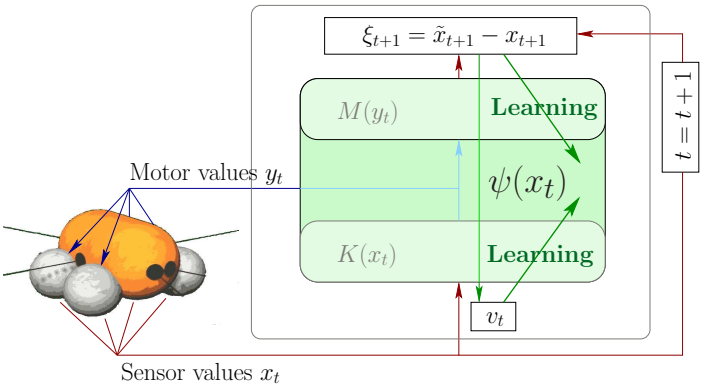


Figure 3.7: Homeokinetic controller in the sensorimotor loop. The time-loop error (TLE) is $E = |v_t|^2$ (Eq. (3.25)). v_t is obtained by backpropagating the modeling error ξ_t through ψ.

Sensitivity and predictability are indeed opposing forces because predictability is best for inactive behavior whereas high sensitivity leads to unpredictable behavior due to the amplification of small perturbations. In order to minimize the error function E, the prediction error has to be small and the sensitivity of the sensorimotor loop large. Since both contradict each other, the resulting robot behavior will be a compromise. Note that the prediction error cannot be zero because we assume a certain sensory noise. Nevertheless, if the prediction error is small (and the behavior is active) then the gradient, Eq. (3.19), is smaller and thus a slower change in the system parameters takes place. In general, well predictable and active behavior will persist for a longer time than inactive or unpredictable behaviors.

3.6 Learning Rule of the Homeokinetic Controller

So far the world model was only introduced in an abstract manner. We will use the linear function

$$M(y_t) = a_t y_t + b_t\,, \tag{3.26}$$

where a_t and b_t are the parameter to be adapted. In the following, the time index on all parameters will be omitted to avoid overly cluttered formulas. The adaptation of the parameters can be performed by a supervised learning technique with training patterns (y_{t-1}, x_t) obtained online. The term supervised learning is sometimes a bit misleading since it does not imply an external teacher, but rather that desired outputs are available for the given inputs. In our case the environment provides the desired values for the world model. A standard learning rule for rate-based neural networks is the so-called delta-rule,

which is the gradient decent on the prediction error E^{pred} (Eq. (3.20)). We obtain the following updates

$$E^{\text{pred}} = |\xi_t|^2 = (x_t - \tilde{x}_t)^2 = (x_t - (ay_{t-1} + b))^2\,, \tag{3.27}$$

$$\Delta a = -\epsilon \frac{\partial E^{\text{pred}}}{\partial a} = \epsilon 2 \xi_t y_{t-1}\,, \tag{3.28}$$

$$\Delta b = -\epsilon \frac{\partial E^{\text{pred}}}{\partial b} = \epsilon 2 \xi_t\,. \tag{3.29}$$

More details on the calculation of such gradients are given later in Section 4.1.4. Now we derive the update rules for the controller parameter in the one-dimensional case. The multidimensional case and some extensions are discussed in the next chapter. For clarity, we will omit the time index also on the state variables, which are taken at time t. From now on the activation function of the controller neuron is denoted by

$$g(z) = \tanh(z)\,. \tag{3.30}$$

Let us recall the system description Eqs. (3.6, 3.18, 3.26)

$$\begin{aligned} K(x) &= g(cx + h)\,, \\ M(y) &= ay + b\,, \\ \psi(x) &= M(K(x)) = a\,(g(cx + h)) + b\,. \end{aligned}$$

We have

$$L = \frac{\partial \psi}{\partial x} = acg'\,, \tag{3.31}$$

where g' is taken at $z = cx_t + h$. We can now calculate the gradient decent on the TLE $E = |\frac{\xi}{L}|^2$ (Eq. (3.25)) as follows (note that we write g instead of $g(z)$, and the same for g' and g'')

$$\begin{aligned} \Delta c &= -\epsilon \frac{\partial E}{\partial c} = -\epsilon 2 \frac{\xi}{L} \left(-\frac{\xi}{L^2} \frac{\partial L}{\partial c} + \frac{1}{L} \frac{\partial \xi}{\partial c} \right)\,, \\ \frac{\partial L}{\partial c} &= ag' + acg'' x\,, \\ \frac{\partial \xi}{\partial c} &= 0\,, \\ \Delta c &= \epsilon \frac{\xi^2}{L^3} \left(ag' + acg'' x \right)\,. \end{aligned}$$

We define a time-dependent learning rate $\tilde{\epsilon}_t = \epsilon \frac{a\xi^2}{L^3} g' > 0$. Using $\tanh'' = -2 \tanh \tanh'$ and $y = g = \tanh$, we obtain the update rule for c as

$$\Delta c = \tilde{\epsilon}_t \left(1 - 2cyx \right) \tag{3.32}$$

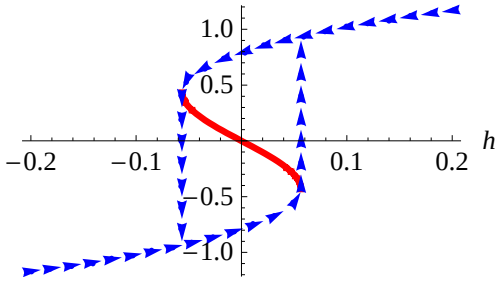


Figure 3.8: Evolution of z plotted in a hysteresis diagram depending on h. The arrows indicate the direction of the h-dynamics. The state of the system oscillates between the stable fixed points. The red line depicts the unstable fixed points.

and similarly for h as

$$\Delta h = \tilde{\epsilon}_t \, (-2cy) \; . \tag{3.33}$$

Inspecting Eq. (3.32) one sees that it consists of a driving term, namely 1, and an anti-hebbian term containing $-yx$, which will moderate the driving term. The update formula for h, Eq. (3.33), reveals that h is driven in the opposite direction of y.

3.7 Fixed Points, Hysteresis, and Self-Switching Dynamics

Let us now consider again the toy world $x_{t+1} = \alpha y_t + \varsigma_t$, Eq. (3.7), and find the fixed points of the learning dynamics. For simplicity, we assume the hardware constant $\alpha = 1$. The world model $M(y_t) = ay_t + b_t$, Eq. (3.26), will have the parameters $a = 1$ and $b = 0$ after successful learning. Setting the right side of Eq. (3.32) to zero gives

$$0 = 1 - 2c \tanh(cx + h)x \, , \tag{3.34}$$

which alone is not enough to get a value for c. However, x has its own fixed point dynamics considered earlier (Eq. (3.8)) namely,

$$x = a \tanh(cx + h) \, . \tag{3.35}$$

For $h = 0$ we get[1] $c \approx 1.19$ and $x \approx \pm 0.648$ and therewith the membrane potential $z = cx + h$ is about ± 0.77. With $a = 1$ we find $r = c$, which is just above the bifurcation point, see Fig. 3.6, where switches between the stable fixed points are possible with moderate external perturbations or small changes in the value of h, see Fig. 3.6(c).

[1] There is another solution for $c < 0$, which is neglected because it corresponds to a dynamics with alternating sign of the motor values each time step.

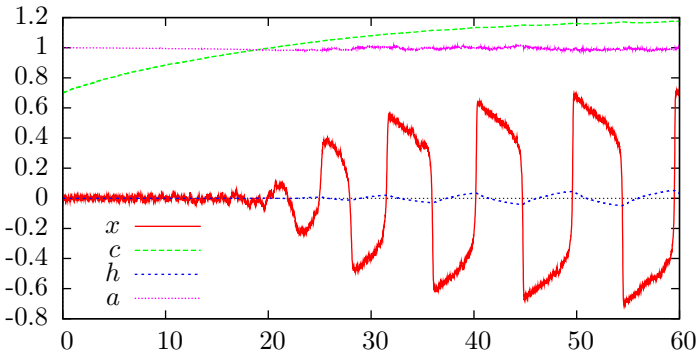


Figure 3.9: State and parameter dynamics in the one-dimensional case. c increases until it reaches its fixed point at 1.19. The bias h oscillates around zero and causes the state x to jump between the positive and negative fixed points. See also Fig. 3.8. Note that $r = ac$ and $z = cx + h$. Parameter: $\varsigma \in (-0.02, 0.02)$, $\epsilon = 1$.

The dynamics of h is counteracting the dynamics of z because the terms c and $\tilde{\epsilon}$ in Eq. (3.33) are positive and $y = g(z) = \tanh(cx + h)$ has the same sign as z. Therefore the following relation holds

$$\operatorname{sgn}(\Delta h) = -\operatorname{sgn}(z)\,. \tag{3.36}$$

Note that the membrane potential z has essentially the same dynamics as x, since $c \approx 1.2$ and $h \ll 1$ in the case considered here. Applying this to the hysteresis diagram for z reveals that the parameter dynamics will cause the state dynamics to cycle along the hysteresis between the two fixed points, as illustrated in Fig. 3.8.

The speed of parameter changes depends on the size of the error. If the prediction error vanishes then the TLE is zero and no parameter change occur. However, in realistic applications we assume a sensory noise, thus the prediction error is always non-zero. In setups where not noise occurs one has to add additional noise to the sensor values.

3.8 System Dynamics in One Dimension

To illustrate this rather complicated interplay between state dynamics and parameter dynamics we plot a trace of the state x and the parameters c, h, and a for the one-dimensional system with $\alpha = 1$ obtained from computer simulation in Fig. 3.9. As expected from the fixed point of the update dynamics, c is seen to increase to a value of around 1.2. The dynamics of h causes the state x to switch between the positive and negative fixed points with the hysteresis clearly pronounced as predicted before, cf. Fig. 3.8. Note that the white noise $\varsigma \in (-0.02, 0.02)$ added to the sensory input is the only cause of errors in this simple setup.

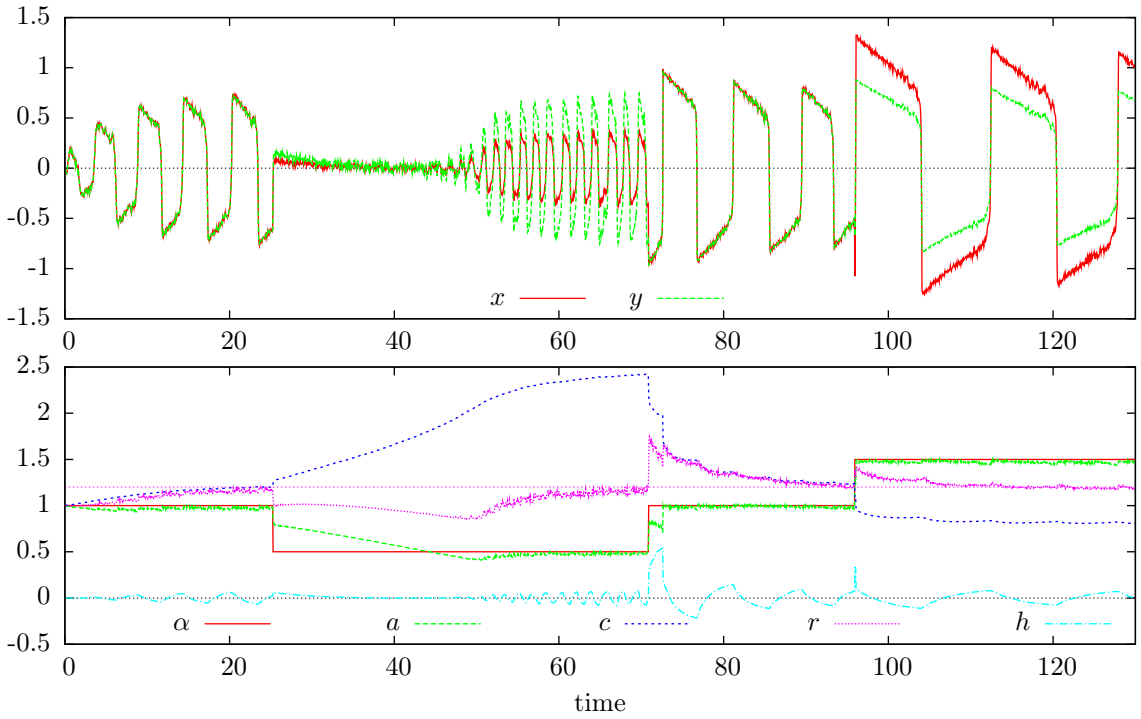


Figure 3.10: Simulation of the one-dimensional system with changing hardware constant α. The world model parameter a and the controller parameter c adapt to the new situation after α was changed at second 25, 70, 95. The linear feedback strength r, however, always recovers to the value around 1.2. The bias h oscillates and causes the state x to jump between the positive and negative fixed points. The oscillation frequency depends on the size of the prediction error, which essentially consists of the sensory noise ς multiplied by c if a converged to α. Parameter: $\varsigma \in (-0.02, 0.02)$, $\epsilon = 1$.

In order to illustrate the adaptivity and self-regulation to the working regime let us now consider a changing world. In Fig. 3.10, the results of a simulation are displayed where the hardware constant α (Eq. (3.7)) was drastically changed several times. The system converges to its working regime after about 20 sec. (Note that c was initialized with 1.0). At second 25 the hardware constant α was suddenly changed to 0.5. The world model a slowly adapts to the new value and the controller parameter c reacts to the change by slowly increasing until twice of the previous value. The adaptation is comparably slow because the system becomes subcritical and therefore the error is only of the order of the noise. The linear feedback strength r, however, is stabilized around 1.2 independent of the value of α, as predicted by the theory. The hardware constant was again changed to $\alpha = 1$ at second 70, and to $\alpha = 1.5$ after further 25 sec. The working regime is again self-organized, but now much quicker because the system is set to a supercritical state. In this state the membrane potential $z = cx + h$ (Eq. (3.9)) becomes large (in its absolute value) such that the neuron is insensitive, i. e. the derivative of the activation function (Fig. 3.4) is small. This region is called *saturation region*. The frequency of switches in the state dynamics

changes for different values of α because the additive noise ς is essentially multiplied by c, which is larger for smaller α.

3.9 Summary

In this chapter we introduced the concept of homeokinesis in a mathematical way. First, we formalized a dynamical system description of the sensorimotor dynamics in a closed-loop setup. Already for the one-dimensional case with a hyperbolic tangent controller neuron we found an interesting dynamics. The state, meaning the sensor value, undergoes a pitchfork bifurcation when the feedback strength in the loop is sufficiently large. Including the second parameter, namely the bias of the controller neuron we find a cusp-bifurcation which brings about a hysteresis effect. To be able to define a learning rule for an autonomous system, that is based on the dynamical properties of the dynamical system a fully internal description is required. Hence, the unknown part of the loop, namely the dynamics of the world, is substituted by an adaptive internal world model. With this we have been able to formulate the time-loop error (TLE) which is an implementation of the concept of homeokinesis. Therefore, the controller is called *homeokinetic controller* in the following. We derived the parameters dynamics for the one-dimensional case and showed the self-regulation to the working regime. This self-regulation is the result of two opposing forces. On the one hand, there is the drive to increase the sensitivity of the loop, meaning an increase of the Lyapunov exponent. This also implies that trivial fixed points like complete inactivity will become unstable and the symmetry in the system will be broken through noise-amplification. On the other hand, there is the requirement to maintain high predictability of future sensations based on current actions. Since chaotic behavior cannot be predicted this acts as a counterforce to the sensitivity. We also find that predictable behavior persists longer than less predictable behavior. In total the system is governed by the interplay between two dynamical systems. The first one is the dynamical system of the sensor values, which is called the state dynamics. The second is the learning dynamics, which changes the parameters of the first systems and is hence called parameter dynamics. In general, a high adaptivity through fast parameter (synaptic) dynamics is used, such that the world model and the controller parameters adapt quickly to new situations, which was demonstrated using a simple toy example.

Chapter 4

Homeokinesis: Multidimensional, Properties and Extensions

> If everything seems under control,
> you're not going fast enough.
>
> *Mario Andretti*

A central point of this work is to analyze, improve and extend the homeokinetic controller. First, we will consider the controller in multidimensional systems. This has been studied before [44, 62], but here we present a uniform view for systems with different numbers of sensors and motors and give a new formulation of the dynamics in motor space, which is particularly suitable for systems with many sensors. Further, we propose essential improvement on the numerical robustness of the algorithm by introducing a couple of regularizations. New robotic platforms that became available through the robot simulator (cf. Section 2.1) enabled us to test the controller in various ways. Some of the robotic platforms required extensions to the original homeokinetic controller in one way or the other, which will be elaborated on as well. Often, however, we could reveal certain problems and demonstrate their solution using simple toy systems allowing for an analytical presentation. In this chapter we also investigate the properties and capabilities of the controller, which are important for robotic applications.

While the first sections are largely mathematical, in later sections we support the theoretical results with experimental evidence, illustrated in plots and with video references. Those who do not plan to follow the mathematical details, can proceed with Section 4.3. In any case the nomenclature and the list of symbols are situated on pages vii and viii and can serve as a useful reference.

4.1 Multi-dimensional Case and Motor Space

In chapter 3 the homeokinetic control algorithm based on the time-loop error (TLE) was presented in the one-dimensional case. In this section, the generalization to the multi-dimensional case will be provided. First we will give a general formulation of the sensorimotor dynamics in sensor space, which is the same as in the one-dimensional case. Then we will introduce a novel formulation of the dynamics in motor space. For both approaches we derive the learning dynamics and discuss aspects of the parameter initialization.

4.1.1 The Time-Loop Error in Sensor Space

Essentially, we consider the same system as in Section 3.3 et seq. but with a multi-dimensional state space. The system is considered again at discrete times $t = 1, 2, 3, \ldots$ with sensor values $x_t \in \mathbb{R}^n$ and motor values $y_t \in \mathbb{R}^m$ where $n, m \in \mathbb{N}^+$. We have a controller function $K : \mathbb{R}^n \to \mathbb{R}^m$ that maps sensor values to motor values

$$y_t = K(x_t)\,, \tag{4.1}$$

and an adaptive world model $M : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that predicts new sensor values $\tilde{x}$

$$\tilde{x}_{t+1} = M(x_t, y_t)\,. \tag{4.2}$$

In contrast to the one-dimensional formulation (Eq. (3.16)) we added the additional dependence on x_t, which will be used later. The prediction error reads

$$\xi_{t+1} = x_{t+1} - \tilde{x}_{t+1}\,. \tag{4.3}$$

Now the dynamical system of the sensorimotor loop $\psi : \mathbb{R}^n \to \mathbb{R}^n$ can be formulated as

$$\psi(x_t) = M(x_t, K(x_t))\,, \tag{4.4}$$

$$x_{t+1} = \psi(x_t) + \xi_{t+1}\,, \tag{4.5}$$

which is a stochastic time discrete dynamical system. Note that K and M are also time dependent due to adaptive parameters. At first we will calculate the terms in a general manner, without considering specific implementation of the controller and model functions.

Analogously to the one-dimensional case, we define the postdiction error v, cf. Eq. (3.21), as

$$x_{t+1} = \psi(x_t) + \xi_{t+1} = \psi(x_t + v_t)\,. \tag{4.6}$$

and we call $\hat{x}_t = x_t + v_t$ the virtual input. To calculate v it is necessary to invert the sensorimotor loop function ψ. For that, we linearize the system by using the first order Taylor expansion, i. e.

$$\psi(x_t + v_t) = \psi(x_t) + \psi'_x(x_t)v_t + O(v_t^2)\,, \tag{4.7}$$

where $\psi'_x(x_t)$ is the partial derivative of ψ with respect to x. The derivative ψ'_x is given by the Jacobian matrix with dimension $n \times n$ (all variables at time t)

$$L_{ij} = \frac{\partial \psi(x_i)}{\partial x_j}. \tag{4.8}$$

Thus we can write

$$\xi_{t+1} = \psi'_x v_t = L_t(x) v_t. \tag{4.9}$$

Using the derivatives of K and F we can also write

$$L_t = M'_y(x_t, y_t) K'_x(x_t) \tag{4.10}$$

in matrix notation. The linear approximation is reasonable as long as we can assume that the prediction error ξ is small. If we assume that L is invertable, we obtain the input shift $v \in \mathbb{R}^n$ and the TLE as

$$v_t = L_t^{-1} \xi_{t+1}, \tag{4.11}$$

$$E_t = |v_t|^2 = v_t^\top v_t = \xi_{t+1}^\top (L_t L_t^\top)^{-1} \xi_{t+1}. \tag{4.12}$$

The structure of the error formula already reveals which kinds of behaviors are favored when E is minimized. Since it is a product of the prediction error ξ and the inverse of the Jacobian matrix L, behaviors with low prediction error and high response strength on all sensory channels are preferred. It also tells us that the eigenvalues of the Jacobian L are increased when decreasing the error. The eigenvalues are a measure of the stability of the system, and their logarithm is known as the local Lyapunov exponents. Hence, the system is driven towards instability, which is counteracted by the non-linearities and by the prediction error. A more detailed discussion of the general idea behind the TLE was presented in Section 3.5.

4.1.2 The Time-Loop Error in Motor Space

So far, the dynamical system was defined in terms of sensory dynamics. To have an invertable system we made the assumption that there are at least as many motors (m) as sensors (n). However, when we think of a robot we realize that most likely the opposite is the case, meaning $m < n$. This is called a bottleneck setup because the sensory dynamics has a bottleneck at the motor values. As suggested by Ralf Der [40], it is advantageous to consider an alternative definition of the sensorimotor loop in motor space, which circumvents the bottleneck and expresses the dynamics in terms of motor values.

The dynamical system in motor space is denoted by $\phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ and reads

$$\phi(x_{t-1}, y_{t-1}) = y_t = K(M(x_{t-1}, y_{t-1}) + \xi_t). \tag{4.13}$$

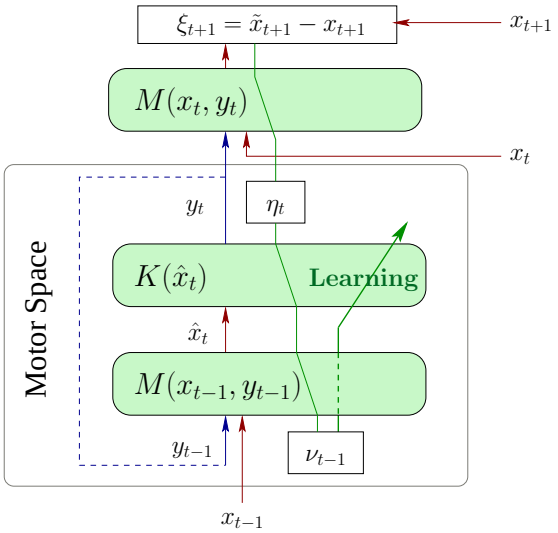


Figure 4.1: Motor space scheme. The prediction error ξ_t is transformed into an error in terms of motor values η_t, which is then used to obtain the input shift ν_{t-1} by inverting the system, see Eq. (4.16).

Note that the system still depends on the sensor values. If the world model only depends on the motor values this dependency can be eliminated. Instead of virtual sensor values which we have derived in the sensor space setting, we will now create virtual motor values denoted by $\nu \in \mathbb{R}^m$. The idea is to first propagate the prediction error through the world model to obtain a mismatch $\eta \in \mathbb{R}^m$ in motor space, i. e.

$$M(x_t, y_t) + \xi_{t+1} = M(x_t, y_t + \eta_t)\,.$$

Using again a linearized inverse we can find η explicitly as

$$\eta_t = {M'_y}^{+} \xi_{t+1}\,, \tag{4.14}$$

where ${M'_y}^{+}$ denotes the pseudoinverse (discussed in the next section) of the derivative of M (which is a matrix) with respect to y. From this we calculate the necessary change ν in motor values of the last time step that would minimize the mismatch η in analogy to the calculations of v above (Eq. (4.6)), hence

$$\begin{aligned} y_t + \eta_t &= \phi(x_{t-1}, y_{t-1}) + \eta_t \\ &= \phi(x_{t-1}, y_{t-1} + \nu_{t-1})\,. \end{aligned}$$

An overview of the setup is given in Fig. 4.1. Analogously to Eq. (4.7) we linearize the system and compute the Jacobian matrix, which is in motor space denoted by $J \in \mathbb{R}^{m \times m}$, as

$$J_t = \phi'_y = K'_x(x_t) M'_y(x_{t-1}, y_{t-1})\,. \tag{4.15}$$

If J is invertible we obtain ν directly as

$$\nu_{t-1} = J_t^{-1}\eta_t\,. \tag{4.16}$$

Hence, the formula for the error reads

$$E_t = \nu_{t-1}^\top \nu_{t-1} = \eta_t^\top \left(J_t J_t^\top\right)^{-1} \eta_t\,. \tag{4.17}$$

The consequence of the error minimization is essentially the same as in the sensor space. One difference is that the prediction error at the motor values η contains the inverse of the world model derivative (Eq. (4.14)). For a low error the world model has to have large enough derivatives along all motor channels. The assumptions that are made for the size of η become quickly invalid if the world model is degenerated.

Considering the dimensionality of J, we find that it is an $m \times m$ matrix, whereas in the sensor space L has the dimensions $n \times n$. Since we assumed that $m \leq n$, the matrix we are to invert has smaller dimensions. Let us also consider the rank of J. For this we have to consider the dimensionality of the derivatives K'_x and M'_y (Eq. (4.15)). K'_x is an $m \times n$ matrix and M'_y is an $n \times m$ matrix. While both multiplied give an $m \times m$ matrix, the intermediate dimension (n) is higher. Additionally, the controller is updated to increase the eigenvalues of J (Eq. (4.17)) so that it will depart from any singularity. Hence, J has full rank if initialized correctly and is therefore invertible. For systems with $m \geq n$ the same holds for L in the sensor space dynamics.

4.1.3 Calculation Rules

Before we can look into the multidimensional version of the learning rules we have to familiarize ourselves with some aspects of matrix calculus. The equations in this section are labeled with (R1, R2, ...). In the later text we refer to them as "rule" with the respective label.

First some simple relations that should serve as a reminder:

$$A^\top B = \left(B^\top A\right)^\top,$$
$$a^\top b = b^\top a\,,$$

where the latter holds because the result is a scalar number.

A less common operation is the row-wise matrix and vector multiplication which we denote with $\circ$. For the application to two vectors we define $\circ : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$ as

$$(a \circ b)_i = a_i b_i \tag{R1}$$

i. e. a componentwise multiplication. Additionally, we define the row-wise multiplication of a matrix with a vector $\circ : \mathbb{R}^{m\times n} \times \mathbb{R}^m \to \mathbb{R}^{m\times n}$ as

$$(A \circ b)_{ij} = A_{ij} b_i\,. \tag{R2}$$

Let us take a look at the properties of this operator. It is associative, commutative and distributive for vectors, Eq. (R1). With matrices, Eq. (R2), it is commutative and distributive with respect to the vector, i. e.

$$\begin{aligned} A \circ b &= b \circ A\,, \\ (A + B) \circ b &= A \circ b + B \circ b\,. \end{aligned}$$

Interestingly the following equality holds as well:

$$(A \circ b)c = (Ac) \circ b\,. \tag{R3}$$

The row-wise multiplication can also be written in terms of normal matrix multiplication as

$$A \circ b = BA \qquad \text{with } B_{ij} = \begin{cases} b_i & \text{if } i = j\,, \\ 0 & \text{otherwise}\,. \end{cases} \tag{R4}$$

For the forthcoming calculations the differentiation of functions with respect to matrices and vectors is required. These functions can most commonly be decomposed into a scalar product of vectors like $a^\top b$. Fortunately, the calculations become much easier after some rules are identified. The differentiation of a square norm

$$\frac{\partial}{\partial X} a^\top a = 2a^\top \frac{\partial}{\partial X} a\,, \tag{R5}$$

follows straight forward from the product rule.

If we assume now that a, b, c are independent of X, the following equations hold:

$$\frac{\partial}{\partial X} a^\top X b = ab^\top\,, \tag{R6}$$

$$\frac{\partial}{\partial X} a^\top g(Xb + c) = (a \circ g')b^\top \qquad \text{with } g' = g'(Xb + c)\,. \tag{R7}$$

When a row-wise product is subject to differentiation then it behaves like a normal product, meaning

$$\frac{\partial A \circ b}{\partial X} = \frac{\partial A}{\partial X} \circ b + A \circ \frac{\partial b}{\partial X} \tag{R8}$$

which follows from Eq. (R4).

Finally the derivative of the inverse of a square matrix is given by

$$\frac{\partial A^{-1}}{\partial X} = -A^{-1} \frac{\partial A}{\partial X} A^{-1}\,. \tag{R9}$$

The derivation of the equations (R6 - R9) can be found in appendix A.1.

Apart from the derivatives we also need the pseudoinverse of possibly rectangular or singular matrices. We use the Moore-Penrose inverse

$$A^+ = \lim_{\delta \to 0} \left(A^\top A + \delta \mathbb{I}\right)^{-1} A^\top .$$

Usually A^+ is obtained by singular value decomposition. However, since in our applications we are only interested in approximate solutions it is sufficient to use a fixed, but small δ. We use two forms of the pseudoinverse

$$A^+ = \left(A^\top A + \delta \mathbb{I}\right)^{-1} A^\top \qquad \text{and} \tag{R10}$$

$$A^+ = A^\top \left(A A^\top + \delta \mathbb{I}\right)^{-1} . \tag{R11}$$

The choice between Eq. (R10) and Eq. (R11) depends on the form of the matrix A. The idea is that the matrix to invert has smaller dimensions, so if A has more columns than rows then Eq. (R10) is used and vice versa.

4.1.4 Pseudo-linear Controller and Linear World Model

In the following we will derive the update rules for the case of a pseudo-linear controller and a linear world model, as it was done in the one-dimensional case in Section 3.6.

The controller $K : \mathbb{R}^n \to \mathbb{R}^m$ is given by

$$K(x_t) = g(C x_t + h), \tag{4.18}$$

where $g : \mathbb{R}^m \to \mathbb{R}^m$, $g(z_i) = \tanh(z_i)$. Note that we always mean componentwise application, when scalar functions (written in lowercase letters) are applied to vectors. The controller can be interpreted as a one-layer neural network in the rate-coding paradigm with the synaptic weight matrix C $(m \times n)$, the bias or offset vector h and the hyperbolic activation function g. The motor values which are generated by the controller lie in the interval $(-1, 1)$.

We use a linear world model, depending solely on the motor value, namely

$$M(x_t, y_t) = A y_t + b, \tag{4.19}$$

where A is an $n \times m$ weight matrix and $b \in \mathbb{R}^n$ is a bias or offset vector. For robotic applications this implies that the sensors are proprioceptive sensors, i. e. sensors that measure internal quantities like joint angles or wheel velocities. Sensors without a direct motor-depend value cannot be modeled with this approach. Later, in Section 4.8, we will consider extended world models.

Let us first calculate the update rules for A and b. The error function is, as before, the square norm of the prediction error $\xi_t = x_t - M(x_{t-1}, y_{t-1})$, i. e.

$$E^{\text{pred}} = \xi_t^\top \xi_t . \tag{4.20}$$

The update for the weight matrix A follows the gradient descent with a small damping as

$$\begin{aligned}\frac{1}{\epsilon_A}\Delta A &= -\frac{\partial \xi_t^\top \xi_t}{\partial A} - \frac{1}{\tau_A}A \\ &= -2\xi_t^\top \frac{\partial\left(x_t - (Ay_{t-1}+b)\right)}{\partial A} - \frac{1}{\tau_A}A && \text{using rule (R5)} \\ &= 2\xi_t y_{t-1}^\top - \frac{1}{\tau_A}A, && \text{using rule (R6).} \end{aligned} \tag{4.21}$$

The damping is used to reduce initial components that are not influenced by the learning dynamics. The timescale τ_A is chosen to be very large, in order to minimize its perturbing effect. For b we find similarly

$$\frac{1}{\epsilon_A}\Delta b = -\frac{\partial \xi_t^\top \xi_t}{\partial b} = 2\xi_t\,. \tag{4.22}$$

4.1.5 Learning Dynamics in Sensor Space

Let us now derive the update rules for the controller parameters C and h in sensor space. By putting Eq. (4.18) and (4.19) into the system equation (4.5), we get:

$$x_{t+1} = A_t\left(g(C_t x_t + h_t)\right) + b_t + \xi_{t+1}\,. \tag{4.23}$$

For the sake of clarity we will omit the time index on all variables during the derivation. In order to minimize the TLE (Eq. (4.12)) the parameters of the controller are to be adapted according to the gradient descent, i. e.

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} = -\frac{\partial v^\top v}{\partial C}\,, \tag{4.24}$$

$$\frac{1}{\epsilon_C}\Delta h = -\frac{\partial E}{\partial h}\,, \tag{4.25}$$

where ϵ_C is the learning rate of the controller, usually between 0.1 and 1 (fast synaptic dynamics). Let us first calculate the Jacobian matrix $L = M'_y K'_x$. The derivative of M is simply given by the matrix A. The derivative of the controller also contains the non-linearities, i. e.

$$K'_x = g' \circ C\,, \tag{4.26}$$

where $g' = g'(Cx + h)$ and hence

$$L = A(g' \circ C)\,. \tag{4.27}$$

Recalling that $v = L^{-1}\xi$, the update for C is

$$\begin{aligned}
\frac{1}{\epsilon_C}\Delta C &= -v^\top \frac{\partial}{\partial C} v && \text{using rule (R5)} \\
&= -\xi^\top {L^{-1}}^\top \left(\frac{\partial L^{-1}}{\partial C}\xi + L^{-1}\frac{\partial \xi}{\partial C} \right) \\
&= -\xi^\top {L^{-1}}^\top L^{-1} \left(-\frac{\partial L}{\partial C} v + \frac{\partial \xi}{\partial C} \right), && \text{using rule (R9)} \qquad (4.28)
\end{aligned}$$

where the factor of 2 is absorbed by ϵ_C. The derivative of the prediction error can not be calculated directly, because we have no information about the actual dependence of the prediction error in the parameters of the controller. This will be the subject of Section 4.7.3. So we are left with the first summand of Eq. (4.28). To shorten the equations and to obtain the scalar product form for further application of calculation rules we introduce the vector $\chi \in \mathbb{R}^m$ as $\chi^\top = \xi^\top (LL^\top)^{-1} A$. Then

$$\begin{aligned}
\frac{1}{\epsilon_C}\Delta C &= \chi^\top \frac{\partial g' \circ C}{\partial C} v && \text{see (4.27)} \\
&= \left(\chi^\top \left(g' \circ \frac{\partial C}{\partial C} + \frac{\partial g'}{\partial C} \circ C \right) v \right) && \text{using rule (R8)} \\
&= \left(\chi^\top \frac{\partial C}{\partial C} v \right) \circ g' + \chi^\top \left((Cv) \circ \frac{\partial g'}{\partial C} \right) && \text{using rule (R3)} \\
&= (\chi v^\top) \circ g' + \chi^\top \left((Cv) \circ \frac{\partial g'}{\partial C} \right). && \text{using rule (R6)}
\end{aligned}$$

As $g = \tanh$ it holds that $g'' = -2gg'$. Recalling that $g' = g'(Cx + h)$ we obtain the update rule as:

$$\frac{1}{\epsilon_C}\Delta C = \left(\chi v^\top\right) \circ g' - 2\left(\chi \circ (Cv) \circ gg'\right) x^\top. \qquad \text{using rule (R7)} \qquad (4.29)$$

Since the formulation of the error in Eq. (4.12) involves future observations we use the error of the previous time step for the actual update, i. e.

$$\frac{1}{\epsilon_C}\Delta C_t = -\frac{\partial E_{t-1}}{\partial C_t}.$$

It is important to note that the only quickly varying variables are x, ξ and therewith v, L, g and g'. The parameters A, b, C, and h change at a slower timescale and are therefore always taken at the current time step (no additional memory for the parameters is required). The full update rule with time indices reads

$$\frac{1}{\epsilon_C}\Delta C_t = \left(\chi_t v_{t-1}^\top\right) \circ g'_{t-1} - 2\left(\chi_t \circ (C_t v_{t-1}) \circ g_{t-1} g'_{t-1}\right) x_{t-1}^\top, \qquad (4.30)$$

where $\chi_t^\top = \xi_t^\top \left(L_{t-1} L_{t-1}^\top\right)^{-1} A_t$ and $v_{t-1} = L_{t-1}^{-1} \xi_t$.

The update for h can be similarly calculated, thus

$$\frac{1}{\epsilon_C} \Delta h_t = -2 \left(\chi_t \circ (C_t v_{t-1}) \circ g_{t-1} g'_{t-1}\right) . \tag{4.31}$$

4.1.6 Learning Dynamics in Motor Space

In this section the learning dynamics for the formulation of the sensorimotor dynamics in motor space will be given. We use the pseudolinear controller, Eq. (4.18) and the linear world model that is restricted to use motor values only, Eq. (4.19). As before, we perform a gradient descent to adapt the parameters. However, the error is now defined by Eq. (4.17) as

$$E_t = \nu_{t-1}^\top \nu_{t-1} = \eta_t^\top \left(J_t J_t^\top\right)^{-1} \eta_t , \tag{4.32}$$

with $\eta_t = {M'_y}^+ \xi_{t+1}$, see Eq. (4.14). The system equation is given by

$$y_t = K(M(x_{t-1}, y_{t-1}) + \xi_t) = g\left(C(A y_{t-1} + b + \xi_t) + h\right) . \tag{4.33}$$

As before, the parameters (C, h, A, b) are taken at time t because they are varying on a slower timescale. Let us first calculate the Jacobian matrix. Using the derivatives of $M'_y = A$ and $K'_x = g' \circ C$ (Eq. (4.26)) we find according to Eq. (4.15)

$$J_t = g'_t \circ C_t A_t = g'_t \circ R_t , \tag{4.34}$$

with $R_t \in \mathbb{R}^{m \times m}$ and $R_t = C_t A_t$. This gives rise to a different formulation for the shift ν as

$$\nu_{t-1} = J_t^{-1} \eta_t = R_t^{-1} \left({g'_t}^{-1} \circ \eta_t\right) = R_t^{-1} \zeta_t , \qquad \text{using rule (R4)} \tag{4.35}$$

with $\zeta_t = {g'_t}^{-1} \circ \eta_t$. Assembling the error function gives

$$E_t = \nu_{t-1}^\top \nu_{t-1} = \zeta_t^\top {R_t^\top}^{-1} R_t^{-1} \zeta_t . \tag{4.36}$$

Now, let us compute the update of the controller matrix C. Omitting the time indices again we get:

$$\begin{aligned} \frac{1}{\epsilon_C} \Delta C &= -\frac{\partial E}{\partial C} = -2\nu^\top \frac{\partial \nu}{\partial C} = -2\nu^\top \left(\frac{\partial R^{-1}}{\partial C} \zeta + R^{-1} \frac{\partial \zeta}{\partial C}\right) \\ &= 2\nu^\top R^{-1} \frac{\partial R}{\partial C} R^{-1} \zeta - 2\nu^\top R^{-1} \frac{\partial \zeta}{\partial C} , \qquad \text{using rule (R9)} . \end{aligned}$$

We again define the vector $\chi \in \mathbb{R}^m$, here as $\chi = {R^{-1}}^\top \nu = \left(RR^\top\right)^{-1} \zeta$ and neglect the factor of 2. Thus,

$$\begin{aligned}\frac{1}{\epsilon_C}\Delta C &= \chi^\top \frac{\partial C}{\partial C} A\nu - \chi^\top \frac{\partial \zeta}{\partial C} \\ &= \chi(A\nu)^\top - \chi^\top \frac{\partial {g'}^{-1} \circ \eta}{\partial C}, \qquad \text{using rule (R6).}\end{aligned}$$

The derivative $\frac{\partial \eta}{\partial C}$ is zero, because we have no information of the dependence of the prediction error on the controller parameters. Using $\frac{\partial {g'_i}^{-1}}{\partial C} = -g''_i {g'_i}^{-2}$ and $g''_i = -2 g_i g'_i$ we find

$$\frac{1}{\epsilon_C}\Delta C = \chi(A\nu)^\top - 2\left(\chi \circ \frac{g}{g'} \circ \eta\right) x^\top, \tag{4.37}$$

where $\frac{g}{g'}$ denotes a componentwise division.

Now we need to assign the right time indices. According to Eq. (4.14) η_t depends on time $t+1$, and therefore the update rule has to be calculated using past inputs. As before, we use the time t for all parameters (A, b, C, h), because they vary on a slower timescale. Hence,

$$\frac{1}{\epsilon_C}\Delta C_t = \chi_{t-1}(A_t \nu_{t-2})^\top - 2\left(\chi_{t-1} \circ \frac{g_{t-2}}{g'_{t-2}} \circ \eta_{t-1}\right) x_{t-2}^\top, \tag{4.38}$$

with $\chi_{t-1} = {R_t^\top}^{-1} \nu_{t-2}$.

The update for h can be similarly calculated,

$$\frac{1}{\epsilon_C}\Delta h_t = -2\left(\chi_{t-1} \circ \frac{g_{t-2}}{g'_{t-2}} \circ \eta_{t-1}\right). \tag{4.39}$$

4.1.7 Initialization

Following the paradigm of tabula-rasa initial conditions we initialize the controller parameters C with small random values. However, one should check whether the sign of the determinant of L (Eq. (4.27)), or J (Eq. (4.34)) is positive, and if not then reinitialize. The reason is that the error E diverges if L or J is singular. The sign of the determinant defines the nature of the bifurcations taking place. If the determinant is negative, the feed-back strength in the sensorimotor loop is driven towards large negative values. Once beyond the flip bifurcation, the signs of the controller outputs are inverted in each time step. This is difficult to realize for the robot and in general of questionable utility. In some of the following examples the small random initialization is used. However, it has the

disadvantage of taking a long time for the system to reach its working regime. For most experiments it is useful to initialize with a scaled unit matrix. Its scaling factor depends on the assumed environmental feedback and is chosen such that the system is subcritical at initialization time. The world model is also initialized with a unit matrix, as it adapts quickly to the observed correspondence.

4.2 Regularization

This section is rather technical and mainly addresses readers who are planing to implement the algorithm. The calculation of the update terms involves a couple of numerically unpleasant parts that need to be treated appropriately in order to obtain a stable algorithm. The difficultly is that we have to solve the differential equation of the weight update ($\dot{C} = -\epsilon \frac{\partial E}{\partial C}$) in a discrete way without the possibility for a step width regulation or other tricks. Moreover, the derivatives of the function E can be very large so that one should either use a small step width or regularize. While a small step width would imply a short cycle time, the sensorimotor loop cannot be driven too fast because *i*) the robotic hardware will only allow for a given computational power and *ii*) inertial effects are too strong. Remember that the executed actions need to show an effect in the sensations in order to be conceivable by the world model. In real world applications the cycle time is also bound by the typical signal traveling time. In most applications we use a update rate of 10 to 100 Hz. In fact, the solution we will pursue uses regularization in two places:

- the inversion of possibly singular matrices and
- the artifacts of linearization.

4.2.1 Pseudoinverse

The Moore-Penrose pseudoinverse was already introduced in Section 4.1.3. It is used to calculate the inverse of the world model matrix since it is possibly rectangular. For reasons of simplicity and performance, we use the formula Eq. (R10) namely

$$A^+ = \left(A^\top A + \delta \mathbb{I}\right)^{-1} A^\top$$

or its counterpart Eq. (R11), which are both approximations. The parameter δ determines the degree of regularization. The eigenvalues of the inverse should be bounded from above for stability reasons. In Fig. 4.2 the size of the largest eigenvalue $\lambda_1^{A^+}$ of the pseudoinverse A^+ is plotted against the size of the smallest eigenvalue λ_n^A of A for different δ, which follow the equation $\lambda_1^{A^+} = \lambda_n^A/(\delta + (\lambda_n^A)^2)$. Note that only real eigenvalues are considered. For our applications we found a value of $\delta = 0.001$ to be appropriate.

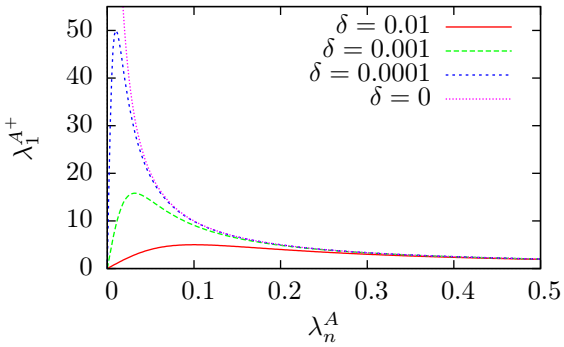


Figure 4.2: Influence of the regularization parameter δ on the eigenvalues of the approximated pseudoinverse.

In the update equations we also find the inverse of symmetric and positive definite matrices such as $LL^\top$. Here we perform a simpler regularization, with

$$\left(LL^\top\right)^+ = \left(LL^\top + \delta\mathbb{I}\right)^{-1}. \tag{4.40}$$

Since the eigenvalues of $\left(LL^\top\right)^+$ behave as $1/(\lambda^{LL^\top} + \delta)$, we use $\delta = 0.05$ to bound their size similarly to above.

4.2.2 Disarm the Non-Linearities

Let us now consider the non-linearities. The only non-linearity in the system is the hyperbolic tangent activation function $g(z) = \tanh(z)$ with its derivatives and inverses. The exact formulas for the first two derivatives are

$$g'(z) = 1 - g^2(z), \tag{4.41}$$
$$g''(z) = -2g(z)g'(z), \tag{4.42}$$

as illustrated in Fig. 4.3(a). We must regularize the first derivative since it is used in the denominator (e. g. Eq. (4.30) with Eq. (4.27)). The second derivative will be regularized for consistency reasons. The most obvious way to regularize the first derivative is to add a small constant $\delta > 0$, similar to the Tikhonov regularization, such as

$$g'_\delta(z) = 1 + \delta - g^2(z). \tag{4.43}$$

For our applications we will use $\delta = 0.02$. The second derivative is automatically regularized when g'_δ is used instead of g' in Eq. (4.42). See Fig. 4.3(b) for both functions.

Using middle value theorem: A more elaborate regularization of the activation function (Eq. (4.41)) uses the middle value theorem. For this we have to consider where the

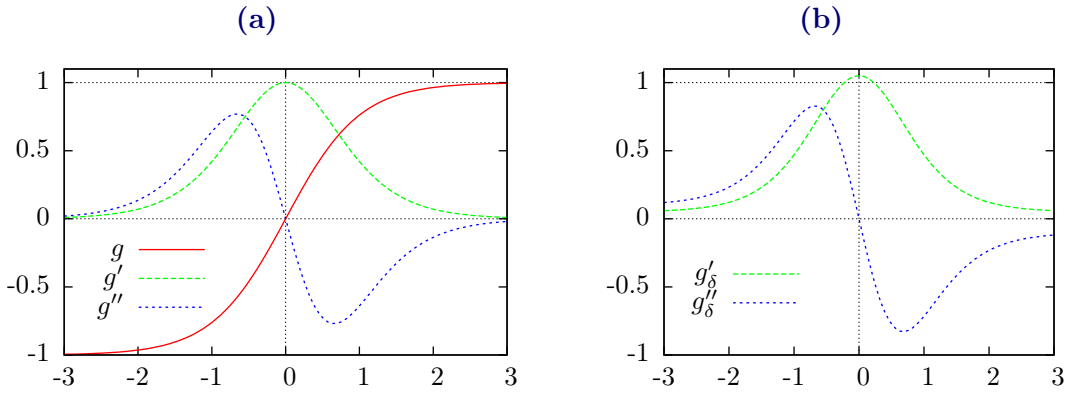


Figure 4.3: Derivatives of hyperbolic tangent with regularized versions. **(a)** activation function $g(z) = \tanh(z)$ and its first two derivatives; **(b)** simple regularized version of the derivatives with $\delta = 0.05$ (chosen larger for visibility).

inverse of the derivative is used. It is used for the virtual inputs v, that are obtained by the inversion of the linearized system using the Jacobian. Let us repeat the one-dimensional case, see Eqs. (3.24, 3.31),

$$v_t = L_t^{-1}\xi_{t+1} = \frac{1}{acg'(z_t)}\xi_{t+1}\,. \tag{4.44}$$

Remember that ξ is the prediction error in sensor space and z is the membrane potential. For simplicity we omit the time index and rewrite Eq. (4.44) with $\zeta = vc$ and $\eta = \frac{1}{a}\xi$ (ζ is now a quantity in terms of membrane potential and η is a quantity in motor space). The resulting (linearized) relation is just

$$\zeta = \frac{1}{g'(z)}\eta\,. \tag{4.45}$$

This linearization is not appropriate in the regions of high curvature of the activation function (g'' in Fig. 4.3). To improve the inversion let us make use of the middle value theorem, which says that if g is continuous and differentiable we can write

$$g(z) + \eta = g(z + \zeta) = g(z) + \zeta g'(Z) \tag{4.46}$$

for a particular $Z \in [z, z + \zeta]$. Thus, ζ can be written as $\zeta = g'(Z)^{-1}\eta$. While with the correct Z the linearization would be exact, let us first consider a crude but efficient approximation using $Z \approx \tilde{Z} = z+\eta$. This can be done since the derivative of the hyperbolic tangent is in the interval $(0, 1]$ and hence $\tilde{Z} \in [z, z + \zeta]$. An illustration of the approach is given in Fig. 4.4.

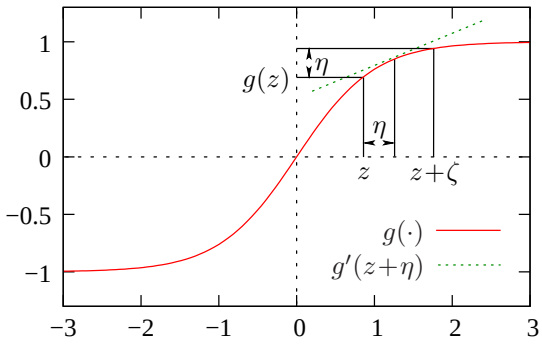


Figure 4.4: Improved linearization of the activation function using middle value theorem. We use $z + \eta$ as a crude approximation for the correct middle value. Hence $g'(z + \eta)$ (**dotted green**) is used instead of $g'(z)$.

In order to avoid a too small derivative we additionally restrict $\tilde{Z}$ to the interval $[-3, 3]$. For that let us introduce the clipping function

$$\Upsilon_{[a,b]}(x) = \begin{cases} a & x < a \\ b & x > b \\ x & \text{otherwise} \end{cases} \tag{4.47}$$

that confines the argument to the interval $[a, b]$. In the formulas where $g'(z)$ was used we now write $g'(\Upsilon_{[-3,3]}(\tilde{Z}))$. Values outside the clipped interval are anyway far outside of the working regime at $z \pm 0.77$ (Section 3.7).

Using exact inverse: Instead of the approximation we can also perform an exact calculation. Remember that we want to obtain ζ in the form

$$\zeta = \hat{g}'(z, \eta)^{-1}\eta, \tag{4.48}$$

where $\hat{g}'(z, \eta) = g'(Z)$ for the correct value of Z that satisfies Eq. (4.46). Using now the inverse of the activation function (remember that $g(z) = \tanh(z)$), namely the area hyperbolic tangent, which is defined as

$$\operatorname{artanh}(x) = \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right) \qquad \text{for } |x| < 1, \tag{4.49}$$

we can calculate ζ directly as

$$\zeta = \operatorname{artanh}(g(z) + \eta) - z, \tag{4.50}$$

see Eq. (4.46). However, this is only valid as long as $g(z) + \eta \in (-1, 1)$, which we will ensure using the clipping function (Eq. (4.47)). To obtain the correct $\hat{g}'(z, \eta)$ for Eq. (4.48) we can simply use the difference quotient and obtain

$$\hat{g}'(z, \eta) = \frac{\eta}{\operatorname{artanh}(\Upsilon_{[-0.999,0.999]}(g(z) + \eta)) - z}. \tag{4.51}$$

All occurrences of $g'(z)$ can be now substituted with $\hat{g}'(z, \eta)$.

4.2.3 Limiting Updates

Even after regularization the updates of the controller parameters can still be comparably large. To avoid extreme changes in one iteration the updates are clipped to the interval $[-0.1, 0.1]$ componentwise. Using the clipping function (Eq. (4.47)) the parameters update for e. g. C becomes

$$C_{t+1} = C_t + \Upsilon_{[-0.1,0.1]}(\Delta C)\,. \tag{4.52}$$

This can safely be done because the clipping only happens in extreme cases. A large change in parameters is nevertheless possible since a consistent change will act on several cycles.

4.2.4 Square Root and Logarithm of the Error

In order to limit the effect of extreme events and to make the algorithm more stable against large prediction errors, we propose to use the square root or even the logarithm of the error function. This can be simply implemented by applying a scalar factor to the normal update, which we take to be $\frac{1}{10\sqrt{E}}$ in the square root case and $\frac{1}{100E}$ in the logarithmic case. The empirically determined factors (1/10, 1/100) are used to achieve a similar learning rate as in the case of the unmodified error for most applications.

4.3 Emergent Embodied Behavior – The Rocking Stamper

After all the theoretical considerations let us turn to an example that demonstrates the performance of the control system when applied to a physical body. What are the general properties of the behavior we expect from the parameter dynamics described above? (cf. Section 4.1.4 et seq.) One of the interesting phenomena observed is the active closing of the sensorimotor loop such that the system is set in motion. Note again that we do not specify a certain task nor provide domain-specific information. However, the generated motion is not just random but comes about from the interplay between internal dynamics and the interaction of the robot with its environment. Through the internal predictive unit – the world model – the sensorimotor dynamics should settle into a continuous and smooth trajectory.

In order to demonstrate this phenomenon we consider here a robotic system called Rocking Stamper, consisting of a bowl-like object with a pole mounted on it that is driven by two motors in orthogonal directions, see Section 2.2.3 for more details. The only sensors are the two infrared sensors mounted at the front end of the trunk looking down and slightly sideways. Their values x_1 and x_2 depend on the distance to the ground or to the wall in a highly nonlinear way. This is because the infrared sensor characteristics is not linear. Also,

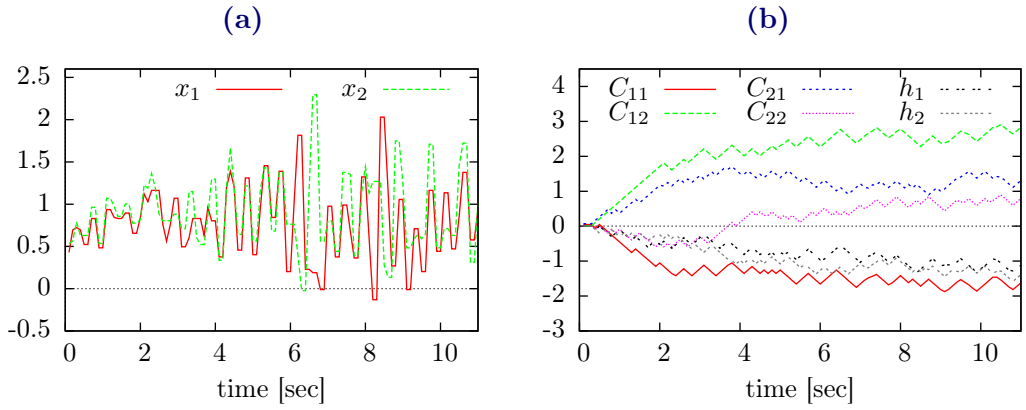


Figure 4.5: Behavior of the Rocking Stamper, from low initialization, represented by sensor readings and controller parameters. **(a)** Sensor values from left (x_1) and right (x_2) infrared sensor. One can see clearly how the controller becomes sensitive. **(b)** Controller parameter values C over time. The sensitization is seen in the increase of $|C_{ij}|$. The controller matrix is adapted to map the difference of both sensors to servo 1 (y_1) and the sum of both sensors to servo 2 (y_2). The bias terms h_i adapt such as to compensate for the positive average of the sensor values.

the robots trunk tilts such that their angle to the ground varies. The controller outputs (y) are the nominal positions of the servo motors which determine the angles of the pole relative to the trunk.

After initialization, we at first have subcritical values for the feed-back strength of the sensorimotor loop so that the influence of the noise is damped and we observe only small fluctuations of the pole position. The learning dynamics increases the values of the controller parameters C and therefore the pole movements become stronger. Eventually the bifurcation point (Fig. 3.6) is reached and an (irregular) oscillatory motion sets in. In Fig. 4.5 the behavior, reflected by the sensor readings, and the parameter adaptation is displayed.

These experiments are interesting in that, despite the extremely nonlinear and nondeterministic behavior of the mechanical system, the controller learns to produce a motion which probes the possibilities of its body in a more or less controlled manner. In Fig. 4.6 the behavior in a later stage of the experiment is shown.

We observed a rocking (oscillatory) as well as a walking like behavior, the latter being caused by a rotational mode of the pole with suitable phase shift. The emergence of these modes is a direct consequence of the sensitization paradigm. In fact, it is in these modes that the controller – based on the current sensor values – can evoke the maximum change in the sensor values over the time step. The modes preferred by the physical systems, e. g. oscillations at the eigenfrequencies, are excited for two reasons. First, the physical system stabilizes against perturbations so that a better prediction is possible and second

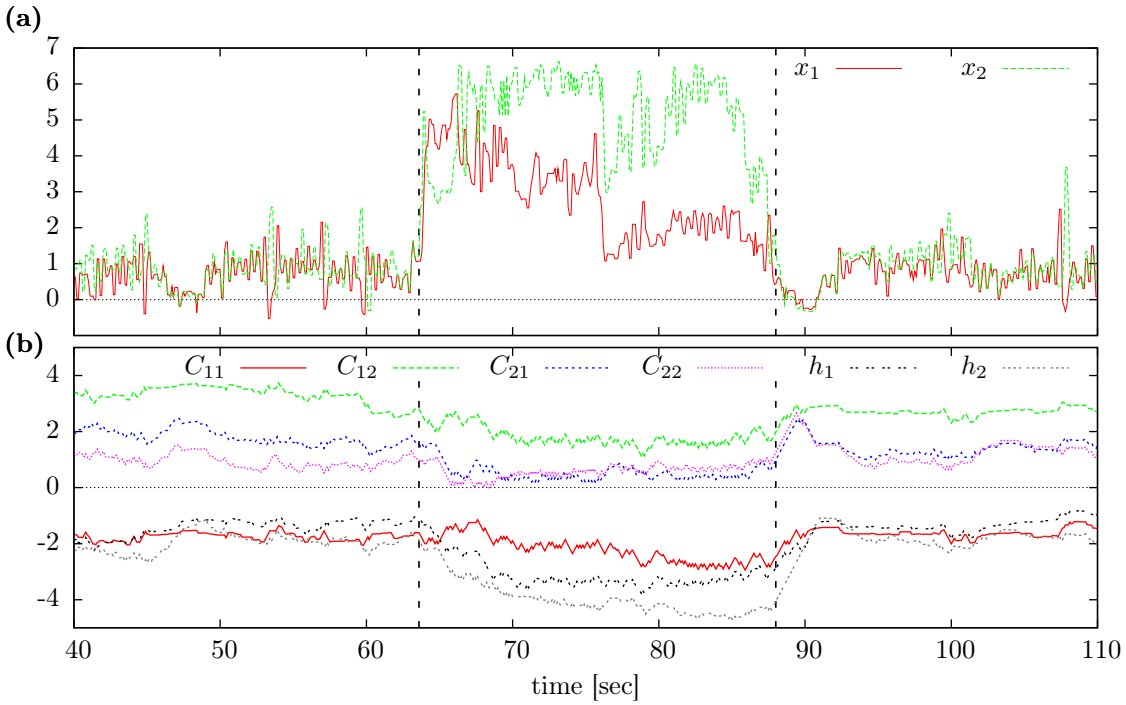


Figure 4.6: Environment sensitive behavior of the Rocking Stamper. **(a)** Sensor values from left and right infrared sensor over time; **(b)** Parameter values over time. Until second 64 we observed rocking (oscillatory) motion with a short break around second 48. Then the robot was manually set into a corner. The infrared sensors measure much shorter distances since they see the walls instead of the ground. At second 88 the robot was pulled back into free space. After each change of the environment the robot was calm for a while (low sensor fluctuation) and probed the new environment, however, after a short time the robot rocked again.

these modes comply with the requirement of high activity. We are tempted to say that the controller develops a "feeling" for the body. In order to demonstrate the environment related nature of the emerging behaviors we put the performing robot into a corner where the infrared sensors measure the distance to the wall, which is much shorter and also has a different characteristics, cf. Fig. 2.9 (p. 23). As a consequence the robot became calm for a short time. Then the parameters adjusted to the new situation, so that again an oscillatory behavior sets in, see Fig. 4.6. The same readaptation scenario occurred when the robot was manually moved away from the corner.

In another experiment we showed that the robot always remains sensitive to its sensors, which can be seen in [Video 1], where the infrared sensor values are changed by the hand of the operator and an immediate reaction is observed. In [Video 2], we demonstrated the quick adaptation to new situations by disabling a sensor or changing the sensor setup during the experiment. In both cases the robot finds back to an active rocking behavior after a short time. A nice sequence of locomotive behavior can be seen in the [Video 3]. The results presented in this section have been published in [43].

4.4 Sweeping Through the Behavior Space

In this section we want to investigate the explorative character of the homeokinetic controller. For this we consider two different robots and study their behavior over time. Before we look at the experiments we want to make a theoretical prediction by considering the error function $E = \xi^\top \left(LL^\top\right)^{-1} \xi$ again. Now the interesting part is the positive definite matrix $LL^\top$. When we consider $L \in \mathrm{SO}(n)$ (special orthogonal group) which means that $L^\top = L^{-1}$ and $\det L = 1$, we find that $LL^\top = \mathbb{I}$. The set of rotation matrices are also element of $\mathrm{SO}(n)$ so that $LL^\top$ is invariant to the rotation angle. A rotation matrix for L causes a oscillation in the state space (sensor values) and the rotation angle decides about the frequency of the oscillations. So in theory the oscillation frequency is not influenced by the learning dynamics. This is not entirely true since L is not a perfect rotation matrix because of non-linearities in the system. Additionally we have the dynamics of the bias h to interfere, so that by way of the non-linearity the gradient descent can change the rotation angle and very interesting phenomena arise. First we will consider a cylindrical robot with two degrees of freedom. The robot exhibits a consequent frequency sweeping effect, meaning that it accelerates and decelerates its rolling motion in a systematic manner. This effect was first discovered by Naglaa Hamed [60] in a simplified toy world and verified by the following experiment, published in [90]. Afterwards, we will consider the Spherical robot with a ball-shaped body and three degrees of freedom. There we observe a change of rotation axes over time. Within the behavioral mode of rotating around one axis we observe a frequency sweep, but less pronounced than in the case of the Barrel robot.

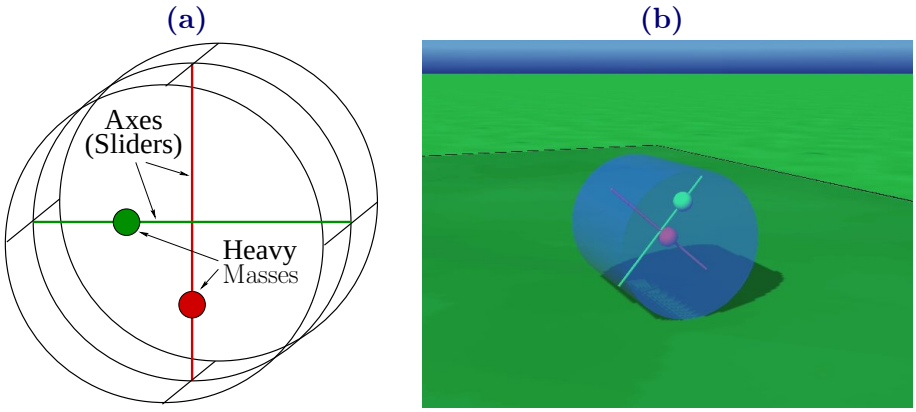


Figure 4.7: Barrel robot. **(a)** Schematic view of the robot; **(b)** Screenshot from a simulation.

4.4.1 Application to the Barrel robot

Let us consider first the Barrel robot as depicted in Fig. 4.7. This robot has a cylindrically shaped body and two internal masses that can be moved along orthogonal axes. The induced change of the center of gravity causes the robot to move. A more detailed description of the robot is given in Section 2.2.4. The Barrel is a (simulated) physical object with strong inertia effects so that it is difficult, for instance, to drive it with a pattern generator emitting a fixed frequency. In this case the Barrel will, for most stimulation frequencies, exhibit a rather inhomogeneous behavior. However, if connected to the homeokinetic controller with both the C and A matrix initialized to a unit matrix, the parameter dynamics will, after a short time, excite a rolling mode with the velocity systematically increasing up to a maximum value. After this the velocity decreases to zero and increases again with inverted sign, cf. [Video 4].

In Fig. 4.8 one can see a part of the state and parameter dynamics of the system for such a cycle. To improve stability we used the square root of the error as described in Section 4.2.4. Note that the velocity of the robot can be directly read from the oscillations of the sensor value x_1, where high frequencies correspond to high velocities. The direction, however, depends on the phase relation between x_1 and x_2 (not shown). We can analyze the matrix R during the course of time. The diagonal elements are $R_{11} \approx R_{22}$, whereas $R_{12} \approx -R_{21}$. This means that R has essentially the structure of a rotation matrix with a scaling factor

$$R = u \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}, \tag{4.53}$$

with different rotation angles θ (see Fig. 4.8). The same holds for A and C. In the experiment, the matrix R runs through the entire range of rotation angles θ and hence through the accessible velocities of the robot. The described sweeping behavior repeats

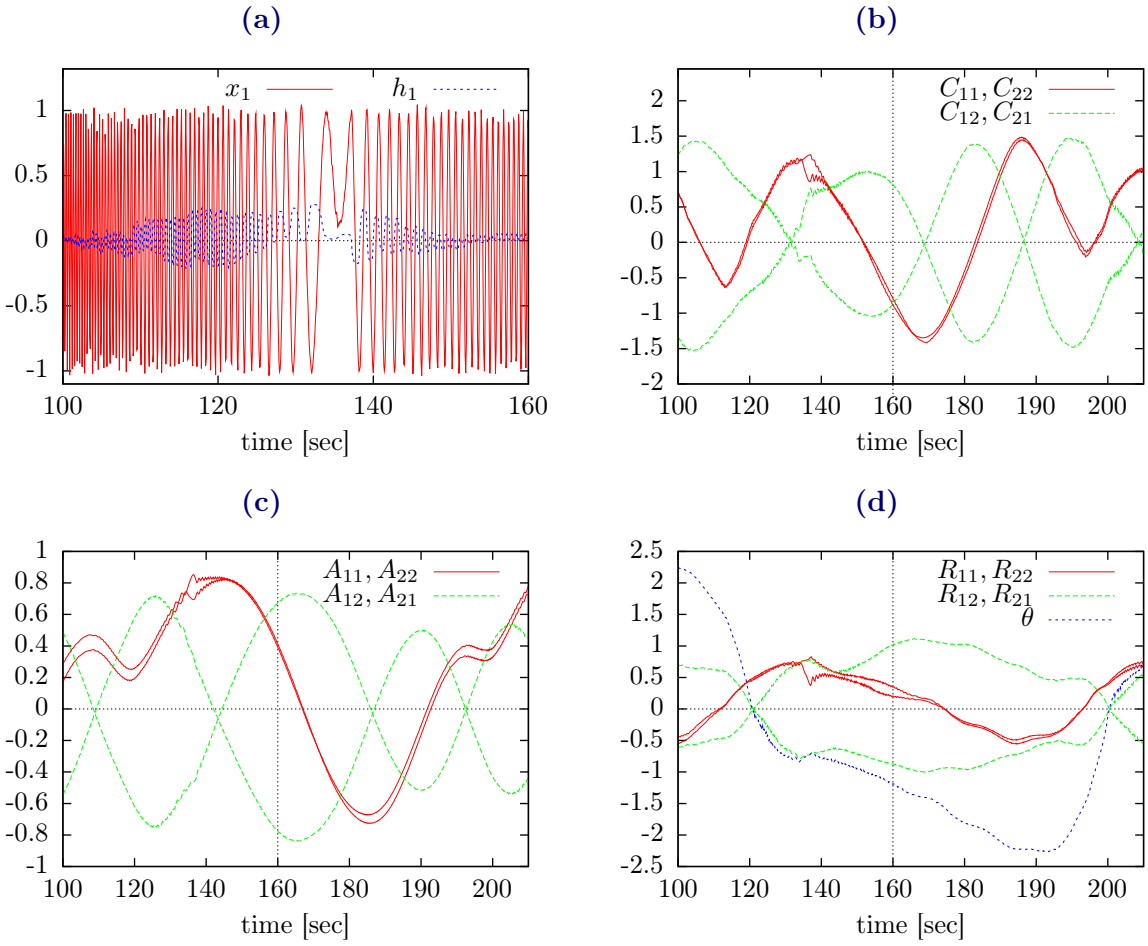


Figure 4.8: Behavior and parameter evolution for the Barrel robot in the time interval 100 to 210 sec. The region up to time 160 covers the period where the robot actively slows down and then inverts its velocity and rolling backwards with increasing speed. After that the speed eventually decreased again and so on. **(a)** sensor value $x_1(t)$ and bias term $h_1(t)$ (for clarity only up to time 160); **(b), (c)** elements of model matrix A and controller matrix C respectively; **(d)** Elements of matrix $R = CA$ and rotation angle θ. Parameters: $\epsilon_C = \epsilon_A = 0.05$, $\sqrt{E}$, update rate 25 Hz.

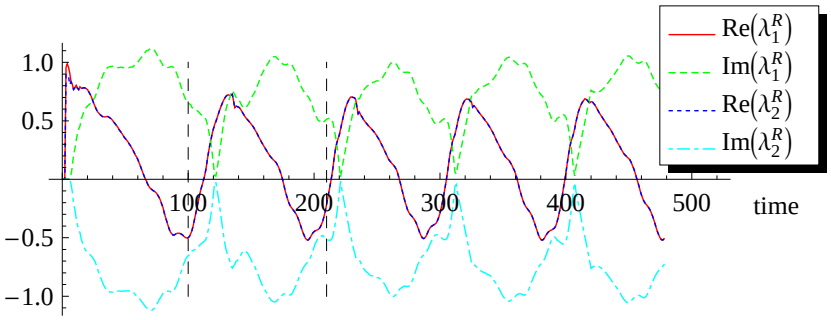


Figure 4.9: Eigenvalues of the controller matrix $R = CA$ during the entire simulation run of the Barrel. The time is measured in seconds. The dashed vertical lines mark the interval used in Fig. 4.8. Several periods of increasing and decreasing oscillation frequency are observed.

more or less periodically as reflected by the eigenvalues of R depicted in Fig. 4.9. What do the eigenvalues λ_i^R tell us? They tell us in general how the system, i. e. the state, will evolve in a qualitative manner. If they are outside of the unit circle ($|\lambda_i^R| > 1$) then the system is unstable, otherwise the system is stable. However, we have to keep in mind that if we consider R alone then we neglect the non-linearities which constrain the expansion. The complex components of the eigenvalues found here indicate that the state of the system performs an oscillation, which was expected from the form of R and is seen in Fig. 4.8(a). The larger the complex components the larger the oscillation frequency. The complex parts of the eigenvalues occur pairwise conjugated. At the zero-crossings of the complex parts the robot is changing direction of motion. Note that the eigenvalues are ordered according to their size, such that they swap roles at the zero-crossings.

One can understand the behavior of the robot partially by considering the term $LL^\top$ in the error function Eq. (4.12). When neglecting the non-linearities this becomes $RR^\top$. If R is a matrix of the SO(n) (special orthogonal) group, then $RR^\top = \mathbb{I}$ (unit matrix) which is invariant against changes in the structure of R. In our case we find that R is a scaled rotation matrix (Eq. (4.53)) so that we obtain $RR^\top = u^2\mathbb{I}$ independent of θ. The scaling is required to compensate for the non-linearities. This, however, brings effects like phase locking which are shortly discussed in Section 4.7.2. It is important to note that the systematic sweeping is a consequence of the interplay between the state dynamics and the learning dynamics, especially of the threshold values h_i cf. [60] for further details.

4.4.2 Application to the Spherical Robot

Let us now consider the Spherical robot on a flat surface. The Spherical robot is a ball shaped robot equipped with three internal masses whose positions are controlled by motors, see Section 2.2.5 for a detailed description. In this setup, the orientation sensors of

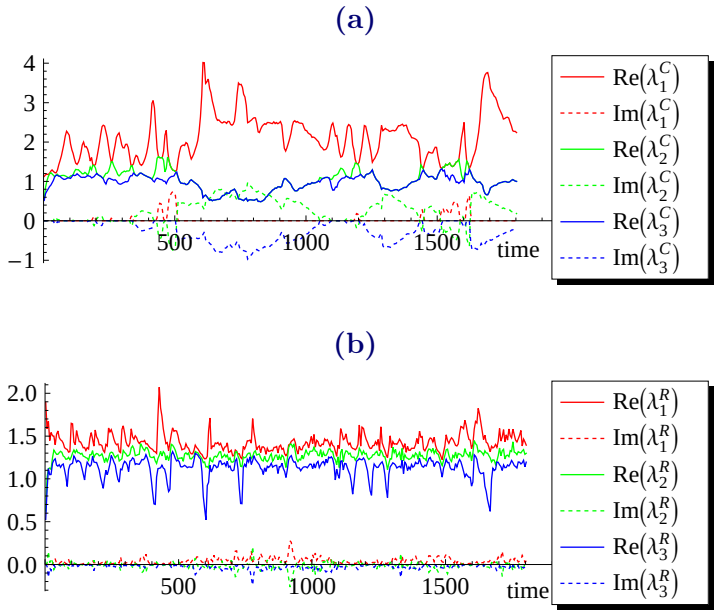


Figure 4.10: Eigenvalues of the controller matrices during a simulation of the Spherical robot on a flat surface. **(a)** Eigenvalues of the controller matrix C; **(b)** Eigenvalues of the matrix $R = CA$. The time is measured in seconds. Parameters: $\epsilon_C = \epsilon_A = 0.3$, $\sqrt{E}$, update rate 100 Hz.

the axes are used, providing three sensor values for each time step. The sensor prediction causes high error values so that we use the square root of the error (Section 4.2.4) to avoid divergence. For a wide range of learning rates the robot will basically perform a continuous rolling motion. The rotation axis and the velocity of the robot are varied over the course of time. However, the velocity does not change in the way it was observed at the Barrel robot. Instead, the system changes the effective subspace of rotation, see also the eigenvalues of the controller matrix C and of the matrix R in Fig. 4.10.

If the Spherical robot rolls around one of its internal axes, then the sensor value for this axis has a little amplitude. Through the gyro effects, the position of the massive weight on the rotation axis has little influence on the belonging sensor value (axis orientation). Instead it will result in a curved movement. Since the reaction of the system to actions (mass movements) along this axis is low, the sensitivity increasing mechanism will increase synaptic weights to this motor neuron and will therefore lead to stronger actions along this axis. This will eventually lead to a change in the rotation axis, as illustrated in Fig. 4.11 and demonstrated in [Video 5]. As an indicating quantity we use the envelope of the sensor values, see Fig. 4.11(a), which is the maximum sensor value reached in a certain sliding window. Thus, they reflect the amplitude of the sensor value oscillations, which is in turn low for a rotation around the respective axis. In the experiment the amplitude varies and there are periods of low amplitude for each sensor, reflecting the changes in rotation axis.

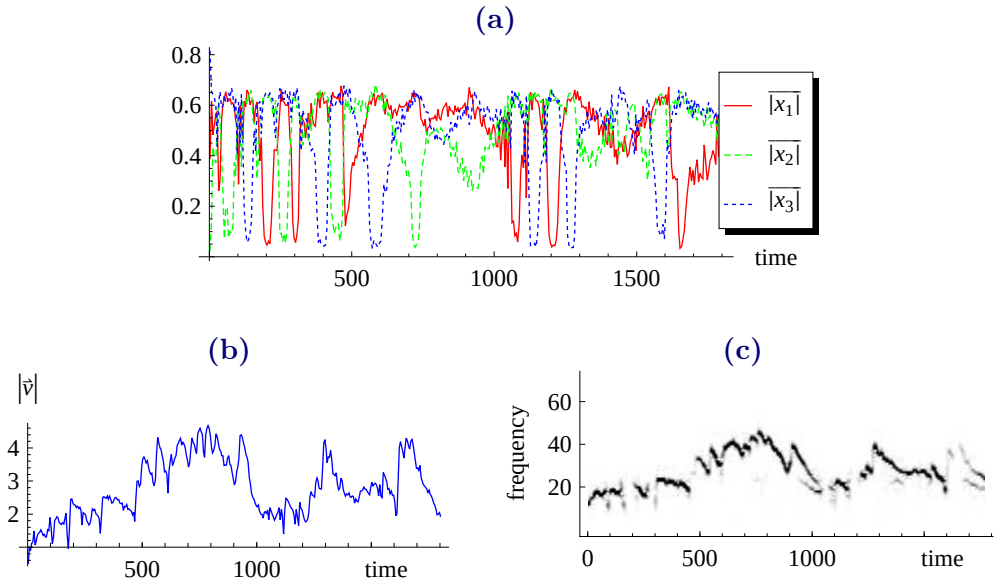


Figure 4.11: Behavior of the Spherical robot on flat surface. **(a)** Positive envelop of the sensor values; **(b)** Velocity of the robot (smoothed); **(c)** Time evolution of the power spectrum (spectral density) of the first sensor value (window size 20 sec, overlap 18 sec). The time is given in seconds. The averaging for **(a,b)** was performed using a sliding window of 5 sec.

In a second experiment, the robot uses six infrared-sensors which are mounted at the surface of the hull at the intersection points with the internal axes. They point perpendicular to the surface of the hull and measure the distance to a wall or to the ground, cf. Section 2.2.5. In this setup we also use a slower update rate without the square root regularization and obtain a qualitatively different behavior. The update dynamics also reaches high-frequency regimes not visited otherwise. The reason is that the rate of change of the controller is proportional to the prediction error. For large errors the controller changes so quickly that the adaptive model cannot follow and thus new regimes are visited. However, after some time these modes are left again. The behavior of the robot is illustrated in Fig. 4.12. To get an idea about the excited oscillations, Fig. 4.12(a) shows a power spectrum[1] of the sensor values over time. High frequency corresponds to high velocity. We can also observe a breakdown in the eigenvalues when very high frequencies are reached and subsequent recovery after some time. This phenomena will be the subject of discussion in the next sections.

[1]The term 'power spectrum' is often used for the 'power spectral density', which is the square of the Fourier transformed signal.

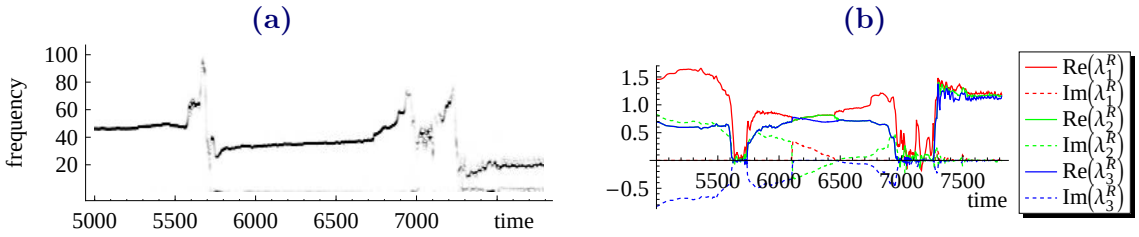


Figure 4.12: Behavior of the Spherical robot with IR-sensors. **(a)** Time evolution of the power spectrum of the first sensor value (window size 20 sec, overlap 18 sec). One can see that a broad range of different frequencies (oscillations) are visited; **(b)** Eigenvalues of the matrix R. The time is given in seconds.
Parameters: $\epsilon_C = \epsilon_A = 0.01$, update rate 25 Hz.

4.5 Cognitive Deprivation and Informative Actions

In this section we want to focus in particular on the consequences of the simultaneous learning of the controller and the internal world model. In an engineering approach, the world model would be trained by trying random actions. However, in high dimensional systems such as complex robots or even humans it cannot be purely random, because of the curse of dimensionality well known from statistical learning theory. It was realized to be a serious problem in learning sensorimotor tasks already by Bernstein [19]. Moreover, usually it is not even necessary to try all actions, but just those that contribute most to the information gain of the model. In artificial intelligence these kinds of actions are called epistemic actions, but the usage of the word is controversial such that we use the term *informative* actions here. Our approach aims at the realization of self-exploration with emergent informative actions instead of motor babbling.

The simultaneous learning of both, the controller and the model, faces among others the so-called cognitive bootstrapping problem or learning paradox [18, 53, 152, 168]. Starting at a "do nothing" and "know nothing" initialization of the controller and the world model respectively, the robot does not have any information about the structure and dynamics of its body, so that the world model has to learn this from scratch. However, in order to learn effectively, the controls have to be informative so that the world model is provided with the sensorimotor patterns necessary for its improvement. On the other hand, these actions require a certain knowledge of the reactions of the body – information is acquired best by informative actions. This bootstrapping situation in principle reappears on all stages of the developmental process. We focus here, as before, on the low-level feedback loops to achieve self-exploration of the physical properties of the body.

4.5.1 Model Learning – Problems and Challenges

Internal models are one of the prerequisites for a robot to become a cognitive system. In the case of the human motor system, the role of internal models has been particularly emphasized by the work of Wolpert [177, 178]. In the present work we are concerned with forward models as given by Eq. (4.19), which are learnt in a supervised way on training samples of the sensorimotor patterns (x_{t+1}, y_t). Remember that the training patterns are obtained from the interaction, such that no external supervisor is required. However, in order to learn the relevant information about the world, the training instances must be guaranteed to sufficiently sample not only the sensor space, but also the action space. In practice it is complicated to ensure this sampling property. In case of on-line learning there is always only a part of the state action space covered in a restricted interval of time. This fact is widely recognized. We will demonstrate a possible solution to the bootstrapping problem using the homeokinetic controller.

4.5.2 Deprivation Effect

Let us first look at how the world model is effected if the controller elicits limited actions. For that we consider a controller restriction, which makes the actions exclude a certain subspace. We will see that the world model gets a limited scope of accuracy and undergoes therewith a process which we call cognitive deprivation. The term "cognitive" is used because the world model reflects a cognitive ability of the robot, namely to perceive and represent the reaction of the world in the current situation.

We consider for now only square matrices for A and C (Eqs. (4.19, 4.18)) and neglect the bias terms h and b. Thus, the world model is given by

$$\hat{x}_{t+1} = A y_t \,. \tag{4.54}$$

Let us assume that the controller matrix C^{restr} has eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ with corresponding normalized eigenvectors u_i, $|u_i| = 1$. We formulate the restriction of the controller by assuming $\lambda_n = \delta \ll 1$, which implies $|C^{\text{restr}} u_n| = \delta$. The controller reads

$$y = \tanh\left(C^{\text{restr}} x_t\right) . \tag{4.55}$$

The restriction implies that y has only small components in the direction of u_n, because $\tanh(z) \approx z$, for $z \ll 1$. Note that we do not assume a particular dynamics here, but we expect the sensor value x to be bounded, e. g. to $(-1, 1)$.

First, we want to prove that the world model will also have this restriction after a certain time of learning, meaning $\lim_{t\to\infty} |A_t u_n| \ll 1$. The prediction error is again

$$\xi_t = x_{t+1} - \hat{x}_{t+1} \,. \tag{4.56}$$

The world model is trained using standard gradient descent on the prediction error $\xi_t^\top \xi_t$, cf. Section 4.1.4. The update of A, see Eq. (4.21), is

$$\Delta A_t = -\epsilon_A \frac{\partial \xi_t^\top \xi_t}{\partial A_t} - \frac{1}{\tau} A_t = \epsilon_A \xi_t y^\top - \frac{1}{\tau} A_t \,, \tag{4.57}$$

with $\epsilon_A < 1$. Note that $\Delta A_t = A_{t+1} - A_t$, thus we can unfold the iteration and find for A_{t+1}

$$A_{t+1} = A_t \sigma + \epsilon_A \xi_t y_t^\top \qquad \sigma = 1 - \frac{1}{\tau} \,, \tag{4.58}$$

$$A_{t+1} = A_0 \sigma^t + \epsilon_A \xi_1 y_0^\top \sigma^{t-1} + \epsilon_A \xi_2 y_1^\top \sigma^{t-2} + \cdots + \epsilon_A \xi_t y_{t-1}^\top \,, \tag{4.59}$$

where $\tau \gg 1$ is a decay time constant. Let us now consider the projection of the world model matrix onto the restricted direction, namely Au_n. Using Eq. (4.59) we can write

$$A_t u_n \approx A_0 u_n \sigma^{t-1} + \epsilon_A \sum_{i=1}^{t} \xi_i y_t^\top u_n \sigma^{t-i} \,, \tag{4.60}$$

where we see that for large times the initial condition vanished:

$$\lim_{t->\infty} A_t u_n \approx \underbrace{A_0 u_n \sigma^{t-1}}_{0} + \epsilon_A \sum_{i=1}^{t} \xi_i y_t^\top u_n \sigma^{t-i} \,. \tag{4.61}$$

Remember that the learning dynamics keeps the system away from the strongly nonlinear regions, so that we can consider the linearized version $y_t = C^{\text{restr}} x_t$ and find that $y_t^\top u_n \approx \delta$ for any time t. Using that $y_t^\top u_n$ is a scalar number we can write the size of the projection $|A_t u_n|$ in the limit as

$$\lim_{t->\infty} |A_t u_n| \approx \epsilon_A \left| \sum_{i=1}^{t} \xi_i \delta \sigma^{t-i} \right| \,. \tag{4.62}$$

Let us now considering the upper bound of $|A_t u_n|$. If we now assume that $|\xi_i| < 1$ (which is reasonable), then we can use for the upper bound $|\xi_i| = 1$. Using the triangle inequality we find

$$\lim_{t->\infty} |A_t u_n| < \epsilon_A \delta \frac{1}{1-\sigma} = \epsilon_A \delta \tau \,. \tag{4.63}$$

From Eq. (4.63) we see that the restriction is transferred to the model matrix A if $\delta \ll \frac{1}{\tau}$. In the upper calculation we made no assumptions about the concise dynamics of the sensor values. Therewith we did not use that the modeling error ξ is usually dependent on C. To get an impression of the real dynamics, the results of simulations with different robots are displayed in Fig. 4.13. The size of the projection $|Au_n|$ approaches zero in an exponential fashion. In all our numerical simulations of restricted systems we found that in the deprived

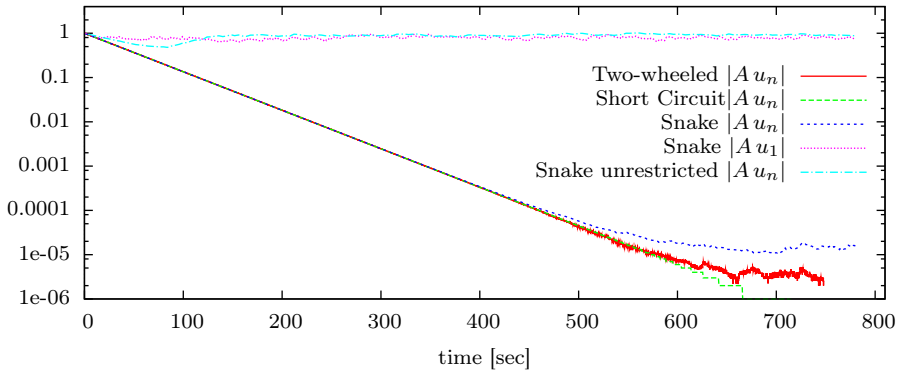


Figure 4.13: Restriction of the world model matrix when a robot is driven by a restricted controller plotted in logarithmic scale. Robots: TwoWheeled – 2 Degrees of Freedom (DoF), Short Circuit – 10 DoF, Snake – 16 segments with 15 DoF. For details on the robots, see Section 2.2. The eigenvector in the nullspace of the controller matrix C is denoted by u_n and the eigenvector belonging to the largest eigenvalue of C is u_1. There is a clear exponential decay for $|A\,u_n|$ expressing the deprivation of the world model, except in the case the controller is not restricted (learned normally) (**dotted blue**). The projections on the largest eigenvalue is not effected by the deprivation (**dashed-dotted cyan**). Parameters: $\tau = 5000, \delta = \lambda_n = 0.0001, \epsilon_A = 0.01$.

situation the eigenvector of the smallest eigenvalue of A is identical to u_n. In other words the kernel of the controller matrix C and of the world model A are identical. However, an analytical proof seems so far impossible.

We have proven that in our setup a controller restriction leads to a deprivation of the world model. More verbosely, if not all dimensions of the action space are covered from time to time, then the world model will become inaccurate for the unvisited subspaces.

4.5.3 The Gradient Flow of the Parameters and Bootstrapping

We have demonstrated that, if the controller and therewith the actions y are restricted in the sense that a certain subspace is excluded, the model will degenerate in that subspace as well. This has the effect that the model might be completely wrong in this subspace. The challenge for the controller is to find out this deprivation of the world model and to issue motor commands which provide the world model with the sensor-action pairs (x, y) necessary for learning the subspace neglected so far.

This is exactly what happens in our approach for the learning of the controller. It is immediately seen when considering the gradient flow of the parameters. Remember that

the adaptation of the controller parameters is realized by the gradient descent of the error function $E = \xi^\top (LL^\top)^{-1}\xi$, i. e.

$$\Delta C = -\epsilon_C \frac{\partial}{\partial C} E\,. \tag{4.64}$$

The resulting dynamics of the parameters can, at a formal level, be argued to produce the desired properties of the system. In fact, since $LL^\top$ is symmetric we may decompose it as

$$LL^\top = \sum_{i=1}^{n} \lambda_i^2 P_i\,,$$

where $P_i = u_i u_i^\top$ is the projector on the space of the eigenvector u_i with λ_i being the corresponding eigenvalue. Then $\left(LL^\top\right)^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i^2} P_i$ and

$$E = \sum_i \frac{1}{\lambda_i^2} \tilde{\xi}_i^\top \tilde{\xi}_i\,, \tag{4.65}$$

with $\tilde{\xi}_i = u_i^\top \xi$ is the projection of the model error into the subspace spanned by u_i. Both λ_i and P_i depend on the parameters C of the controller in an intricate way. This expression is only valid if the $n \times n$ matrix $Q = LL^\top$ is of full rank, such that none of the λ_i is equal to zero. However, if we start with an L not of full rank, the parameter dynamics will drive Q away from impending singularities due to the divergence of E for any $\lambda_i \to 0$.

In more detail, writing Eq. (4.64) using Eq. (4.65) we get

$$\epsilon_A^{-1} \Delta C = \sum_{i=1}^{n} \left(\frac{\tilde{\xi}_i^\top \tilde{\xi}_i}{\lambda_i} \frac{\partial \lambda_i}{\partial C} - \tilde{\xi}_i^\top \frac{\partial \tilde{\xi}_i}{\partial C} \right) \lambda_i^{-2} \tag{4.66}$$

and we see that the gradient flow is driven by two objectives. The first term in the sum obviously tends to increase each of the eigenvalues λ_i and hence increases the instability in the corresponding subspace. The interesting point is in the pre-factor $|\tilde{\xi}_i|^2/\lambda_i$, which implies that the update is strong where λ_i is small (high stability) and $|\tilde{\xi}_i|^2$ is large (high modeling error component in this subspace). This can be interpreted as the tendency of the parameter dynamics to produce in all directions the same degree of instability, with subspaces of higher modeling error being destabilized even more strongly. Exploration corresponds to a higher rate of noise amplification, such that one may say that subspaces, which are less well represented by the model, are explored more intensively.

The second term in sum of Eq. (4.66) essentially counteracts the overshooting destabilization of large error subspaces caused by the first term. The error components $\tilde{\xi}_i$ not only depend on the quality of the model, but depend also in an essential way on the behavior of the robot. Hence, both the $\tilde{\xi}_i$ and eigenvalues λ_i change with changing parameters.

For the restricted direction the factor of the first term of Eq. (4.66) is equal to $|\tilde{\xi}_n|^2\lambda_n^{-3}$. Because there is some noise in the world $|\tilde{\xi}_n|$ is non-zero, so that we essentially have

$$\Delta C^{\text{restr}} \sim \frac{1}{\lambda_n^3}\frac{\partial\lambda_n}{\partial C^{\text{restr}}}\,. \tag{4.67}$$

This implies that C^{restr} will be changed such that λ_n increases drastically as long as the model is wrong. This in turn gives rise to actions along u_n ($y^\top u_n \gg 0$) and training examples for the world model are provided.

We may interpret this by saying that the controller tries more and more actions which force the model to learn also the behavior in the unexplored subspace. Thus, these actions are informative, because they feed the model with the necessary input-output pairs for complete learning. The process is started by some fluctuation in the output of the controller which may be called motor babbling. The exploration of the deprived subspace will start with small actions and if the physical system is not restricted in the direction of the action then the prediction errors will raise and larger explorative steps will be taken. If, however, the controlled system is indeed restricted then the prediction errors will stay low and the explorative actions will stay small. In this way physical restrictions are tolerated.

4.5.4 Application to the TwoWheeled Robot

We are now going to evaluate the theory by controlling a robot. First, with a restricted controller, such that motor commands within a subspace of the actuator space are generated. Then we activate the learning to observe the informative actions. For the experiment we used the TwoWheeled robot with two sensors and two motors as described in Section 2.2.1. The restricted controller is given by

$$C = C^{\text{restr}} = 1.1p^\top p + \delta\,\mathbb{I} \qquad \text{with } p = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} \tag{4.68}$$

and $\delta = 0.001$. The regularization with $\delta\,\mathbb{I}$ is used to avoid a singularity of L, cf. Section 4.1.7. C is a projector into the space spanned by p, hence both wheels receive the same input and the robot will perform only straight forward and backward motion. The factor 1.1 was used to compensate for the non-linearities introduced by the activation function g. The controller is at the beginning of the experiment not subject to the learning dynamics, i. e. $\epsilon_C = 0$. Thus, the bias h was modulated with a sine, so that the robot drives straight forward and backward. As expected, we observe a degeneration of the world model, see Fig. 4.14, which more or less converged to a singularity. Then the learning of the controller was activated by setting $\epsilon_C = 0.01$ and the activity in motor values goes down for a short time, because the sine-modulation of the bias is absent and thus the feedback strength is subcritical. A quick recovery of the parameters is observed and the robot starts to move again. It is important to note that now the controller issues motor commands

which are mainly in the orthogonal subspace, meaning rotational. This is seen at the opposite sign of the two motor values y_1 and y_2, see Fig. 4.14(c). This exploration continues for about 40 sec until the world model is converged to the true relationship and therewith both straight and rotational modes occur with about equal probability. The behavior and the parameter dynamics is displayed in Fig. 4.14 and can be seen in [Video 6].

4.5.5 Spherical Robot in a Basin

Let us now consider a more complicated robot, namely the Spherical robot as introduced in Section 2.2.5. The sensor values of the robot are the projections of the unit vectors in the body coordinate system onto the z-axis of the world coordinate system. The motor commands are the nominal position of the mass points on the inner axes. An interesting effect is produced if we put the Spherical robot into a circular basin as depicted in Fig. 4.15. After an initial period the robot starts to roll back and forth in the basin. The robot then performs rolling motions along a fixed height of the basin. The rolling motions are well predictable by the model, since the sensor and motor values perform a harmonic oscillation. If the robot is rolling at a fixed height then the dynamics remains constant, such that these behaviors do not cause large prediction errors. As a consequence we find that these modes are particularly stable and performed for a long time. In these periods we observe a deprivation of the world model. In Fig. 4.16, the behavior of the robot is characterized using the power spectrum of the sensor values and the smallest eigenvalue of the world model. The initial phase is of the same nature as on the flat surface, i. e. from time 0 – 80 one can see self-explorational modes, where different frequencies are probed. Then a stable rotational mode emerged (time 80 – 110 and 140 – 170), which is the circulation in the basin at a constant height, exhibited over many laps. The circulation mode is seen in the power spectrum at the low frequency excitation. The high frequency excitations are the movements of the axes of the robot due to the rolling and precession motion. One can see that the value of the smallest eigenvalue λ_3^A of A decreases while the robot stays in one mode of behavior, cf. Fig. 4.16(a). The smallest eigenvalue is a measure for the degeneracy of the world model, which deprives due to the restriction to a specific mode of behavior. At time 115 the world model matrix is close to a singularity and the bootstrapping of new actions sets in. This explorational period effectively explores the orthogonal subspace such that the smallest eigenvalue of A is seen to increase rapidly. The same starts again at time 140 and so forth.

In summary, the behavior of the Spherical robot in the basin shows that the cognitive deprivation effect can occur in a behaving robot without additional restrictions. We find that the homeokinetic controller generates informative actions to improve the quality of the world model.

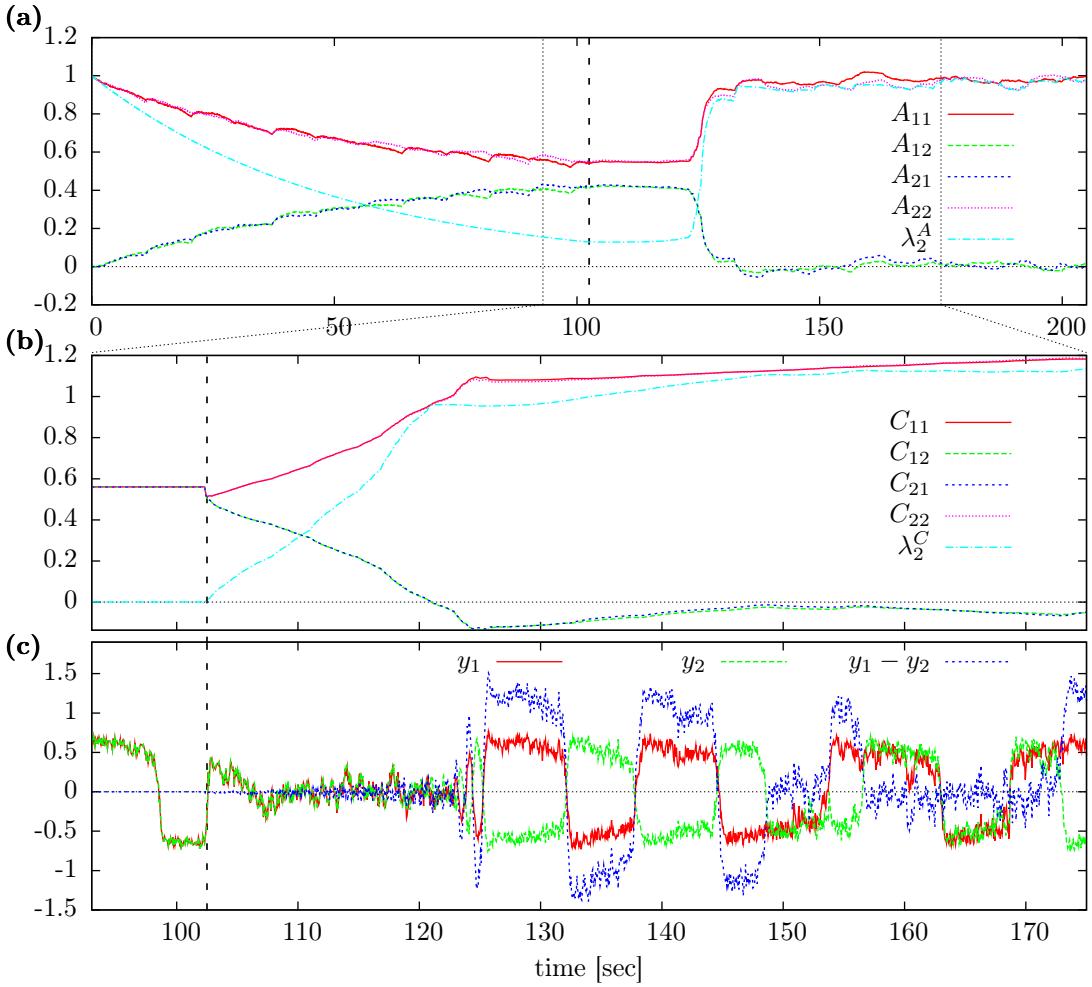


Figure 4.14: Cognitive deprivation and recovery in the case of a TwoWheeled robot. The first 103 sec the robot was controlled by a restricted controller Eq. (4.68) with sine modulated h (oscillating forward/backward). After that the learning of the controller was activated, marked with a dashed line. **(a)** Model matrix A and its smallest eigenvalue λ_2^A for the full time span. A degenerates until the controller is activated and stays more or less a unit matrix after that; **(b)** Controller Matrix C with smallest eigenvalue λ_2^C; **(c)** Motor commands for left and right wheel y_0, y_1, and the steering value $y_0 - y_1$. One can clearly see that the robot performs rotational actions after the activation of learning. Parameters: $\epsilon_c, \epsilon_a = 0.01$, $\sqrt{E}$, update rate 100 Hz.

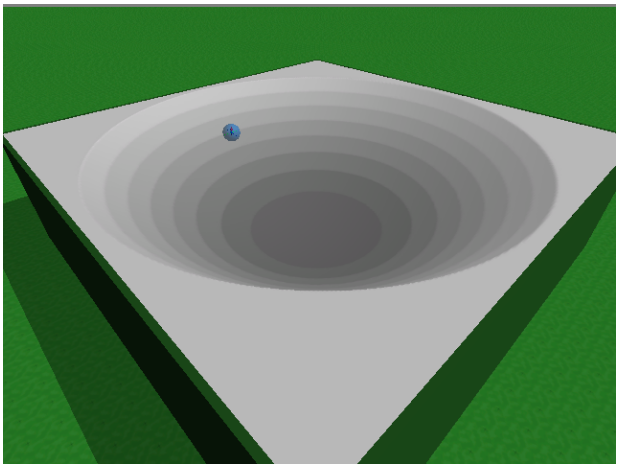


Figure 4.15: Spherical robot in a circular basin.

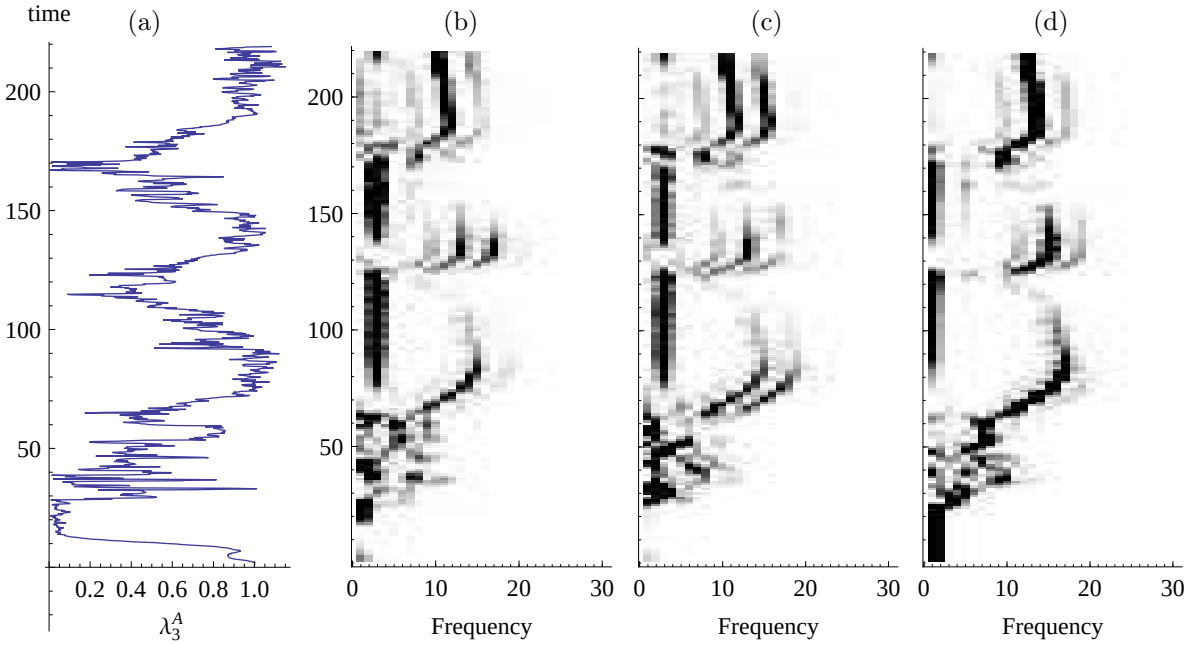


Figure 4.16: Behavior of the Spherical robot in a basin. Note that the time axes are from bottom to top. **(a):** Time evolution of λ_3^A, the smallest eigenvalue of the model matrix A; **(b-d):** Time evolution of the power spectrum of the three sensor values over time. Each row is the power spectrum of a certain time window. Consequent rows are overlapping. Darker pixels stand for more energy. High frequency means here high velocity of the robot. Parameters: $\epsilon_C = \epsilon_A = 0.1$, update rate 50 Hz.

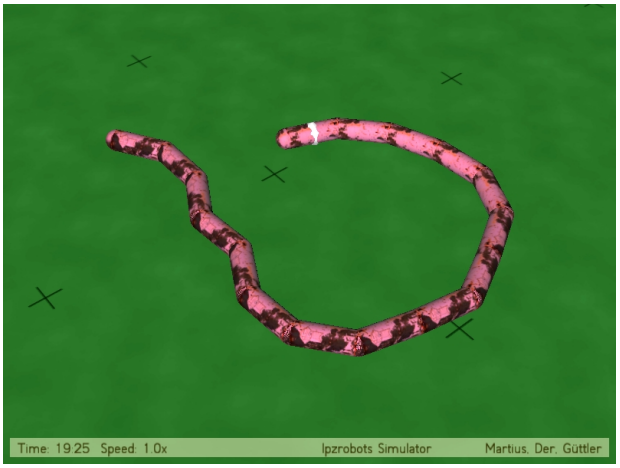


Figure 4.17: Planar Snake robot with 16 segments and 15 degrees of freedom. In this experiment it is fixated at the first segment with a joint (white ring) and has no friction with the ground.

4.5.6 Application to the Planar Snake Robot

Let us now consider the planar Snake robot as depicted in Fig. 4.17 with 16 segments and 15 degrees of freedom. A more detailed description of the hardware can be found in Section 2.2.7. We assume again an artificial controller restriction and investigate the bootstrapping after the release of the restriction in this high-dimensional case. We provide here a more general way to restrict the controller (compared to Section 4.5.4). We define the direction of restriction by the vector $p \in \mathbb{R}^{15}$, with $|p| = 1$, where $P = pp^\top$ is the projection matrix into the subspace to be avoided. The restriction of the controller is achieved by

$$C_t = (\mathbb{I} - P)(C_{t-1} + \Delta C)(\mathbb{I} - P)^\top \quad \text{and} \quad h_t = (\mathbb{I} - P)(h_{t-1} + \Delta h)\,. \tag{4.69}$$

Essentially the restriction is enforced after the parameter update each time step. Hence, the adaptation of parameters C and h is still performed, but the components corresponding to the p-space are projected away. In this example we use $p_i = \frac{1}{\sqrt{15}}(-1)^i$ which corresponds to a zigzag posture of the robot. In Fig. 4.18 the evolution of $|C\,p|$ and $|A\,p|$ are displayed, which are the lengths of the projections of C and A into the space of P. After 120 sec the restriction of the controller was switched on by constantly applying Eq. (4.69). One can see an exponential decay of the world model in the direction of p. At second 360 the restriction was released again and the controller matrix C almost instantaneously recovers (shows a large projection $|Cp|$ into the deprived space). The effectiveness of thereafter elicited actions is shown by the rapid increase of the projections of the model matrix A.

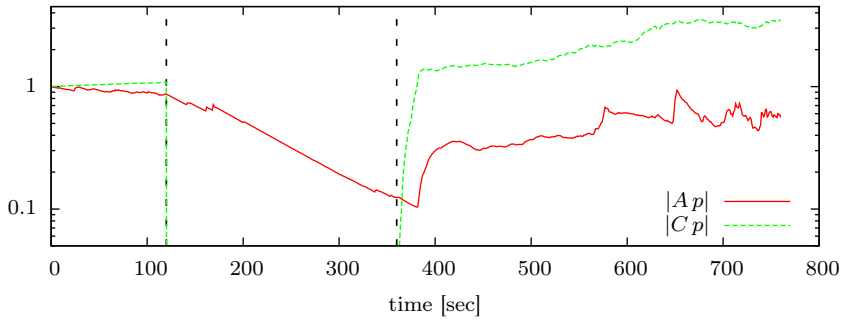


Figure 4.18: Cognitive deprivation and bootstrapping with the Snake robot. p is a vector pointing in the deprived direction. After 120 sec the restriction of the controller was switched on and at second 360 the restriction was released again.
Parameters: $\epsilon_C = \epsilon_A = 0.01$, $\sqrt{E}$, update rate 100 Hz.

4.5.7 Summary

If internal models shall be learned in high dimensional systems then one faces the curse of dimensionality. The random sampling of actions is not very promising in such a setup. Additionally, in the closed-loop setup with simultaneous learning of world model and controller it is well possible that substantial subspaces in the sensor-action space are not visited. We have shown analytically that a restricted controller leads to a deprived world model. However, we found that the homeokinetic approach provides a natural solution for this problem. The controller generates explicitly motor commands that are directed into the unknown regions of the sensor-action space. This has been clearly demonstrated in several experiments, where a restriction was manually added, even in high-dimensional systems with 15 DOF. In all cases the system recovered and ensured correct world model learning. In an experiment with the Spherical robot in a basin, where long-lasting stable behaviors are observed, we could show that the deprivation and recovery actually happens under normal conditions. We understand this as early robot development, i. e. the first step of a self-organized development towards ever-increasing behavioral competencies and understanding of the behavior of the body in its environment. We published the results in [45].

4.6 Low-Dimensional Modes

In this section we report on some phenomenological findings in high-dimensional systems with large inertia. Using, for instance, the Snake robot as in Section 4.5.6, we observe that after some time the robot shows only a few, but highly coordinated behaviors. For example, the robot is shaped like a crescent and moves such that the curvature alternates. The behavior requires all joints to behave in a similar manner – in a collective way. More

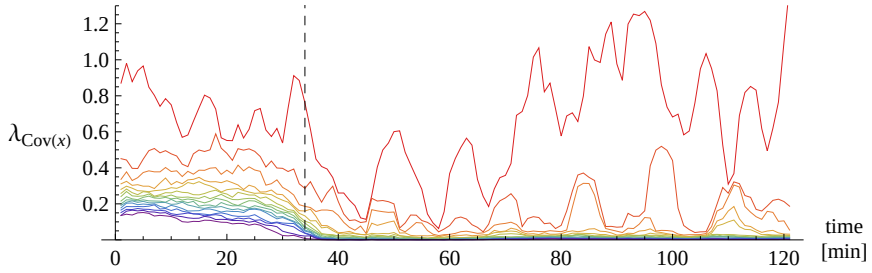


Figure 4.19: Eigenvalues of the covariance matrix of the Snake's sensor values. These are the variances along all 15 principal components of the sensor data of 5 minute sliding windows. From 30 min on the number of effective principal components decreases, so that later only one to three principal components are sufficient to describe most of the data. Parameters: $\epsilon_C = \epsilon_A = 0.01$, $\sqrt{E}$, update rate 100 Hz.

insights into the behavior, especially into the dimensionality, can be obtained with the help of the principal components analysis (PCA) of the sensor values. The sensor values measure the joint angles, such that they are a description of the body posture. The PCA is a method to transform the coordinate system of a given dataset such that the first coordinate accounts for as much variance as possible and each succeeding component accounts for as much of the remaining variance as possible. The transformation can be obtained with the covariance matrix of the data. The eigenvalues of this matrix describe the variance of the data along the belonging principal component which is given by the belonging eigenvector. There are basically two quantities that we can extract from this. First we obtain the number of principal components that are required to describe most of the behavior in a certain time window. This is done by determining how many eigenvalues are well above zero. The second quantity we can derive from the PCA are the principal components in terms of sensor-value vectors, which are given by the eigenvectors of the covariance matrix.

We conducted an experiment with the planar Snake robot with 16 segments (cf. Section 2.2.7) that last over 2 hours simulated time. For the data obtained from the experiment we find that after some time very few principal components are active, see Fig. 4.19. This verifies the low-dimensional character of the observed behavior. We know that the sensor values lie in the space spanned by the eigenvectors with non-zero corresponding eigenvalues. This also means that when only one principal component is active (only one eigenvalue is non-zero), the sensor value vectors point in the direction of this principal component (eigenvector). Figure 4.20 shows the eigenvalues of the covariance matrix (as in Fig. 4.19) but here in a normalized way together with the eigenvectors of the two largest eigenvalues for successive windows of 1 minute. In the first 30 min a large number of principal components are used, represented by the high values in the corresponding columns in Fig. 4.20(a). Later only one or two main principal components dominate. Figures 4.20(b),(c) display these two principal components scaled by their relative variance.

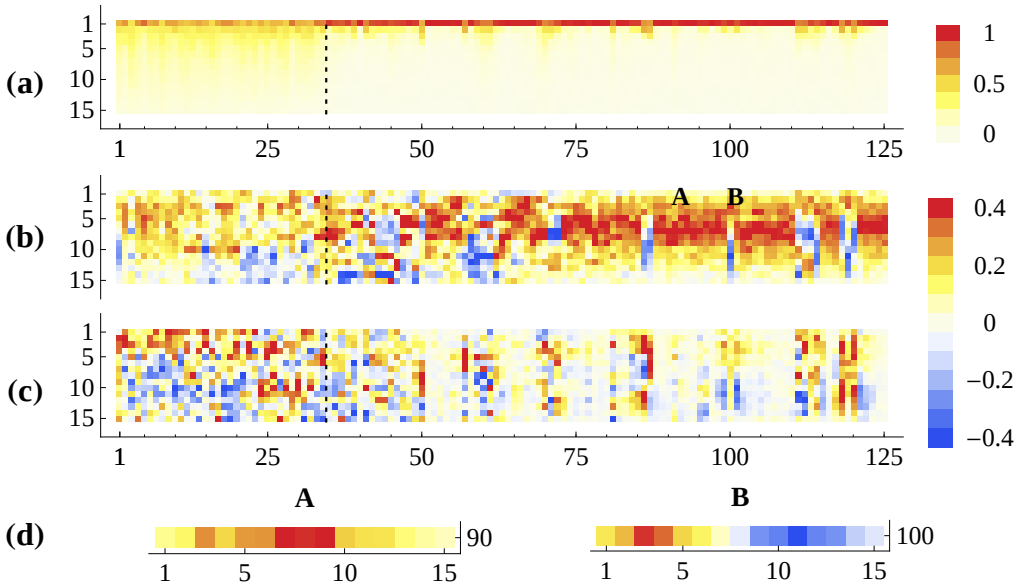


Figure 4.20: Principal components of the Snake's sensor values over time. The time is given in minutes. The covariance matrices have been computed on the sensor data from one minute each. **(a)** Each column shows the normalized vector of eigenvalues of the covariance matrix which is the variance along the principal components of 15 dimensional sensor values during one minute. $\vec{\lambda}^{\mathrm{Cov}(x)}/|\vec{\lambda}^{\mathrm{Cov}(x)}|$; **(b)** Each column shows the eigenvector u^1 corresponding to the largest eigenvalue $\lambda_1^{\mathrm{Cov}(x)}$ (first row of **(a)**). The vector is scaled with the value of eigenvalue, meaning $u_1^{\mathrm{Cov}(x)}\lambda_1^{\mathrm{Cov}(x)}/|\vec{\lambda}^{\mathrm{Cov}(x)}|$. Alternative interpretation: first principal component in sensor space scaled with the relative variance along this component. **(c)** The second eigenvector scaled with the corresponding normalized eigenvalue (second row in **(a)**); **(d)** Enlarged two columns of **(b)** at min 95 (**A**) and 100 (**B**).
After minute 36 a drastic change in the distribution of variances **(a)** occurs and only one or two principal components dominate. Later two principal components are alternately active, marked with **A** and **B**. These correspond to the eigenfunctions of the underlying physical system, meaning a half waveform or full waveform, see also Fig. 4.21.

Figure 4.21: Main modes of the Snake robot The labels **A** and **B** correspond to the two principal components plotted in Fig. 4.20

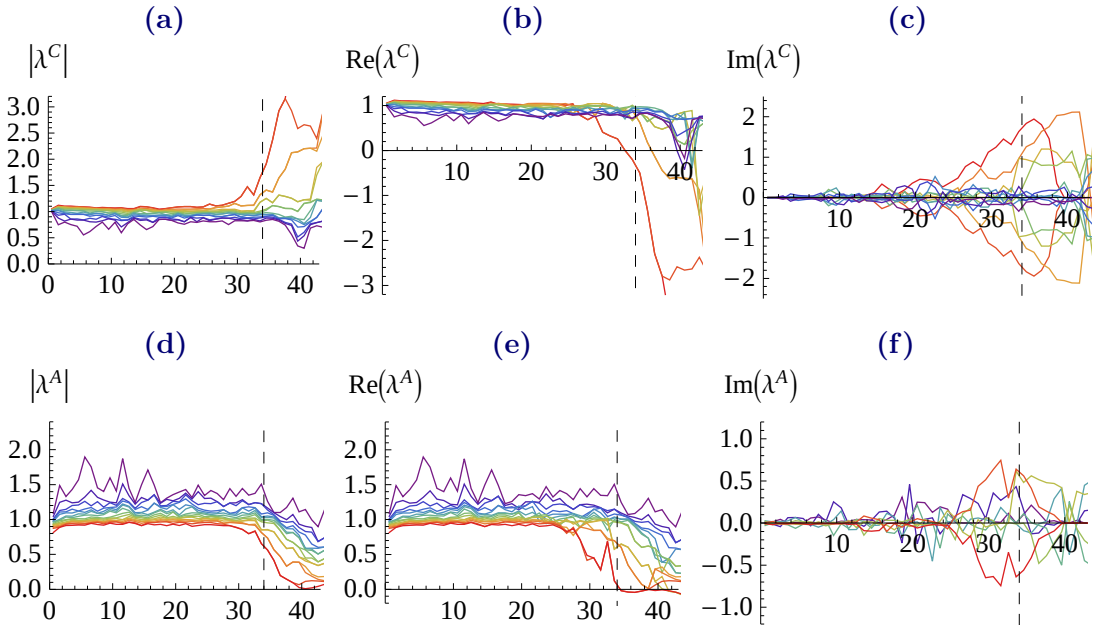


Figure 4.22: The eigenvalues of the controller and world model for the first 45 min. The time is given in minutes and the curves show sliding averages of one minute intervals. **(a-c)** Eigenvalues of controller matrix C; **(d-f)** Eigenvalues of world model matrix A; **(a,d)** Absolute values; **(b,e)** Real components; **(c,f)** Imaginary components, **(f)** only shows a subset for clarity.
The crucial point is at minute 29, when the largest eigenvalue of C flips its sign **(b)**. There are also two eigenvalues with large imaginary components, which correspond to a fast oscillation. At the same time the matrix A starts to degenerate, because the body cannot follow such high frequencies.

The vectors are preprocessed to have a positive sign in the first few components[2], in order to increase clarity. It is striking that after 50 min the first and the second principal component have essentially one of two forms, marked at minute 95 and 100, as displayed in Fig. 4.20(d). Theses modes correspond to a half or a full period of a wave form in terms of sensor values, meaning that the Snake is either shaped like a crescent or an 'S' with different curvature and sign, see Fig. 4.21. This corresponds to the natural modes of the physical system which we call "Eigensnakes", cf. [Video 7].

The reader may wonder, because the reduction to a few low-dimensional modes is a contradiction to the previously discussed bootstrapping phenomenon. Remember that we found that all eigenvalues of the system's Jacobian matrix are kept away from zero, such that high-dimensional modes should occur. To understand this phenomenon, let us have a look at the underlying dynamics in more detail. As a quantitative measure we plot the eigenvalues of the controller matrix C and the world model matrix A for the first 45 min in

[2]The eigenvector is multiplied with -1 if the sum of the first 4 components is negative.

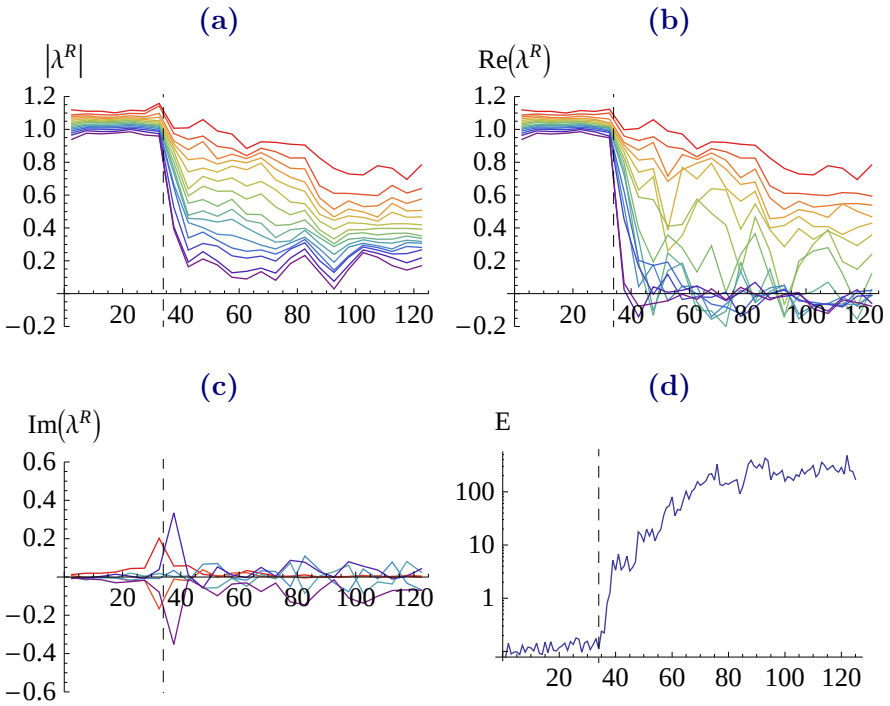


Figure 4.23: The eigenvalues of the linearized system matrix R. The time is given in minutes and the curves show sliding averages of 5 min intervals. **(a)** Absolute values; **(b)** Real components; **(c)** Imaginary components, for clarity only a subset is plotted. **(d)** Value of the error function E.
The eigenvalues of R start to decrease after minute 35 and the error increases rapidly. However, the imaginary parts of the eigenvalues are much lower than those of C and A, see Fig. 4.22.

Fig. 4.22. During the first 25 min we observe a variety of behaviors and all eigenvalues are close to one. Then a period of high frequency oscillations follows, which is also reflected by the raising of the imaginary part of C, Fig. 4.22(c). During this time the value of the TLE raises to a large value. The step width of the gradient descent becomes too large so that large updates occur. After 30 min the largest eigenvalues of C actually gets a negative sign on the real component. The physical body is not capable of following such high frequency oscillations because of strong inertial effects. This causes the smaller eigenvalues of the world model to drop to zero, Fig. 4.22(d). Now, the dynamics follows this pattern and more and more eigenvalues get significant nonzero imaginary values and the world model degenerates in many dimensions.

Let us have a look at the system matrix $R = CA$, which neglects the non-linearities. However, its eigenvalues are very similar to the ones of L, as long as the system is away from the saturation region of the activation function (Section 3.8). In Fig. 4.23 the evolution of the eigenvalues over two hours is plotted. Also here, we see a drop in the real and absolute values between minutes 30 and 40. After that, a rather slow but steady decrease is observed. Essentially only about half of the eigenvalues remain clearly above zero in their real part. Interestingly, the imaginary parts of the eigenvalues are comparably small, much smaller than the ones of A and C. This tells us, again that the physical system cannot perform fast oscillations and the world model acts like an inverse transformation on the action of the controller. A striking figure is given by the value of the error function E, which raises by orders of magnitude at the time of the breakdown, see Fig. 4.23(d).

What is the reason for this dynamics and why does not the bootstrapping mechanism counteract this development? There are actually three reasons. The first one is the oversimplification of the world model, which cannot account for inertia and an action independent dynamics in the environment. This leads to high prediction error in the current setup, which in turn leads to instabilities of the gradient descent. To overcome this problem we will elaborate on a world model extension later (Section 4.8).

The second reason is similar to the first one but applies to the controller. Currently the controller only receives joint positions but no velocities or similar quantities. Thus, in the current example the problems occur when the Snake starts to swing in a resonance frequency and the controller neurons reach their saturation region. This over-excitation leads the controller to damp these oscillations, which is, however, most easily achieved with negative feed back. This in turn leads to negative eigenvalues of the C matrix. In a later experiment in Section 4.8.7 we show that this problem can be circumvented using a new sensor setup and an advanced world model.

The third reason can be found again in the form of the error function itself. Actually, a similar situation was discussed in Section 4.4, where the invariance of the error function to the frequency of sensorimotor oscillations was elaborated. Here, we will make a more general statement and consider again the error function $E = \xi^\top \left(LL^\top\right)^{-1} \xi$ (Eq. (4.12)), where $L = A(g' \circ C)$. The matrix $Q = LL^\top$ is of special interest here, because it is a positive

definite matrix with only real eigenvalues. If λ_i^L are the (possibly complex) eigenvalues of L then Q has eigenvalues $\lambda_i^Q = \lambda_i^L \left(\lambda_i^L\right)^* = \left|\lambda_i^L\right|^2$. The point is that the imaginary parts of the λ^L only contribute to the size of the eigenvalues λ^Q, however the oscillatory character is hidden. Thus, the parameter dynamics is not regulating the frequencies directly. Let λ_i be a complex eigenvalue on the unit circle of the Euler plane. We realize that the sign of the real part can change, even though the absolute value stays constant. Therefore, the dynamics does not need to cross the singularity at $|\lambda_i| = 0$, cf. Section 4.5.3, in order reach the second solution of the fixed point equation of the learning dynamics with negative feedback strengths, which we neglected before as the non-physical solution. To understand what happens in this case, let us consider the one-dimensional case again. In Section 3.7 we calculated the fixed point for positive values of the controller parameters c. Note that $c = \lambda^C$ in the one-dimensional case. For a negative value of c which occurs here, the motor values have alternating sign each time step. With a minimal inertia the robot cannot follow the oscillations and the sensor values will decrease to zero (assuming zero centered joint sensors for example). The fixed point dynamics for c is as before (cf. Eq. (3.34))

$$0 = 1 - 2c \tanh(cx + h)x\,.$$

For $x \to 0$ and $c < 0$ we find $c \to -\infty$.

To avoid this situation we propose a mechanism to reduce the oscillation frequency by minimize the derivative of the motor values, elaborated in Section 4.7.2.

4.7 Controller Extensions

The self-organizing robot behavior that we have seen so far was a product of the interplay between the physical interaction of the robot and the parameter dynamics, which was so far solely driven by the gradient descent on the time-loop error (TLE). Ultimately we aim to guide the self-organization towards desired behaviors. However, for that am appropriate way to influence the parameter dynamics is required. Desired behaviors or constraints to the parameter dynamics can be expressed in terms of an energy function or error function. It seems obvious to combine the TLE and additional constraints into a single error function. This will be the subject of the following section. After that we will apply this method to reduce the frequency of oscillations and to integrate a model of the prediction error.

4.7.1 Integration of Additional Error Functions

In this section we want to set the foundation for the combination of additional error functions with the TLE. This enables to impose additional constraints to the system for example to avoid high-frequency oscillations or to achieve specific behaviors. The combination of self-organization and specific constraints is called guided self-organization [130, 136]. Here,

we investigate two ways to generally integrate additional error functions into our system and apply it afterwards to the reduction of high-frequency oscillations. In Chapter 5 we will make use of the guided self-organization to obtain specific behaviors.

The complicated part in the combination of the self-organizing process and the additional constraints is how to balance the two, because the emergent properties of self-organizing systems are quickly lost if the appropriate working regime is left. Depending on the system the working regime can be confined to a very small parameters regime, e. g. in the case of the aligned magnetized regions in a ferromagnet at the critical temperature (see Section 3.2). In systems that self-regulate the parameters to the appropriate regime, e. g. in self-organized critical systems or in the homeokinetic controller, the integration of constraints is more easy. Nevertheless, it is important to balance the impact of the additional drives and of the self-regulating drives. This is especially important in the case when the additional drives contradict to the self-regulation mechanisms. Remember, the balanced compromise between sensitivity and predictability kept by the homeokinetic controller. If now a constraint leads to a low sensitivity then it can be less fulfilled compared to a compatible constraint. With the appropriate mechanisms there is hope to guide the system gently into a desired direction without loosing the benefits of a self-organizing dynamics.

The most simple approach to combine the TLE and an additional error function would be the plain sum of both functions. This, however, is likely to bring the system out of balance. The reason is that the size of the TLE varies over orders of magnitude, whereas the additional terms are often within one order of magnitude. The decisive quantity is of course the steepness of the gradient rather than the size of the error itself. However, large variations in size also imply a steeper gradient. At a first glance this size difference seems to be no problem, because the TLE could bring the system back into balance when its term dominates. But this is exactly the problem that there are periods when one term dominates the dynamics and periods when the other term dominates. In other words, we cannot find a fixed factor for the additional error function that would cause it to show an effect in the average situation without destroying the self-organization.

Our first proposal is to scale the influence of the additional term with the size of the TLE. Let us call the additional error function E^a, with the superscript a standing for $a$dditional. Note that later we will use other superscripts for particular additional error terms. More formally, we first consider the update for the controller parameters caused by the additional error alone, i. e.

$$\Delta C^a = -\frac{\partial E^a}{\partial C}\,. \tag{4.70}$$

The overall update with the scaled additional term reads now (cf. Eq. (4.24) for comparison)

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} + \gamma_a E \Delta C^a\,, \tag{4.71}$$

where the factor γ_a defines the strength of the additional term with respect to the TLE (E). The entire update size is still controlled by the learning rate ϵ_C.

This method enables the integration of the additional error quantity without destroying the self-organization process, given a conveniently chosen value of γ_a. However, this approach has the disadvantage that the average size of the TLE (E) is unknown, such that the size of the factor γ_a is very application dependent. Furthermore, the influence of E^a on the behavior is very little if the TLE is small. Remember that the additional error specifies some kind of goal, which may not be sufficiently followed in this way. A solution would be the normalization of E in the scaling factor, but this is complicated to achieve online.

Our second solution to the integration was obtained from the idea to combine the additional error directly with the prediction error. The starting point was an additional error function in the special form $E^{a'} = \zeta^\top \zeta$, where ζ is a vector in motor space. For the formulation of the sensorimotor dynamics in motor space as defined in Sections 4.1.2 and 4.1.6, we could write $E = (\eta+\zeta)^\top (JJ^\top)^{-1}(\eta+\zeta)$ as the total error. Unfortunately, this modification of the error function leads to instabilities. Another way to modify the gradient is to transform it into another metric. The plain gradient descent is only optimal if the space on which the gradient is defined is not Euclidean. For an arbitrary Riemann metric the transformed gradient is called natural gradient. The gradient has to be multiplied with the inverse of the metric to produce the steepest direction as proven by Amari [3]. A more detailed discussion can be found in [60], where the natural gradient was applied to the gradient of the TLE directly to achieve a noise-dependent sensor integration. For our application we can use the metric defined by the Jacobian matrix, i. e. $JJ^\top$. Instead of a scaling factor, as in Eq. (4.71), we get here a 'scaling' matrix. The Jacobian matrix J of the sensorimotor loop contains the non-linearities of the controller but not the prediction error. Depending on whether we use the sensor space or the motor space implementation of the controller we have to use the scaling matrix as a right-hand or left-hand factor. In sensor space we find

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} + \gamma_a \Delta C^a \left(LL^\top\right)^{-1}. \tag{4.72}$$

Note that $\frac{\partial E}{\partial C}$ is given by Eq. (4.30). In motor space J is an $(m \times m)$ matrix and hence,

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} + \gamma_a \left(JJ^\top\right)^{-1} \Delta C^a, \tag{4.73}$$

where $\frac{\partial E}{\partial C}$ is given by Eq. (4.38). For the update of the parameter h we apply the same procedure, i. e. in motor space we find

$$\frac{1}{\epsilon_C}\Delta h = -\frac{\partial E}{\partial h} - \gamma_a \left(JJ^\top\right)^{-1} \frac{\partial E^a}{\partial h}, \tag{4.74}$$

cf. Eqs. (4.25, 4.39). Intuitively, the step size in the direction of low response strength (low sensitivity) is larger compared to the normal gradient. Also, the factor γ_a is less dependent on the particular application.

To summarize, we now have two ways to integrate additional goals in terms of error function into the update rule of the homeokinetic controller. The first one uses a pure scaling with

the TLE to obtain a balance between the self-organization and a desired drive. The second way uses a transformed gradient using a metric defined by the "sensitivity" term of the original TLE. It is less dependent on the size of the prediction error and maintains a better balance between TLE and the additional error function.

4.7.2 Continuity Preference

In experiments with multidimensional systems we often observe high frequency oscillations of the motor values. This is rarely desired in robotic applications, except maybe for the Spherical robot, which can physically perform fast rotational modes. The phenomena of high frequencies is a side-effect of the earlier discussed exploratory character of the algorithm that spans to the frequency domain as well. More precisely, in systems with more than one degree of freedom oscillations in state space of any frequency can be achieved (assuming the physical system is capable of performing them) when the system dynamics (Eqs. (4.4, 4.13)) performs a rotation of the state each time step. This happens if the Jacobian L is a rotation matrix or more generally if $L \in SO(n)$ (special orthogonal group). Then the product $LL^\top = \mathbb{I}$ and, therefore, E is invariant with respect to the rotation angle, cf. Eq. (4.12). In practice L might not be exactly a rotation matrix, e. g. due to non-linearities, but it is close enough, such that a large range of rotation angles can be visited in a more or less random walk fashion, see Section 4.4 for more details.

In order to induce a preference for low frequencies we require a continuity in the motor value stream. In other words, the derivatives of the motor values should be small. This can be achieved by including an additional error function that minimizes the difference between subsequent motor values. Thus we define E^c, where c stands for *c*ontinuity, as

$$E^c = \left(|y_{t-1} - y_t|^2\right)^2 . \tag{4.75}$$

The power 4 is used to have only little influence on slow oscillations and a large impact on fast oscillations. The easiest way to minimize Eq. (4.75) is to consider the last motor value (y_{t-1}) as given and use supervised learning schema. Using the gradient descent we find the additional update for C as

$$\Delta C^c = |y_{t-1} - y_t|^2 \cdot ((y_{t-1} - y_t) \circ g')\, x_t^\top , \tag{4.76}$$

where $g' = g'(Cx_t + h)$. This update is integrated into the total learning rule using the natural gradient as discussed in Section 4.7.1. The total update in motor space is thus,

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} + \gamma_c \left(JJ^\top\right)^{-1} \Delta C^c . \tag{4.77}$$

Experiment using Short Circuit

Let us consider again the trivial toy world (Short Circuit)

$$x_{t+1} = y_t + \varsigma_t\,,$$

where ς is a small white noise term. We can, for instance, initialize the controller as a scaled rotation matrix

$$C = 1.15 \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{4.78}$$

with the rotation angle θ to directly obtain a certain frequency of oscillation. The factor of 1.15 is used bring the system close to the fixed point of the learning dynamics, which is at $c \approx 1.19$ in the one-dimensional noise-free case, see Section 3.7. Since here we have noise in the sensors values ($\varsigma \in (-0.1, 0.1)$) the fixed point is at $c \approx 1.15$. In the multidimensional case the learning dynamics would bring the eigenvalues of C to this value (for a linear world and $A = \mathbb{I}$). Remember that the system is time-discrete so that we obtain either true periodic or quasi-periodic behavior. For $2\pi/\theta \in \mathcal{N}$ we have periodic behavior with the period $2\pi/\theta$. Since the controller neurons have a non-linear activation function we observe preferred periods such as period 2 and 4. This phenomena is discussed in detail in e. g. [60, 64]. For example, if we start with $\theta = \pi/3$ (period 6), we find the system locking into a period 4 oscillation ($\theta = \pi/2$). Figure 4.24(a),(b) shows this behavior in terms of controller parameters and the corresponding state space. The transition to the period 4 oscillation is clearly visible. When Eq. (4.77) is used the system is quickly changing to the low frequency regime (small θ) even if the system was before already in a period 4 oscillation. This is shown in Fig. 4.24(c),(d) for the initial rotation angles $\theta = \pi/3$ and $\theta = \pi/2$.

This method works up to a certain frequency. In the case of a period 2 oscillation ($\theta = \pi$)it is not possible to come back to low frequencies. The motor values in this case have the same value with alternating sign every time step, i. e. $y_t = -y_{t-1}$. Thus, the error function (Eq. (4.75)) has a maximum at this rotation angle. The error decreases for both lower and higher rotation angles in the same way, such that the gradient descent may as well increase θ.

To summarize, in a simple example we successfully implemented the preference for low frequency oscillations. This was achieved by integrating the derivative of the motor values as an additional error function into the total error function using the scaled gradient method. A more complex example with the snake-like robot is given later in Section 4.8.7, because a combination of the following extensions is used.

4.7.3 Model of the Prediction Error

Let us now consider another part of the time-loop error function (Eq. (4.12)), namely the prediction error. When deriving the learning rules in Section 4.1.5, we realized that the

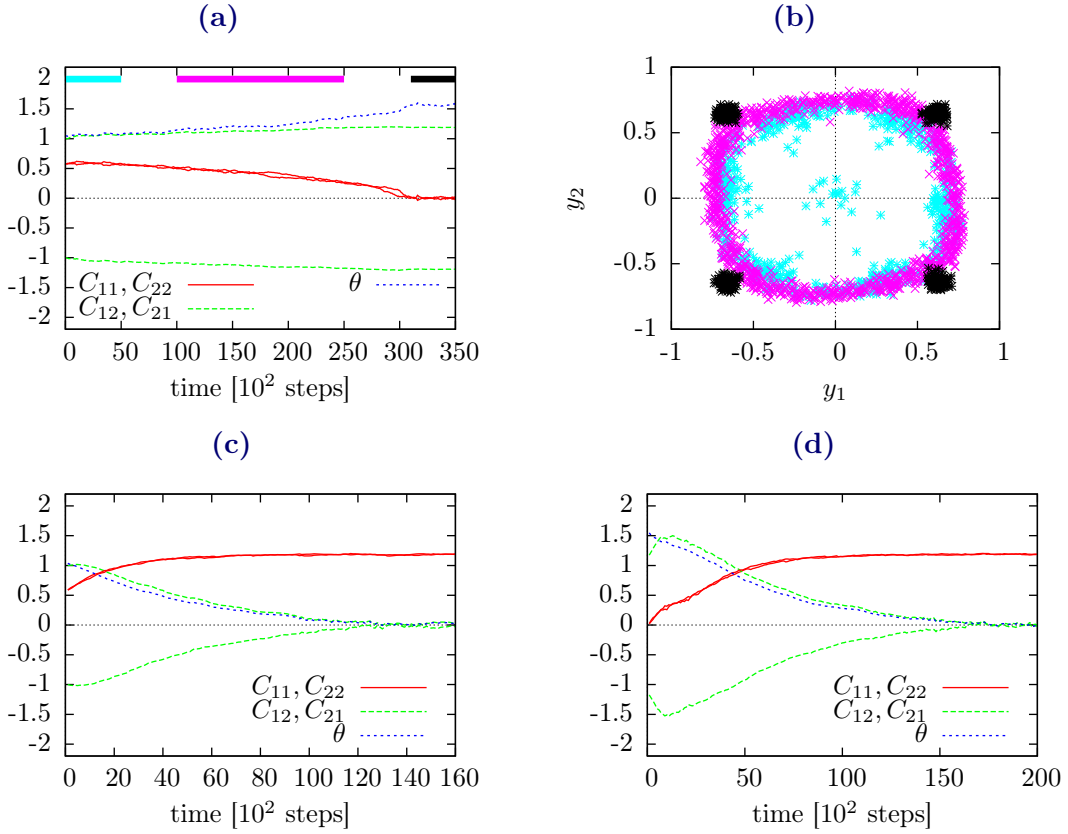


Figure 4.24: Parameter dynamics for oscillatory behavior with and without continuity preference using the Short Circuit setting. The panels **(a,c,d)** show the evolution of the controller matrix C and the rotation angle θ (Eq. (4.78)). **(a)** Normal update rule (Eq. (4.38)), initialized with $\theta = \pi/3$ (60°); **(b)** Motor values corresponding to panel **(a)** for different time intervals, as marked with respective colored bars in **(a)**; **(c)** Dynamics with continuity preference (Eq. (4.77)), initialized with $\theta = \pi/3$, $\gamma_c = 0.01$; **(d)** The same as in (c) but initialized with $\theta = \pi/2$, $\gamma_c = 0.01$. Parameters: $\epsilon_C = \epsilon_A = 0.1$, $\varsigma \in (-0.1, 0.1)$.

actual dependence of the prediction error on the controller parameter is not known. The reduction of the prediction error happens, so far, via the step size of the gradient. For large prediction errors large steps are taken and vice versa. However, the steepest gradient to reduce the prediction error could not be determined. This is not a mathematical problem but rather a fundamental problem. In order to obtain a gradient of the prediction error based on the control parameters one would need to replay each situation with different control parameters. Since the world changes if we execute actions, a repetition of the exact same situation is not possible. A general solution might be conceivable in the framework of reinforcement learning if a sufficiently frequent repetition of similar situations is assumed. In this section we will show how it is possible to directly reduce the prediction error in setups with multiplicative noise. So far we have only considered additive sensory noise, but many physical sensors show rather multiplicative noise characteristics. Let us recall the definition of the prediction error (Eqs. (3.17, 4.3)),

$$\xi_t = x_t - \tilde{x}_t$$

The predicted sensor values $\tilde{x}_t$ have a known dependence on the controller parameters C and h, because $\tilde{x}_t = Ag(Cx_{t-1} + h) + b$ (cf. Eq. (4.23)). However, the true sensor values x_t are also dependent on the actions and therewith on the controller, but in an unknown way.

Here, we propose a method to model the influence of the actions on the prediction error, in order to counteract the upcoming errors directly. To understand the dependencies we consider first a system with artificially applied multiplicative noise. The main difference to additive noise is that multiplicative noise is directly action dependent, so that we can actively counteract the upcoming noise strength. Let us consider again a simple one-dimensional toy world described by

$$x_{t+1} = \alpha y_t + \varsigma_t f(y_t)\,, \tag{4.79}$$

where ς is uniform white noise and $f(y)$ represents the action dependence of the noise strength. In this case the modeling error (one-dimensional) would look like

$$\xi_{t+1} = \alpha y_t - a y_t + \varsigma_t f(y_t)\,. \tag{4.80}$$

Remember that the world model parameter a is adapted to minimize the error ξ^2. Averaging over a long time, we get $\overline{\varsigma} = 0$ (zero mean noise) and thus

$$\begin{aligned}\overline{\xi_{t+1}^2} &= \overline{(\alpha y_t - a y_t + \varsigma_t f(y_t))^2} \\ &= \overline{(\alpha - a)^2 y_t^2} + \overline{\varsigma^2 f(y_t)^2}\,.\end{aligned}$$

After the convergence of a to its optimal value α the modeling error simply consists of the multiplicative noise, i. e.

$$\xi_{t+1} = \varsigma_t f(y_t)\,. \tag{4.81}$$

Since the noise ς is of course unpredictable, the remaining prediction error cannot be modeled. Nevertheless, if we consider the size of the error we can access the dependence described by $f(y)$. This becomes clear if we consider the absolute value averages over time, i. e. we find $\overline{|\xi_{t+1}|} = \overline{|\varsigma_t||f(y_t)|}$ for uncorrelated f and ς. Now we will assume that the multiplicative noise strength f is symmetric for positive and negative y (point or mirror symmetric)[3], such that we can write

$$\overline{|\xi_{t+1}|} = k\overline{f(|y_t|)}\,, \tag{4.82}$$

with $k = \overline{|\varsigma_t|}$ being the fixed noise strength. Intuitively, the size of the remaining prediction error is a function of the size of the motor action.

Before we can introduce the new method we need to specify the componentwise absolute value of vectors, which is required in the multidimensional case. Let $u \in \mathbb{R}^n$ be a vector then we denote its componentwise absolute value by $|u| : \mathbb{R}^n \to \mathbb{R}^n$,

$$|u|_i = \mathrm{abs}(u_i) = |u_i|\,, \tag{4.83}$$

for all $i = 1 \ldots n$.

Using the above considerations, we introduce an adaptive model of the absolute value of the prediction error ξ. Based on Eq. (4.82), we make the ansatz that the size of the prediction error can be modeled by an additional neural network based on the absolute motor values. The network is called ξ-model and we write

$$|\xi_{t+1}| = P(|y_t|) + \zeta_{t+1}\,, \tag{4.84}$$

with the neural network $P(\cdot)$,

$$P(|y_t|) = s(W|y_t|)\,, \tag{4.85}$$

where W is a weight matrix, $s(z) = \frac{1}{1+e^{-z}}$ is the component-wise sigmoid function, and ζ is the residual error. We use here the sigmoid function s instead of the hyperbolic tangent because the image domain of s is solely positive and thus sufficient for the calculation of an absolute value. The parameters of the neural network (W) are adapted to minimize the residual error $\zeta^\top\zeta$ using the gradient descent.

Now we can adapt the controller matrix C to minimize ξ explicitly by following the gradient on $E^P = |\xi|^\top|\xi|$, using Eq. (4.84). First, let us consider the one-dimensional case, where we use lower case letters, i. e. w and c instead of W and C. Remember that $y_t = g(cx_t + h)$ and $\mathrm{abs}'(x) = \mathrm{sgn}(x)$, where sgn denotes the sign function, defined as

$$\mathrm{sgn}(x) = \begin{cases} -1 & x < 0\,, \\ 0 & x = 0\,, \\ 1 & x > 0\,. \end{cases} \tag{4.86}$$

[3]The calculation could also be performed without this assumption. Eq. (4.84) would then read $|\xi_{t+1}| = |P(y_t)| + \zeta_{t+1}$ and all following calculations need to be adapted accordingly.

Using Eq. (4.84) we find that

$$\Delta c^P = -\frac{1}{2}\frac{\partial|\xi_{t+1}|^2}{\partial c} = -|\xi_{t+1}|s'(w|y_t|)\cdot \mathrm{sgn}(y_t)\cdot g'(cx+h)wx_t\,.$$

In the multidimensional case it is a bit more complicated. Let us first define the quantity $\tilde{\xi} = \xi \circ s'(W|y_t|)$. Using the rules for matrix calculation (Section 4.1.3), especially rule (R7), we obtain an additional update term for the controller matrix C as

$$\Delta C^P = -\left[W^\top|\tilde{\xi}_{t+1}| \circ \mathrm{sgn}(y_t) \circ g'\right] x_t^\top\,, \tag{4.87}$$

where g' is taken at (Cx_t+h). Note that the sign function $\mathrm{sgn}(\cdot)$ is applied componentwise. The derivative of the sigmoid function $s(\cdot)$ is given by

$$s'(z) = s(z)(1-s(z))\,. \tag{4.88}$$

The additional update Eq. (4.87) is integrated into the total update term using the natural gradient as discussed in Section 4.7.1. The total update in motor space is thus,

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} + \gamma_P (JJ^\top)^{-1}\Delta C^P\,, \tag{4.89}$$

where γ_P needs to be chosen conveniently.

Experiment

In order to evaluate the new update term for the controller let us consider a simple setup with sensors that produces a different amount of noise depending on the sensory value. This is a common feature of physical sensors, such as infrared distance sensors. We consider a two dimensional system, with multiplicative noise, similar to the one considered at the beginning of this section, cf. Eq. (4.79). Let the world be given by

$$\begin{aligned} x_{1,t+1} &= \alpha y_{1,t} + (1.1 - |y_{1,t}|)\varsigma_{1,t} \\ x_{2,t+1} &= \alpha y_{2,t} + (0.1 + |y_{2,t}|)\varsigma_{2,t}\,, \end{aligned} \tag{4.90}$$

where $\varsigma_{i,t} \in (-0.2, 0.2)$ is uniformly distributed noise. More intuitively, x_1 shows high noise around zero and x_2 shows high noise around high absolute sensor values. Since this perturbation cannot be modeled by the internal world model (Eq. (4.19)) we expect an action-dependent error ξ. By including the learning dynamics based on the ξ-model Eq. (4.89), we expect the connection strengths to y_1 (C_{11}, C_{12}) to become further supercritical to avoid errors at smaller motor values, whereas the weights connecting y_2 (C_{21}, C_{22}) will be decreased to prefer small motor values. The results of a simulation are displayed in Fig. 4.25. During the first 2500 steps the parameter evolution was determined by the TLE only $(\gamma_P = 0)$. We observe the usual development of C, although C_{11} is a already little

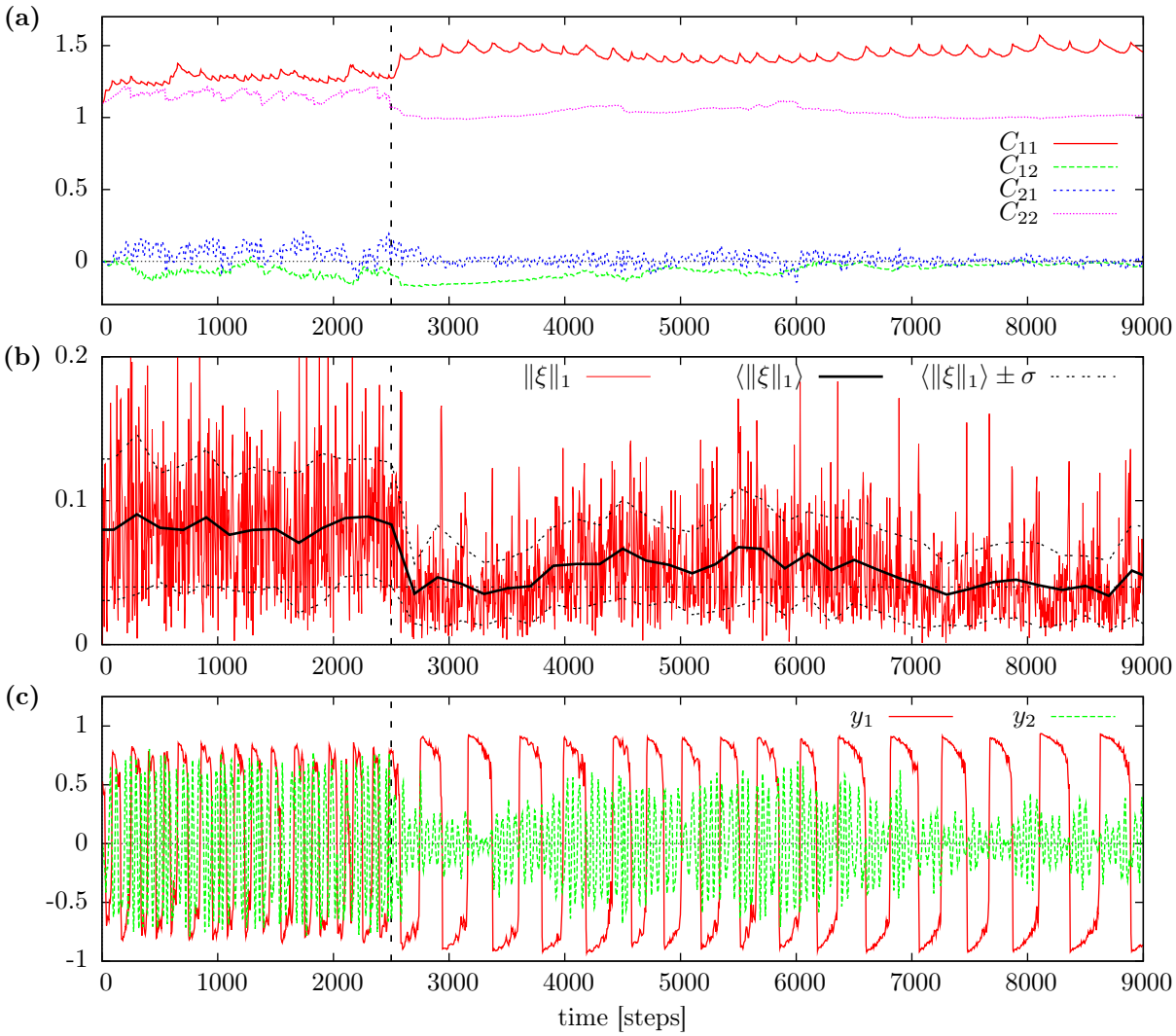


Figure 4.25: Effective error reduction through the ξ-model. **(a)** Values of the controller matrix C; **(b)** Size of the modeling error ξ (Manhattan-norm) with mean and standard deviation σ within sliding windows of 200 steps; **(c)** Motor values.
In the first 2500 steps the unmodified TLE was used ($\gamma_P = 0.0$). After that the learning using the ξ-model, Eq. (4.89), was activated ($\gamma_P = 0.1$). The error is decreased to the noise level (dotted line at 0.04 in **(b)**). The parameter dynamics increases C_{11} and decreases C_{22} such that the errors at small y_1 and large y_2 are avoided. Parameters: $\epsilon_C = 0.75, \epsilon_A = 0.1$.

bit larger than C_{22}. This is because the x_2 shows a stronger noise for the typical value of $x \approx 0.65$ and thus the controller neuron more frequently reaches the saturation region (Section 3.8) causing a decrease of the corresponding connections, especially of C_{22}. After the activation of the new update ($\gamma_P = 0.1$) the prediction error depicted in Fig. 4.25**(b)** decreases rapidly due to increasing C_{11} and decreasing C_{22} as expected, cf. Fig. 4.25**(a)**. The error drops to the noise level which is at 0.04. This value is the result of the Manhattan-norm[4] of the minimal size of the noise in each channel which is $0.1\varsigma \in (-0.02, 0.02)$. The motor values clearly show the opposing effects of the update on the two channels. Nevertheless, different frequencies and amplitudes are exhibited, causing the error to slightly increase around step 4000, but it decreases again later.

We have seen in the experiment that the additional learning rule based on the ξ-model helps to directly minimize the prediction error in systems with multiplicative noise. Nevertheless, the ξ-model might be very inaccurate or even wrong and a resulting update would be counterproductive. This can be easily circumvented by checking whether $|\zeta|$ is significantly smaller than $|\xi|$ and only applying the additional update in those cases.

4.8 Model Extension and Ambiguity

In this section we investigate possible improvements of the adaptive internal world model. The accuracy and suitability of the world model is of high importance because it represents the controller's ability to estimate responses to actions and is used to build a dynamical system representation reflecting the sensorimotor dynamics of the robot. It is therefore used in the self-regulatory mechanisms that adapt the controller to perform body compliant behaviors.

First, we will show the insufficiency of the current world model in an experiment with the Spherical robot. Afterwards, we will see that a simple extension of the world model gives rise to the fundamental problem of ambiguity between sensations, caused by the actions of the robot, and those induced by an independent dynamics in the world. In the following sections we will propose different solutions to deal with this ambiguity.

4.8.1 Shortcomings of Simplified World Model

Up to now an oversimplified world model was used, which models future sensor values solely based on current motor actions, cf. Eq. (4.19)

$$M(x_t, y_t) = Ay_t + b.$$

It is obviously insufficient in cases where parts of the sensor value dynamics follow an action-independent dynamics. Consider for example the Spherical robot, cf. Section 2.2.5,

[4]The Manhattan-norm also known as L^1-norm and is defined as $\|x\|_1 := \sum_{i=1}^{n} |x_i|$.

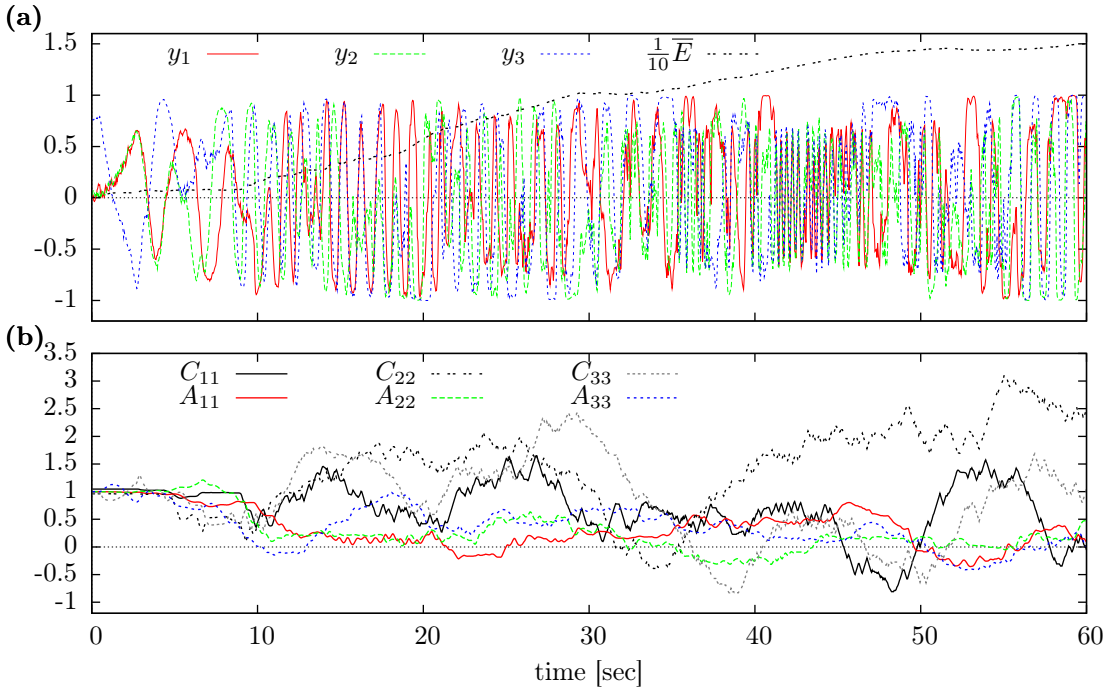


Figure 4.26: Divergent error value for the Spherical robot with light internal masses. Despite the low learning rates $\epsilon_C = \epsilon_A = 0.01$ the parameters fluctuate heavily and the world model collapses. An irregular behavior is observed. **(a)** Motor commands and averaged error ($\overline{E}$) (scaled to fit in the plot); **(b)** Diagonal elements of controller matrix C and world model matrix A. Parameters: $\epsilon_C = \epsilon_A = 0.01$, update rate: 100 Hz, weight of internal masses: 1/2 of hull-mass.

when it is rolling down a slope. The sensors of the robot measure the orientation of the axes with respect to the ground. In this case the sensory dynamics is not caused by the robot actions, but rather by an environmental circumstance. However, a similar effect occurs if the Spherical robot is equipped with light internal masses compared to the weight of its hull and is placed on level ground. After the robot acquired some rolling velocity, the sensory dynamics is again dominated by the current motion and only little influenced by the current actions due to the large inertia. In this hardware configuration the learning dynamics can become unstable, such that a diverging error value is observed. The divergence can happen even with comparably low learning rates of $\epsilon_C = 0.01$, in contrast to the normally used 0.1. Figure 4.26 illustrates the irregular behavior of the robot and the parameter evolution. The high error values are the result of the intrinsic dynamics of the sensor values when the body is rolling, which cannot be captured by the simple world model. In earlier experiments with the Spherical robot we used the square root regularization of the error (Section 4.2.4) and heavier weights to avoid such instabilities.

4.8.2 Ambiguity in the Interpretation of Sensations

For many robotic systems, e. g. the Spherical robot or the Snake robot (Section 4.6), it is necessary to extend the world model to use sensory information as well. In the general formulations in Section 4.1.1 we defined the world model $M(x, y)$ conceptually as a function of sensor values and motor values, cf. Eq. (4.2), but in the particular implementation only the motor values y have been used (Eq. (4.19)). A simple extension of the previous world model is to include a linear dependence on the sensor values as

$$M(x_t, y_t) = Ay_t + Sx_t + b, \tag{4.91}$$

with an $(n \times n)$ weight matrix S.

In a first practical appraisal we found out that S converges to a unit matrix and A to a zero matrix. This seems very surprising, but considering the sensor values of subsequent time steps we find that they are very similar in most applications. Why do we care how the world model chooses to represent the correspondence? It is very important for the correct functioning of the homeokinetic controller that A captures the correct correspondence between actions and future sensations. Remember that the internal representation of the sensorimotor dynamics (ψ) depends on A. It is used to adapt the controller matrix C such that a slightly supercritical feedback strength is obtained. If A now reflects a small response of the sensor values to motor actions, then this leads to large elements in the matrix C and thus the working regime is left.

To get more insight into the problem let us consider again a simple fixed linear controller

$$y_t = Cx_t, \tag{4.92}$$

and a deterministic toy world (without noise)

$$x_{t+1} = W(y_t, x_t) = \mathcal{A}y_t + \mathcal{S}x_t, \tag{4.93}$$

where $\mathcal{A}$ and $\mathcal{S}$ are fixed matrices. Note the different notation of the matrices in the world W (Eq. (4.93)) and the world model M (Eq. (4.91)). Naively we would expect that if Eq. (4.91) is used then A and S converge to $\mathcal{A}$ and $\mathcal{S}$ respectively. However, considering the true sensorimotor loop by putting Eq. (4.92) into Eq. (4.93) we find

$$x_{t+1} = (\mathcal{A}C + \mathcal{S})x_t. \tag{4.94}$$

The model of the loop using Eq. (4.91) reads

$$x_{t+1} = M(x_t, y_t) + \xi_t = (AC + S)x_t + \xi_t. \tag{4.95}$$

The parameters A and S are adapted to minimize $|\xi_t|^2$. Hence, we find $AC + S \stackrel{!}{=} \mathcal{A}C + \mathcal{S}$, which is ambiguous in A and S. The problem is rooted in the fact that y and x are not independent, which is also the case when the controller is subject to constant change due to the learning dynamics. Thus, in the present formulation the ambiguity between self-induced and environment-induced sensations cannot be disentangle. The resolution of this ambiguity is the subject of the following investigations.

4.8.3 Controller Noise to Disentangle Ambiguity

In order to obtain some independence between sensor and motor values we can introduce a perturbation to the motor signal, i. e. for the simple fixed controller we write

$$y_t = Cx_t + \eta_t\,, \tag{4.96}$$

where η denotes an m-dimensional zero mean noise process. The sensorimotor loop reads

$$x_{t+1} = (\mathcal{A}C + \mathcal{S})x_t + \mathcal{A}\eta_t\,. \tag{4.97}$$

The error function for training the model parameters is $E^{pred} = \xi_t^\top \xi_t$, but for simplicity, let us consider the one-dimensional case where the error function is given by $E^{pred} = \xi_t^2$. Hence

$$\begin{aligned}\xi_t^2 &= \left([(\mathcal{A}C + \mathcal{S})\,x_t + \mathcal{A}\eta_t] - [(AC + S)\,x_t + A\eta_t]\right)^2 \\ &= \left([(\mathcal{A}C + \mathcal{S}) - (AC + S)]\,x_t + (\mathcal{A} - A)\,\eta_t\right)^2\,.\end{aligned}$$

In the time average we have $\overline{\eta} = 0$ such that the linear term vanishes and the error is essentially a sum of the original error plus a noise-dependent term, i. e.

$$\overline{\xi_t^2} = \overline{\left([(\mathcal{A}C + \mathcal{S}) - (AC + S)]\,x_t\right)^2} + \overline{(\mathcal{A} - A)^2\,\eta_t^2}$$

We see immediately that $A \to \mathcal{A}$, due to the second term and consequently also $S \to \mathcal{S}$. However, in practice we find two complications. First, we want only a very small additional noise, in order to avoid the noise influencing the behavior too much, and second, the world usually acts as a low-pass filter due to inertia and therefore the perturbations are damped away.

The controller noise can be used to separate world-intrinsic dynamics from action-induced dynamics. However, by adding a noise term we somehow have to decide about its size and distribution. Another more intrinsic definition of the noise term η is via the inverse of the model,

$$x_{t+1} = M(x_t, y_t) + \xi_{t+1} = M(x_t, y_t + \eta_t)\,,$$

as in the motor space calculations in Section 4.1.2. To calculate η we can use the pseudo-inverse (rule (R10)) and find (cf. Eq. (4.14))

$$\eta_t = {M'_y}^{+}\xi_{t+1}\,. \tag{4.98}$$

Since we do not have ξ_{t+1} at the time when η_t is required, we assume some continuity in time and use ξ_t. Using $M(x_t, y_t) = Ay_t + Sx_t + b$ (Eq. (4.91)) and the pseudo inverse we can explicitly calculate η_t by

$$\eta_t = (A^\top A + \lambda\mathbb{I})^{-1}A^\top\xi_t\,. \tag{4.99}$$

What does it mean to use $y + \eta$ as a motor command? Within the internal description of the sensorimotor loop this motor command leads to a minimal prediction error in the next time step, due to the construction of η. Although, one could argue that it decreases the speed of world model learning because of smaller prediction errors. Nevertheless, it follows the idea of bringing external and internal dynamics closer together, which is one goal of the controller. It should be stressed that it is not necessarily important to be able to control and model all possible behaviors, but rather to control the system such that the behaviors are predictable. Note that this mechanism is not as effective in reducing the prediction error as the ξ-model (Section 4.7.3), since it does not change the controller parameters. However, an even more effective and general approach is elaborated in the next section.

4.8.4 Assume Maximal Self-Induced Observations

We want to consider another approach to the disambiguation problem. Let us reiterate the meaning of A and S in the new model (Eq. (4.91)). The weight matrix A captures the direct causes of actions y_t on the forthcoming sensor values x_{t+1}, whereas S represents the intrinsic dynamics of the environment captured by the sensor values. We have seen in the toy example from Section 4.8.2 that both extremes are possible, namely to consider everything self-induced or everything environment-induced, independent of the actual correspondence. The approach we want to pursue now is to assume maximal self-induced changes in the sensor values. We argue that it makes sense for an autonomous agent to assume controllability of its sensations. Remember also that the controller is generally adapted to make the internal representation of the sensor dynamics match the actual sensor dynamics. If the world model is adapted such that an external dynamics is explained in terms of its own actions then the controller tends to produce those actions that are coherent with this model. In fact, we assumed full controllability with the original world model. In the toy example a simple solution would be the damping of S. However, in real applications the environment is more complicated and the correspondence of motors to sensors is not linear because of inertial effects and other perturbations. In this case the size of the damping has to be tuned in order to balance between the preference for self-induced actions and the actual correspondence.

The solution we propose is to request that a small perturbation in the sensor values x, say $\zeta \in \mathbb{R}^n$, can be compensated by a shift $\eta \in \mathbb{R}^m$ in the actions y, i. e.

$$M(x_t + \zeta_t, y_t) = M(x_t, y_t + \eta_t)\,,$$

with η being as small as possible. In other words, the world model is not only trained on the prediction error (Eq. (4.20)) but also on η,

$$E^{\text{pref}} = |\xi_t|^2 + |\eta_t|^2\,. \tag{4.100}$$

In general, η_t can be approximated by

$$\eta_t = {M'_y}^{-1}(x_t, y_t) M'_x(x_t, y_t)\zeta_t\,, \tag{4.101}$$

which reads in the case Eq. (4.91) is used for the model

$$\eta_t = A^{-1} S \zeta_t \,. \tag{4.102}$$

Following the calculations in Section 4.1.4, especially Eq. (4.21), we need to calculate $\frac{\partial \eta^\top \eta}{\partial A}$ and $\frac{\partial \eta^\top \eta}{\partial S}$ as

$$\begin{aligned}
\frac{\partial \eta^\top \eta}{\partial A} &= -2\eta^\top A^{-1} \frac{\partial A}{\partial A} A^{-1} S\zeta \\
&= -2 \left(A^{-1}\right)^\top \eta\eta^\top \,, \\
\frac{\partial \eta^\top \eta}{\partial S} &= 2\eta^\top A^{-1} \frac{\partial S}{\partial \zeta} \\
&= 2 \left(A^{-1}\right)^\top \eta\zeta^\top \,,
\end{aligned}$$

to obtain the new update equations for A and S as follows:

$$\frac{1}{\epsilon_A} \Delta A = \xi_t y_{t-1}^\top + \left(A^{-1}\right)^\top \eta_t \eta_t^\top - \frac{1}{\tau_A} A \,, \tag{4.103}$$

$$\frac{1}{\epsilon_A} \Delta S = \xi_t x_{t-1}^\top - \left(A^{-1}\right)^\top \eta_t \zeta_t^\top - \frac{1}{\tau_A} S \,. \tag{4.104}$$

Inspecting Eq. (4.103) we find that the second term (which is new) depends on the size of η which in turn depends on S. If e. g. ζ is strongly contracted by the application of S in Eq. (4.102) (meaning S does not model this direction) then the impact of the additional term is small. The perturbation ζ can be chosen to be a noise vector or the prediction error. We use the latter and thus set $\zeta_t = \xi_t$.

Experiment with the TwoWheeled Robot

We conducted an experiment with the TwoWheeled robot on flat ground. The robot is described in detail in Section 2.2.1. For simplicity the robot had no obstacles and could move freely. The motor values control the torque of the motors, which are attached to the wheels. The sensors read the actual rotation speed of the wheels.

In Fig. 4.27(a) the physical properties of the robot are tested with impulse shaped actions in order to demonstrate the effects of inertia and slip, see also [Video 8]. We observe that the wheels are shortly slipping when the robot starts to accelerate abruptly, as indicated in the plot. Otherwise the mass of the robot leads to a slow incline of the measured wheel velocity. When decelerating sharply, the wheels slide again so that a seemingly abrupt response is measured. However, the slip occurs only rarely when the robot is controlled with the homeokinetic controller (not shown), because the motor values change more smoothly, so that only inertial effects remain. In any case, the assumed linear correspondence of

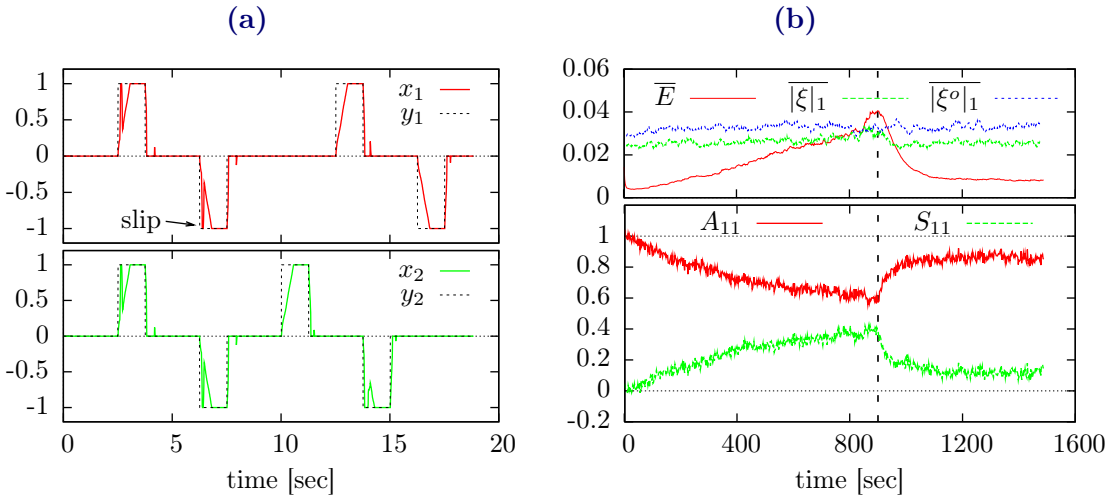


Figure 4.27: TwoWheeled robot: Illustration of inertial effects and the experiment with extended world model. **(a)** Reaction of the robot to impulse-shaped actions for both wheels, y: predefined actions, x: wheel velocity sensors; **(b)** Experiment with homeokinetic controller and extended world model using Eqs. (4.103, 4.104). For the first 900 sec the new update term was switched off by setting $\eta = 0$. **Top**: Prediction error (ξ) and TLE (E) for the case with the extension, and prediction error for a run without the extension (ξ^o) (sliding averages of 10 sec intervals). **Bottom**: First diagonal elements of extended world model matrices A and S. After the activation ($\eta > 0$) a quick recovery to high values of A_{11} occurs and the prediction error decreases as well. Parameters: update rate $100\,Hz$, $\epsilon_C = \epsilon_A = 0.1$, $\tau_A = 1000 \cdot 100$ ($1000\,\text{sec}$).

actions and sensation is not correct. Let us calculate the velocity v of the robot for the one-dimensional case (both wheels are considered as one wheel), which is given iteratively by

$$v_{t+1} = v_t + a\Delta t\,, \tag{4.105}$$

where a is the acceleration caused by the motors. The motors are such that their torque depends on the difference between actual and desired velocity. For simplicity we assume a wheel with radius 1, such that effective translational force is equal to the motor torque. Hence,

$$a = \frac{f}{m}(v^s - v_t)\,, \tag{4.106}$$

where v^s is the desired velocity, f is the motor torque[5] and m is the mass. Putting Eq. (4.106) in Eq. (4.105) we get

$$v_{t+1} = \frac{f}{m}\Delta t\, v^s + \left(1 - \frac{f}{m}\Delta t\right) v_t\,. \tag{4.107}$$

For our robot the velocity is measured by the wheel counters, thus $x_t = v_t$. The nominal velocity is given by the motor value and therefore $y_t = v^s$. Hence, we get $x_{t+1} = f/m\Delta t\, y_t + (1 - f/m\Delta t)\, x_t$, which has the same form as the extended world model, $\tilde{x}_{t+1} = Ay_t + Sx_t + b$ (Eq. (4.91)). Thus, the extended world model is suitable for this case.

In the experiment the additional term in the update rule (Eqs. (4.103, 4.104)) was switched off by setting $\eta = 0$ during the first 900 sec. Indeed, the learning leads to the increasing influence of S which results in a decrease of the values of A, cf. Fig. 4.27(b). After 900 sec the full update rule is used and a quick regulation to a high value of A takes place. This is correct because the inertia effect is comparably small. As expected the diagonal elements of S maintain a certain non-zero level. The system is symmetric, so that the two diagonal elements are identical and the non-diagonal elements of A and S are close to zero (not shown). In comparison to the case without the extended world model the prediction error is reduced as depicted in Fig. 4.27(b), thus, the additional update term performs as expected.

Experiment with the Spherical Robot

In the following experiment we will see that the extended world model makes it possible to control the Spherical robot with light internal masses, which was problematic before, cf. Section 4.8.1. The hope is that the extended world model is able to make better predictions especially before stable modes are found. The world model can thus correctly predict the dynamics in the environment that is not or only partially controlled by the robot's actions. This is, for instance, the case with the currently considered Spherical

[5] f is actually the motor torque per velocity deviation of 1 (units are arbitrary)

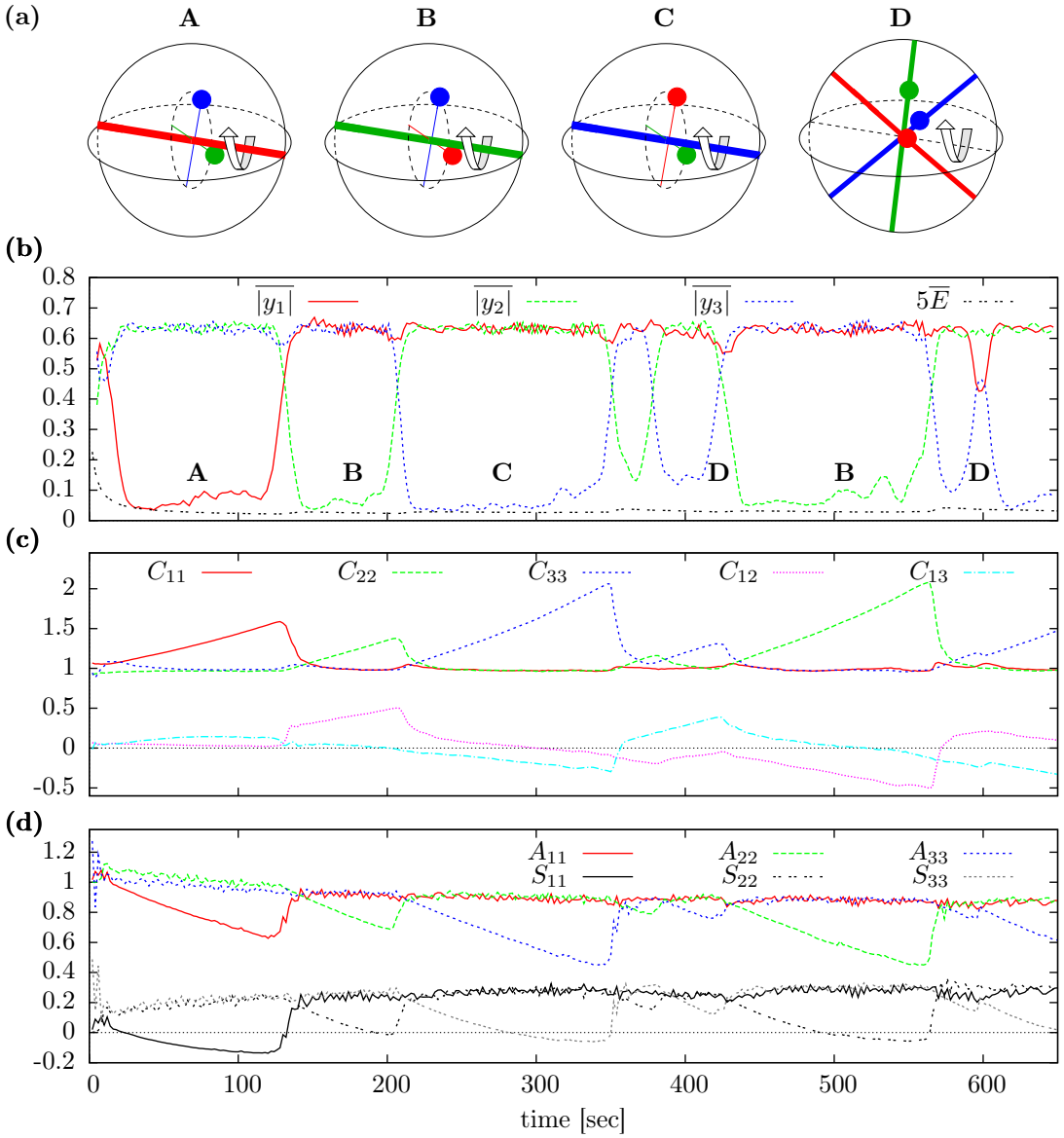


Figure 4.28: Smoothly behaving Spherical robot using the extended world model. The error function has a low value and the behavior is smooth but still diverse. **(a)** Sketch of four typical behaviors (**A-D**); **(b)** Envelop of motor commands and the error averaged over 10 sec (scaled for visibility). Corresponding behaviors are indicated with letters **A-D**; **(c)** Diagonals and two non-diagonal elements of the controller matrix C; **(d)** Diagonals of the world model matrices A and S. Parameters: update rate $100\,Hz$, $\epsilon_C = \epsilon_A = 0.1$, use of extended world model (Eqs. (4.103, 4.104)).

robot which can be excited to perform a rolling mode which will persist for some time even in the non-actuated situation. The evolution of the behavior and the parameters during the experiment with the Spherical robot are depicted in Fig. 4.28. In contrast to the earlier experiment (Fig. 4.26) without the extended model, the error now falls to a low value. Beside the error curve the positive envelops of the motor commands are plotted in Fig. 4.28(b). The envelops are used since the actual motor values oscillated at a frequency corresponding to the rolling speed of the robot. Hence, the envelops reflect the amplitude of the oscillating masses on each axis. If the robot is controlled so that one of the internal axes is the rotation axis, then the mass movement along this axis is small, which is seen in a drop of the envelop. For further illustration the behavioral modes are schematically depicted in Fig. 4.28(a), namely, the rolling mode around the three internal axis (**A-C**) and around another axis (**D**). The latter requires a qualitatively different coordination. The influence of the additional term S varies in different situations due to the additional update term for the preference of self-induced changes, cf. Eqs. (4.103, 4.104). Right at the start a short period of larger values of the diagonals of S are observed, see Fig. 4.28(d). This is due to the fact that the internal model of the sensorimotor dynamics is incorrect and has to adapt first. Hence, the additional term S captures most of the dynamics. After the first 15 sec the parameter dynamics is smooth and periods of stable rolling behavior follow. In the modes of rolling around a particular internal axis the sensory response along this particular rotation axis is low, due to gyro effects. Thus, the corresponding weights shrink (e. g. A_{11} up to second 120). At the same time a destabilization along this axis occurs by a raising value of the corresponding element of C, as displayed in Fig. 4.28(c), until a change of behavior is initiated.

To summarize, we implemented a learning rule for the extended world model that has a bias towards self-induced sensory changes. This has proven effective in the considered examples. In the next section we will consider a simpler and more scalable method to achieve the same effect.

4.8.5 Enhanced World Model

The method proposed in the previous section works well, but it is computationally rather expensive. Here, we will propose a simpler way to achieve a preference for self-induced observations. We will use a special discounting for the training of the world model.

The new world model is again given by $M(x_t, y_t) = Ay_t + Sx_t + b$, see Eq. (4.91). Since the model matrix A (action to sensation mapping) should capture most of the correspondence, we discount the additional term (S) during learning. Thus, there are two error function for the learning process. The terms S and b are trained to minimize the unmodified prediction error (Eq. (4.20)), i. e. $E^{pred} = \xi^\top \xi$ with $\xi_t = x_t - (Ay_{t-1} + Sx_{t-1} + b)$ for the new world model. The adaptation of A is done using $E'^{pred} = \xi'^\top \xi'$, where ξ' is given by

$$\xi'_t = x_t - (Ay_{t-1} + (1-\delta)Sx_{t-1} + b) \,, \tag{4.108}$$

with the discount factor $\delta \ll 1$. Though the minimization of E'^{pred} the matrix A will adapt to model a small part of the mapping represented by S at each update cycle, if that is consistent. The effect of this amended learning on the prediction quality of the model is negligible. The update rules for all parameters of the enhanced world model are

$$\frac{1}{\epsilon_A}\Delta A = \xi'_t y_{t-1}^\top - \frac{1}{\tau_A} A, \tag{4.109}$$

$$\frac{1}{\epsilon_A}\Delta b = \xi_t, \tag{4.110}$$

$$\frac{1}{\epsilon_A}\Delta S = \xi_t x_{t-1}^\top - \frac{1}{\tau_A} S. \tag{4.111}$$

Eq. (4.109) can also be written as in terms of ξ_t using Eq. (4.108) and we obtain

$$\frac{1}{\epsilon_A}\Delta A = (\xi_t + \delta S x_{t-1})\, y_{t-1}^\top - \frac{1}{\tau_A} A, \tag{4.112}$$

which illustrated the implementation of the discounting more clearly.

In the case of a linear and fixed controller $y_t = Cx_t$ Eq. (4.92) and the toy world $x_{t+1} = \mathcal{A}y_t + \mathcal{S}x_t$ Eq. (4.93), the dynamics converges to $A = \mathcal{A} + \mathcal{S}C^{-1}$ and $S = 0$. The derivation can be found in the appendix in Section A.2. With a non-linear and changing controller this extreme bias towards self-induced observations changes into a balance, which can be adjusted using the discount factor δ. This can be seen in the following simulation. We conducted the same experiments as in the previous section to validate the approach. In the first experiment with the TwoWheeled robot we show the effect of the discount factor. The second experiment with the Spherical robot shows essentially the same result as presented in the previous section, such that we do not show them here, but rather in the appendix in Section A.3.

Experiment

The following experiment uses the TwoWheeled robot (Section 2.2.1) and is identical to the one conducted in Section 4.8.4, except the usage of the enhanced world model with discounted learning. Here we compare the results for different values of the discount factor δ. During the first 900 sec the bias towards self-induced observations was disabled using $\delta = 0$ in Eqs. (4.109–4.111) to show the degeneration of A. Afterwards, depending on δ a quick or slow regulation to higher values of A takes place, see Fig. 4.29. As expected the diagonal elements of S maintain a certain non-zero level, which is larger for low discount factors. The system is symmetric, so that the two diagonal elements are identical and the non-diagonal elements of A and S are close to zero (not shown). The prediction error is not shown here because it behaves essentially like in Fig. 4.27.

To conclude, the enhanced world model works as well as the method proposed in the previous section but is computationally much less demanding. Both methods implement a bias

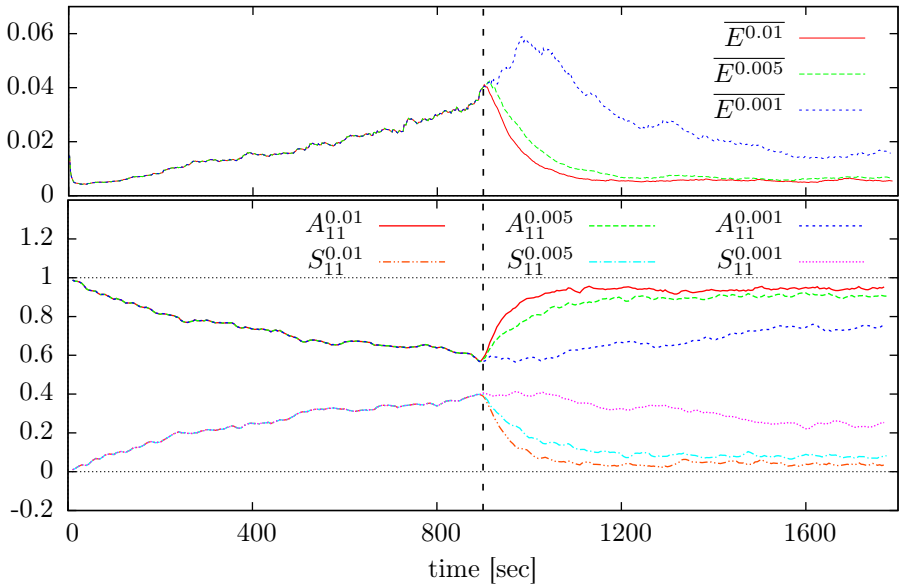


Figure 4.29: TwoWheeled robot with enhanced world model. The discounting in Eqs. (4.109–4.111) was switched off using $\delta = 0$ during the first 900 sec. Displayed are three runs with identical noise process, where δ was set to 0.01, 0.005, and 0.001 respectively after second 900.
Top: Sliding averages of 10 second intervals of the TLE (E). **Bottom**: First diagonal elements of world model matrices A and S for the three runs. (sliding averages of 10 second intervals).Parameters: update rate $100\,Hz$, $\epsilon_C = \epsilon_A = 0.1$, $\tau_A = 1000 \cdot 100$.

towards self-induced observations in the sensor values, which ensures that the estimated action dependence of the sensor values is at least as large as it is in the actual world. This is a promising approach to solve the problem of ambiguity between self-induced and environment-induced observations. Such ambiguity arises quite generally when modeling closed-loop sensorimotor dynamics and thus this approach might be interesting for a wider range of applications.

4.8.6 Advanced Sensor Configuration

Now that a solution to the ambiguity problem was found we want to discuss some consequences for further world model and controller extensions. Up to now, only sensors that measure the same physical quantity that is controlled by the motor actions have been possible. For example, if the motor values control a joint angle then the sensors also had to measure the joint angle, but not its velocity or the like. Nevertheless, we also used sensors that have a less direct correspondence to the motor actions, such as the infrared sensors of the Rocking Stamper in Section 4.3, or the orientation sensors of the Spherical robot. However, in these setups the sensors still show a proportional relationship to the

motor values – at least in certain behavioral modes. An additional constraint to the sensor setup is the number of sensors. Even though the formulation of the controller in motor space made it possible to use more sensor than motors, it caused severe problems at the level of sensor predictions. Recall that in the original world model (Eq. (4.19)) only the current actions are used to predict future sensor values. With the extended world model many senors are possible because the prediction takes the past sensor values into account. Thus, a large range of sensor types, like time-delayed sensor values and derivatives can be used and we can expect the controller to pick the most appropriate ones. The formulas remains identical, because the new sensor values are simply combined into the sensor value vector x. Let us consider the typical case when original sensor values and their derivatives are used. Hence,

$$x = \begin{pmatrix} s_1 & \cdots & s_n & \dot{s}_1 & \cdots & \dot{s}_n \end{pmatrix}^\top , \tag{4.113}$$

where s_i is the i-th original sensor value and $\dot{s}_i$ denotes its temporal derivative. This setup is an important step towards more and more complex systems which we will demonstrate in the following experiment.

4.8.7 Application to Planar the Snake Robot II

In order to test the extensions on a complex example we turn again to the previously considered planar Snake robot, cf. Section 2.2.7. In Section 4.6 we identified several problems with the control of this system. Large inertial effects revealed the insufficiency of the world model and the system also entered high-frequency oscillations. In the experiment considered here we use the same setup and the same parameter as in Section 4.6, but apply the following three extensions:

- the continuity preference for the controller (Section 4.7.2),
- the enhanced world model (Section 4.8.5), and
- the advanced sensor setup as discussed in the previous section (4.8.6).

The robot has 16 segments resulting in $m = 15$ powered joints and 15 joint angle sensors. The sensor values received by the controller consist of the 15 angular sensors, as before, and their first derivative, summing up to $n = 30$ sensors in total. Thus, the controller matrix C has dimensions 15×30, the world model matrices A and S have dimensions 30×15 and 30×30 respectively. To get an idea of the complexity let us calculate the dimensionality of the state space and the parameter space. We use the controller implementation in motor space (Section 4.1.2), and thus we have a 15 dimensional state space with intermediate 30 dimensions and 465 adapting parameters (synaptic weights) of the controller. Together with the 1380 parameters of the adaptive world model this sums to 1845 synaptic weights.

The conducted experiment last 200 min. Over the entire time we observe frequent changes of behaviors, where the eigen-modes of the physical body are occasionally excited, cf.

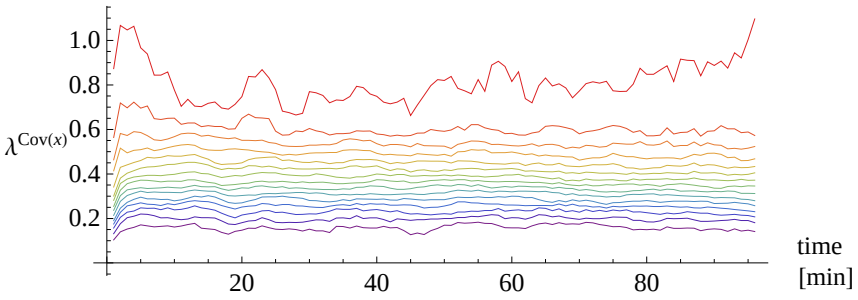


Figure 4.30: Eigenvalues of the covariance matrix of the Snake's sensor values. These are the variances along all 15 principal components of the sensor data of 5 minute sliding windows. All principal components remain active, reflecting high-dimensional behaviors. Parameters: $\epsilon_C = \epsilon_A = 0.01$, $\sqrt{E}$, update rate 100 Hz, $\gamma_c = 0.001$, $\delta = 0.005$.

[Video 9]. In Fig. 4.30 the eigenvalues of the covariance matrix of the sensor values of 5 min intervals are presented. The size of the eigenvalues reflects the variance in the 15 principal components. All eigenvalues remain non-zero and they almost equally span over the interval 0 to 1. This is in contrast to the experiment in Section 4.6, where the eigenvalue spectrum broke down after some time (Fig. 4.19). We can conclude that the behaviors exhibited within 5 min keep a high dimensional structure. A more detailed view on the major two principal components is given in Fig. 4.31, now for intervals of 2 min. The first two principal components scaled by their relative variances are plotted in Fig. 4.31(b),(c). The vectors are preprocessed to have a positive sign in the first few components, in order to increase clarity. Very different major principal components are exhibited over time. However, the main eigen-modes of the physical system are frequently excited in a similar manner as observed before. The normalized eigenvalues of the covariance matrix, Fig. 4.20(a), show that at the time when theses modes occur the first mode dominates the behaviors more strongly (brighter stripes in the rows 3-15).

Let us have a look at the parameter dynamics. The controller matrix C is no longer square, since the derivatives of the sensors values are used as well. We consider a partition of C into two 15×15 square blocks, thus $C = (C^1\ C^2)$. C^1 connects joint angles with motor values. The eigenvalues of this matrix and of the linearized system matrix are displayed in Fig. 4.32. All real parts of the eigenvalues of C^1 stay close to 1 and the two smallest eigenvalues have non-zero complex parts. Nevertheless, the eigenvalues of the system matrix R have very small complex components. This shows again that the Snake robot cannot perform high frequency oscillations. The continuity preference extension is not able to suppress these complex parts because they do not yet lead to oscillations in the motor values, since they belong to the smallest eigenvalues. This is in contrast to the case without the continuity preference in Fig. 4.22, where the largest eigenvalues have large complex components.

Finally, let us consider the parameters of the system at the end of the experiment. For that we plot the matrices C, A, and S in Fig. 4.33. The controller matrix C essentially maps the joint angles and its derivatives to the motor values, which are nominal joint

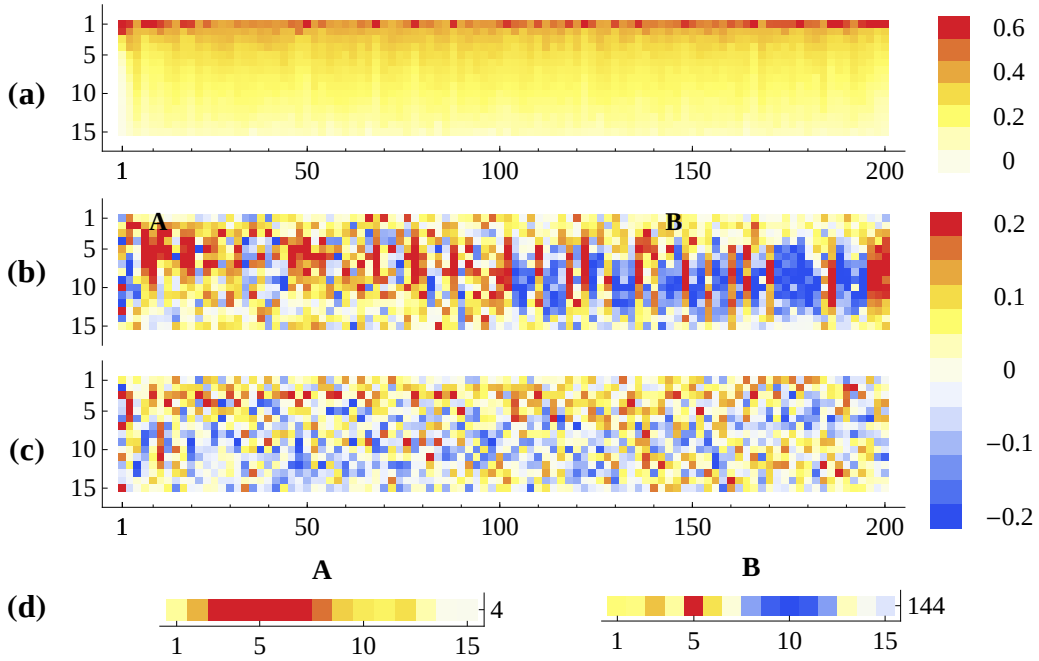


Figure 4.31: Principal components of the Snake's sensor values when using the extended world model. The time is given in minutes. The covariance matrices have been computed on the sensor data from 2 min each. **(a)** Each column shows the normalized vector of eigenvalues of the covariance matrix, i. e. $\vec{\lambda}^{\mathrm{Cov}(x)}/|\vec{\lambda}^{\mathrm{Cov}(x)}|$; **(b)** Each column shows the scaled eigenvector u^1 corresponding to the largest eigenvalue λ_1 (first row of **(a)**), $u^1_{\mathrm{Cov}(x)}\lambda_1^{\mathrm{Cov}(x)}/|\vec{\lambda}^{\mathrm{Cov}(x)}|$. **(c)** The second eigenvector scaled with the corresponding normalized eigenvalue; **(d)** Two columns of **(b)** at min 8 (**A**) and 144 (**B**) enlarged. From time to time the eigen-modes of the physical system are still excited, as marked with **A** and **B**. For more details see Fig. 4.20 (p. 85).

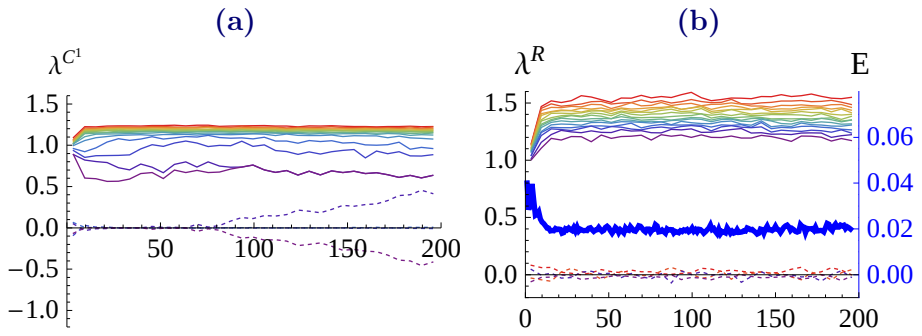


Figure 4.32: Eigenvalues of the controller matrix and linearized system matrix over time for the Snake robot. The time is given in minutes and the curves show sliding averages of 3 min intervals. **(a)** Eigenvalues of left square 15×15 block of C. The solid lines show the real part and the dashed lines show the imaginary part (only 2 are non-zero); **(b)** Eigenvalues of linearized system matrix in motor space $R = CA$ and the TLE E (**blue**, right axes). Parameters: $\epsilon_C = \epsilon_A = 0.02$, update rate $100\,Hz$, $\gamma_c = 0.001$.

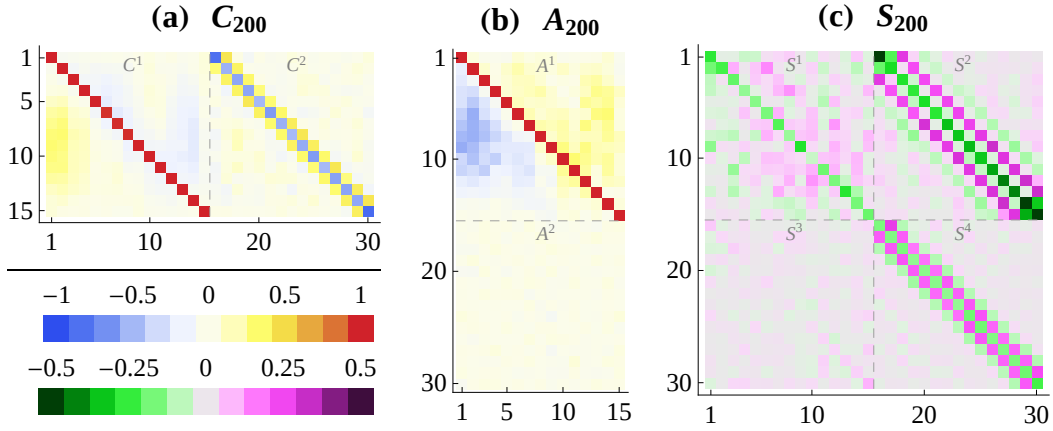


Figure 4.33: Controller and extended world model matrices at the end of the Snake robot experiment. All matrices are taken at minute 200. **(a)** Controller matrix C mapping joint angles (first 15) and their derivatives (angular velocities) (16-30) to motor values (neglecting the non-linearity). A clear separation of the joint angles (left square block C^1) and derivatives (right square block C^2) takes place. Besides the strong diagonal elements of C^1, there are long range connections (yellow and blue areas). The derivatives (C^2) are actually negatively coupled to the motor values, reflecting a damping; **(b)** World model matrix A. It maps motor values to future sensor values. The lower square block (A^2) is essentially zero because there is no correspondence between motor values (nominal joint angles) and angular velocities; **(c)** World model matrix S (extension of the original model). It maps actual sensor values to future sensor values. Note the different color code to account for the smaller elements due to the discount.

angles (neglecting the bias term and the non-linearity in Eq. (4.18)). As above we consider a partition of C into two 15×15 blocks, i. e. $C = (C^1\ C^2)$. Thus, C^1 maps joint angles to nominal joint angles and shows mainly a diagonal structure as expected. Connections between neighboring joints (secondary diagonals) are negative, but very weak. This corresponds to a zig-zag shape of the robot. Moving in such a zig-zag pattern reduces the inertial effects, because the movements are locally compensated. In contrast, imagine that a joint in the center of the Snake robot is controlled to change its angle independent of the other joints. Then the entire body has to be moved which causes a large inertial torque. There are also long range connections formed, as indicated by the pale yellow and blue areas at both sides of C^1 in Fig. 4.33(a). These connections lead the whole body movements indicated by the principal components in Fig. 4.31(d). The second block shows that the angular velocities (sensors 16-30) are coupled inhibitory to the belonging joints (diagonal of C^2). This reflects a damping of the movement speed, which is not only a consequence of the continuity preference extension of the controller, but also of the fact that high angular velocities quickly lead to the saturation region of the controller neurons at high joint angles. However, the angular velocities of the left and right neighbors have an excitatory influence. Note that behavioral changes in such highly physically coupled systems can be achieved with small changes in the controller connections.

The world model matrix A, see Fig. 4.33(b), maps motor values to future sensor values and thus has the dimension 30×15. However, the lower square block (A^2) is essentially zero because there is no direct correspondence between motor values and joint velocities. The new world model matrix S, Eq. (4.91), represents the learned mapping from current sensor values to future sensor values, see Fig. 4.33(c). Its elements are much smaller because of the bias towards self-induced observations realized by the discount factor $\delta = 0.005$, Eq. (4.108). Let us consider the matrix in four 15×15 blocks namely $S^1, \ldots, S^4$ as indicated by the dashed lines in the plot. The diagonal elements of S^1 have small negative values which is due to the following relationship: The new angular position of a joint is the desired position (given by matrix A) minus some fraction of the old position (negative values in S^1) due to inertia. The top-right sub-matrix S^2 represents the mapping from angular joint velocities to angular positions. This is basically determined by the C^2. Interestingly, not the neighbors, but the second next neighbors are coupled inhibitory. This is due to the physical interaction of the joints. If one joint increases its angle, then the second next joint is pulled, such that its angle decreases as illustrated in Fig. 4.34.

4.8.8 Summary

To summarize, the extended world model leads to a decreased prediction error, resulting in a smooth and stable parameter dynamics in the here considered high-dimensional Snake robot. The bias towards self-induced interpretation of the observed sensor values which was implemented into the extended world model, has shown to be effective. This was reflected by the large diagonal elements of the A matrix. Nevertheless, the inertial effects

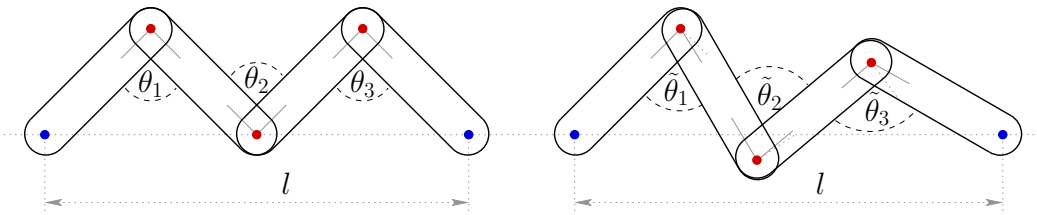


Figure 4.34: Illustration of physical joint interactions of the Snake robot. We consider 4 segments within a longer Snake robot and assume that the endpoints (blue dots) of the considered part of the robot are fixed, reflecting the inertial effects of the following segments. If the angle of the first considered joint (θ_1) is decreased ($\tilde{\theta}_1 < \theta_1$) then the third joint angle increases ($\tilde{\theta}_3 > \theta_3$) whereas the second joint stays almost the same. Thus, a change in one joint angle effects the second next joints in the opposite direction. Note that the joint sensor values are actually $\pi - \theta_i$ (0 for a straight joint), see Section 2.2.7.

and intrinsic joint interaction of the physical system were captured by the additional weight matrix S of the world model. The extension of the controller to prefer continuous motor values has shown the intended effect and no high-frequency oscillations have been exhibited. We also demonstrated that due to the extended world model it is possible to use additional sensors, like the angular velocity sensors used in the previous experiment. The detailed analysis of the controller and world model structure has shown that the additional sensors have been integrated into the sensorimotor dynamics in a reasonable fashion. We also observed how the morphology of the physical body shapes the controller, such that the eigenmodes of system are excited, but also a lot of other behaviors are shown.

4.9 Discussion

In this chapter we formulated the sensorimotor loop as a dynamical system in the multidimensional case and derived the update rules for the controller and the world model parameters following the homeokinetic principle. These are given by the gradient descent on the time-loop error (cf. Sections 4.1.1 and 4.1.5). The control algorithm does not optimize a specific goal, but rather requires, quite generally, that the sensorimotor dynamics is kept at the border of stability. In more intuitive terms the application of this principle to autonomous robots can be phrased as the aim to stay active and act predictably. We applied several regularization mechanisms to obtain a robust and practically usable control algorithm. This was demonstrated by experiments with simulated robots as well as with a real robot, including a driving robot, the Rocking Stamper, the Spherical robot and the Snake robot. In our group many more robots have been controlled with the homeokinetic controller, some of which are depicted in Fig. 4.35. Even though all

Figure 4.35: Other robots that have been controlled by the homeokinetic controller. **Left:** Pioneer robot [98] *playing* with a ball. Experiment conducted by Ralf Der and René Liebscher; **Center:** A simulated dog robot is learning to move; **Right:** Simulated humanoid robot performing athletic exercises. Videos can be found at [47].

of the agents are completely different, the same controller[6] produces very different and highly body and environment related behaviors, see also our collection of videos [47]. In most cases we observe smooth trajectories and a dynamics that exploits the morphology of the particular body. This is especially prominent in the experiments with the Barrel and the Spherical robot, cf. Sections 4.4.1 and 4.4.2 where different highly coordinated rolling modes are exhibited. Also the application to the Rocking Stamper resulted in the exploitation of the eigenfrequencies of the body and showed that the controller remains highly plastic and adapts constantly to new situations. This was demonstrated by changing the sensor configuration or breaking a sensor during the experiment, which led to a quick readaptation, such that the robot was, after a short time, again vividly interacting with its environment.

Interestingly, the learning dynamics based on the time-loop error does not converge to a fixed pointed but rather stays on a transient all the time, such that the constitutes of the dynamical system, meaning the controller and the adaptive world model change constantly. This can result in a systematic sweeping of the behavior space as we demonstrated using different robots. The concomitant learning of the world model and the controller can lead to an incomplete sampling of the behavior space and thus to a limited scope of the world model. We elaborated on the problems of model learning when sub-dimensional modes are persistent for longer times. The observed deprivation of the world model was called "cognitive deprivation" and was analytically proven. Quite generally, systems where both controller and world model have to be acquired at the same time face the so called cognitive bootstrapping problem or learning paradox [53], which might be phrased as the problem that in order to improve the understanding of the world one has to act, but in order to know how to act one needs to have already an understanding of the world. We show analytically that the homeokinetic controller solves the bootstrapping problem by producing explorative actions pointing into unexplored subspaces. The main result proves

[6]The dimensionality, implying the number of input and output neurons, is of course different.

that all subspaces of the sensor-action space are explored by keeping all eigenvalues of the Jacobian matrix high.

The homeokinetic control can be interpreted as an implementation of the ideas of cognitive embodied systems proposed by Rolf Pfeifer [122, 124] and has a promising approach for developmental robotics [20]. It is the first system that has been shown to generate such a variety of coordinated behaviors in physically complex robotic systems, from a single unspecific principle. Let us compare the results to related efforts in evolutionary robotics [104]. The creatures of the Framestick world [78], for instance, show that highly coordinated behaviors in dynamically complex systems can be evolved using artificial evolution. The main difference is that these controller are designed, through the optimization of a specific fitness function, to show a specific behavior and perform only in a narrow range of environmental conditions. Another related study is the "playground experiment" [111]. Using an artificial intrinsic motivation a Sony Aibo [172] explores its surrounding and tests various actions in a fully self-motivated way. Evaluating it critically we find that the action space is a very low-dimensional discrete set and the behaviors consist of predefined actions without dynamical complexity. Other approaches in developmental robotics using reinforcement learning [161] and intrinsic motivation [65, 88] have much longer learning times compared to our approach and suffer from the curse of dimensionality in complex physical systems.

The homeokinetic controller in the original formulation is restricted to the use of specific proprioceptive sensors, which measure the same physical quantity that is controlled by the motor actions. Most robotic devices have many more sensors than motors, thinking of distance sensors, tactile sensors or even visual input. Ideally, we want to supply a range of different sensors to the controller and hope that the most appropriate ones are automatically selected. To pursue this we presented a new formulation of the dynamics in motor space, which is more suitable for systems with more sensors than motors, compared to the original formulation in sensor space. However, the simple internal world model cannot account for a complicated sensor setting and thus produces infeasible prediction errors. Another reason to improve the internal world model was given in Section 4.6, where a snake-like robot with many degrees of freedom and strong inertial effects was considered. Due to the insufficiency of the internal world model, we observed a reduction of the behavioral richness to a few low-dimensional modes. We proposed an extended world model that uses beside the motor values also the sensor values for its prediction. This raised the inherent problem of the estimation of the causal effects of the robot's actions from the sensor values. We find an ambiguity between self-induced and environmentally-induced changes in the sensor value stream. However, the actual effects of the actions have to be properly extracted in order to regulate the controller parameters to the effective regime. We proposed a rather general solution by specifying a bias towards self-induced observations and provided experimental evidence for its functional efficiency. A computationally optimized variation uses a discount in the learning rule and is much simpler to implement.

We identified another shortcoming of the unmodified homeokinetic controller, namely that often high-frequency oscillations are exhibited. However, the influence of the self-organization process requires intricate mechanisms. We achieved a general formulation to integrate additional goals or constraints in terms of error functions into the learning dynamics, without destroying the self-organization process. This is a fundamental step towards guided self-organization, which will be the subject of the next chapter. As a proof of concept we formulated the preference for low frequency oscillations and applied it to the previously floundering Snake robot. In this experiment we also successfully applied the advanced sensor setup and the hierarchical world model. To conclude, it is now possible to control high-dimensional system with strong inertial effects and more complex sensor setups.

Chapter 5

Guided Self-Organization

> Conquest is easy. Control is not.
>
> *Captain Kirk in "Mirror, Mirror"*

In this chapter we will focus on goal-directed behaviors. Usually, this is achieved by directly optimizing the parameters of the control program to achieve a specific goal, for example via reinforcement learning [161] or via evolutionary algorithms [104]. We will follow a different path, namely to combine the self-organizing control with external drives. For this combination we coined the term *guided self-organization* [90] and Mikhail Prokopenko phrased it in a general perspective [130]. Before the term was only rarely used e. g. in nano technology [30] or swarm robotics [136]. The core idea of guided self-organization is to combine goal-oriented design with self-organized development to obtain a system which unites benefits of both. Compared to traditional design which operates mostly in an all-or-nothing fashion, self-organizing systems tend to have a high tolerance against failures and degrade gracefully, rather than catastrophically.

What can we expect from a *guided homeokinetic controller*? We have seen in Chapter 4 that a variety of behaviors emerged solely from the principle of homeokinesis. The process of self-organization has quickly structured the vast space of action sequences into a set of behaviors that show a coherent sensorimotor dynamics of the particular robot in its environment. The hope is now to shape the self-organization process to produce specific behaviors within a short time. Part of the idea is to channel the exploration of the homeokinetic controller around certain desired behaviors, such that modes can be found which fit even better to the particular robotic device. This is especially important in high-dimensional systems where the self-organized search for behaviors can take a long time and it is not guaranteed that all possible behaviors are visited in finite time. With additional soft constraints we can expect to achieve potentially useful behaviors in high-dimensional robotic systems.

What are the challenges of guided self-organization? Let us compare self-organization processes with technical design[1]. Self-organization is the evolution of a system into an organized form in the absence of specific external pressures, whereas a technically designed system has a given, usually rigid, structure without reorganization. In [129] Mikhail Prokopenko emphasizes the differences as follows:

> In fact, one may argue that the notions of design and self-organization are contradictory: the former approach often assumes a methodical step-by-step planning process with predictable outcomes, whereas the latter involves non-deterministic spontaneous dynamics with emergent features.

This should not give the impression that self-organizing patterns are completely unpredictable, but they have at least one unpredictable dimension.

On the methodological level we investigate three mechanisms of guidance. The first one allows for the incorporation of supervised learning signals, e. g. specific nominal motor commands, which we call *teaching signals*. This method is made possible through our study of integrating additional error functions into the learning dynamics, cf. Section 4.7.1. Using a distal learning [74] setup we study the use of teaching signals in terms of sensor values. The second mechanism allow for the formulation of motor couplings. This will be proven to be an effective and simple way to introduce constraints into the system and facilitate the unsupervised development of specific behaviors. The third mechanism for guiding the self-organization is discussed in Section 5.3 and uses online reward signals to shape the emerging behaviors.

5.1 Guiding with Teaching

In this section we investigate different ways to guide the homeokinetic controller with supervised learning signals. We call them *teaching signals*, because they are given externally with respect to the self-organizing system. These signals can be incorporated using an additional error function. An important prerequisite for this method was presented in Section 4.7.1, where a natural gradient was used to maintain the balance between internal self-regulation dynamics and external goal-oriented drives.

One way of influencing the behavior of the robot is to directly provide wanted motor patterns, i. e. *motor teaching signals*. The reader might wonder why this is of practical value since it requires precise information about the motor setup and its dynamics. Furthermore, if this information is already present one could use it to directly control the robot. Nevertheless, there are two aspects which make it indeed useful. The explorative character of the homeokinetic controller can be used to explore around the given motor pattern and thus find a more suitable behavior for the particular situation. More importantly, the

[1] Design is here understood in the engineering sense to design for a particular function.

motor teaching mechanism can be used by higher level guiding mechanisms as discussed in Section 5.2, that are again unsupervised and require much less information about the dynamics of the robotic device.

Thus, we will first investigate direct motor teaching signals and then consider sensor teaching signals.

5.1.1 Direct Motor Teaching

In this section we will show how to incorporate motor teaching signals into the self-organizing control. For that we define an additional error function which penalizes the misfit

$$\eta_t^S = y_t^S - y_t \tag{5.1}$$

between teaching motor values y_t^S and actual motor values y_t. Similarly to the prediction error for the world model, Eq. (4.20), we find

$$E^S = {\eta_t^S}^\top \eta_t^S \,. \tag{5.2}$$

Using the gradient descent we find the additional update for the controller matrix C (Eq. (4.18)) as

$$\Delta C^S = -\frac{\partial E^S}{\partial C} = \left(\eta_t^S \circ g'\right) x_t^\top \,, \tag{5.3}$$

where $g' = g'(Cx_t + h)$. The update for h is similarly obtained and reads

$$\Delta h^S = -\frac{\partial E^S}{\partial h} = \eta_t^S \circ g' \,. \tag{5.4}$$

These additional terms are integrated into the final learning rule using the technique for integrating additional error terms, cf. Section 4.7.1. Using Eqs. (4.73, 4.74) we find for the formulation in motor space the following total update rules for C and h:

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} + \gamma_S (JJ^\top)^{-1} \Delta C^S \,, \tag{5.5}$$

$$\frac{1}{\epsilon_C}\Delta h = -\frac{\partial E}{\partial h} + \gamma_S (JJ^\top)^{-1} \Delta h^S \,. \tag{5.6}$$

The guidance factor γ_S regulates the strength of the additional drive and has to be determined empirically. A small value of γ_S leads to a small influence of the teaching signal and results in a behavior that is mostly dominated by the original homeokinetic controller. For large values of γ_S the teaching signals are followed narrowly and few exploratory actions are performed.

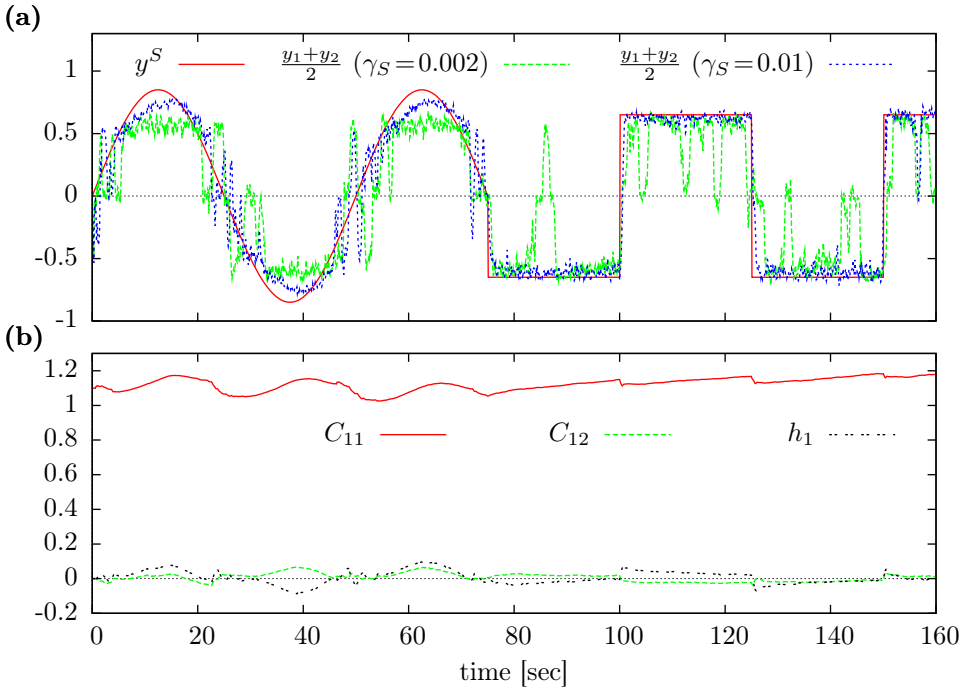


Figure 5.1: TwoWheeled robot controlled with homeokinetic controller and direct motor teaching signals. For different values of γ_S (guidance factor) the teaching signals y^S (Eq. (5.7)) are followed more or less accurately. **(a)** Nominal motor value y^S (identical in both components) and average motor value $\frac{1}{2}(y_1 + y_2)$ for different values of the guidance factor γ_S; **(b)** Controller parameters of the first motor neuron for $\gamma_S = 0.01$. Parameters: $\epsilon_C = \epsilon_A = 0.1$, update rate 100 Hz.

Experiment

Let us evaluate this mechanism of guidance by using the TwoWheeled robot, see Section 2.2.1. We want to show that teaching signals can be used to specify a certain behavior and that the influence of the teaching can be conveniently adjusted using γ_S. For that let us consider two motor teaching signals which are subsequently used. First the nominal motor values are given by a sine wave and then by a rectangular function with the same value for both motors, i. e.

$$(y_t^S)_i = \begin{cases} 0.85 \cdot \sin(\omega t) & t < 75 \\ 0.65 \cdot \operatorname{sgn}(\sin(\omega t)) & \text{otherwise}, \end{cases} \tag{5.7}$$

with $i = 1, 2$ and $\omega = 2\pi/50$. The choice of the teaching signal has no deeper meaning. Note that the nominal motor values should not be too large because otherwise the controller is driven into the saturation region of the motor neurons, see Section 3.8. As we found in Section 3.7, the fixed point of sensor dynamics in the toy example is at $y \approx \pm 0.65$. This is a good mean teaching signal size, which was also used in Eq. (5.7). As a rule of thumb we recommend confining the motor teaching values to the interval $[-0.85, 0.85]$.

In Fig. 5.1 the produced motor values and the parameter dynamics are displayed for different values of the guidance factor (γ_S). For a low value of γ_S the desired behavior is only followed by trend, whereas for higher values, e. g. $\gamma_S = 0.01$, the robot mostly follows the given teaching value with occasional exploratory interruptions. These interruptions cause the robot, for example, to move in a curved fashion instead of strictly driving in a straight line as the teaching signals dictate. The exploration around the teaching signals might be useful to find modes which are better predictable or more active. It was shown in the theory of cognitive deprivation (Section 4.5) that a long performance of a single low-dimensional behavior can lead to the inaccuracy of the adaptive world model. Thus, the explorative actions can supply the adaptive world model with necessary sensation-actions pairs for complete learning.

The experiment demonstrated that motor teaching signals can be used to achieve a specific behavior. This result is not very surprising, because the system is very simple and the taught behavior did not conflict with the homeokinetic principle (sensitive and predictable). However, it served as a proof of principle and showed that the balance between taught behavior and remaining self-organized behavior can be adjusted using a single parameter.

5.1.2 Direct Sensor Teaching

In this section we transfer the direct teaching paradigm from motor teaching signals (Section 5.1.1) to sensor teaching signals. This is a useful way of teaching because desired sensor values can be more easily obtained than motor values, for instance by passively moving the robot, or parts of the robot, assuming that proprioceptive sensors are present. This kind of teaching is also commonly used when humans learn a new skill, e. g. think of a tennis trainer that teaches a new stroke by moving the arm and the racket of the learner. Thus, a series of nominal sensations can be acquired that can serve as teaching signals. Setups where the desired outputs are provided in a different domain than the actual controller outputs are called *distal learning* [50, 74, 155]. Usually a forward model is learned that maps actions to sensations (or more generally to the space of the desired output signals). Then the mismatch between a desired and occurred sensation can be backpropagated to obtain the required change of action. The backpropagation can also be done using an inversion of the forward model. Another option is the use of a backward model, which learns the mapping from sensations to actions. The main difference between backward models and inverted forward models is the handling of noisy subspaces. The inverted forward model will expand these subspaces, whereas backward models will shrink them.

In our case a forward model is already at hand, namely, it is given by the internal world model, see Eq. (3.16). Instead of a backpropagation we can invert the world model directly as done in Sections 4.1.2 and 4.1.6. Let the sensor teaching signal be given by x_t^D. The

distal learning error is the misfit ξ_t^D between desired sensations x_t^D and predicted sensations $\tilde{x}_t$ (Eq. (4.2)), thus

$$\xi_t^D = x_t^D - \tilde{x}_t \,. \tag{5.8}$$

From Eq. (5.8) and using the world model M (Eq. (4.2)) we can calculate a misfit η_t^S in motor space that satisfies

$$x_t^D = \tilde{x}_t + \xi_t^D \stackrel{!}{=} M(x_{t-1}, y_{t-1} + \eta_t^S) \,. \tag{5.9}$$

Using a linearization we can write

$$\xi_t^D = M'_y(x_{t-1}, y_{t-1})\eta_t^S + O\left((\eta_t^S)^2\right) \,, \tag{5.10}$$

where M'_y denotes the derivative of M with respect to y. Using the pseudoinverse ${M'_y}^+$ of the derivative of the world model M we can obtain η_t^S in a linearized way as

$$\eta_t^S = {M'_y}^+(x_{t-1}, y_{t-1})\xi_t^D \,. \tag{5.11}$$

Since our particular implementation of the world model is linear ($M(x_t, y_t) = Ay_t + b$, cf. Eq. (4.19)) we can obtain the exact formula for η_t^S as

$$\eta_t^S = A^+ \xi_t^D \,. \tag{5.12}$$

Alternatively, we can calculate the motor teaching signal as $y_t^S = A^+(x_t^D - b)$, which makes it easier to confine them to an appropriate interval, see Section 5.1.1. Now the update formulas for C and h from the direct motor teaching setup are used, cf. Eqs. (5.3, 5.4). Note that the extended world model (Eq. (4.91)) used in Sections 4.8.4 and 4.8.5 has the same derivative with respect to y, such that the formulas remain identical. Analogously to Eqs. (5.5, 5.6) and by using Eq. (5.11) we find the following update rules for C and h:

$$\frac{1}{\epsilon_C}\Delta C = -\frac{\partial E}{\partial C} + \gamma_D \left(JJ^\top\right)^{-1} \left(\left(A^+\xi_t^D\right) \circ g'\right) x_{t-1}^\top \,, \tag{5.13}$$

$$\frac{1}{\epsilon_C}\Delta h = -\frac{\partial E}{\partial h} + \gamma_D \left(JJ^\top\right)^{-1} \left(\left(A^+\xi_t^D\right) \circ g'\right) \,, \tag{5.14}$$

where g' is taken at $g'(Cx_{t-1} + h)$. The guidance factor is here called γ_D and regulates the strength of the additional drive.

The application to the TwoWheeled driving robot, as it was done in the previous section, is trivial since the world model consists essentially of a unit matrix. Among the previously considered robots the Spherical robot has a non-trivial relation between sensor and motor values, hence we will use it in the following experiment.

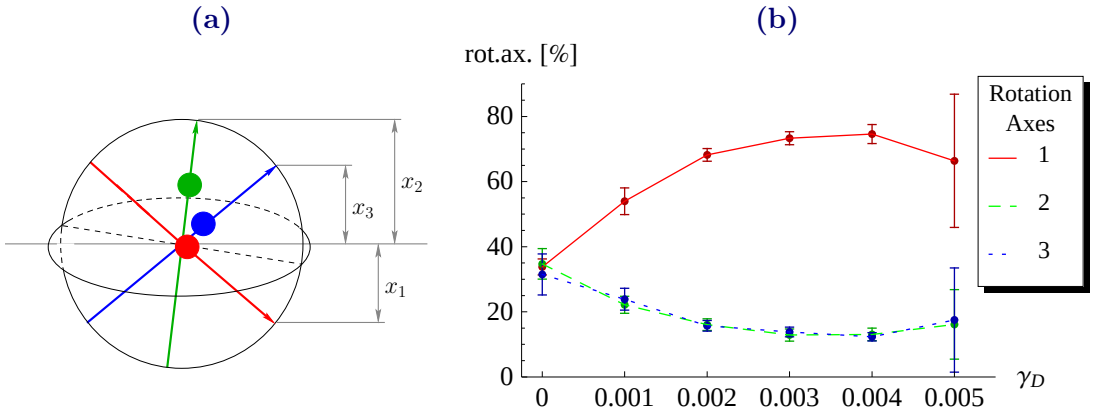


Figure 5.2: SPHERICAL robot and its behavior in a distal learning setup. **(a)** Illustration of the robot with its sensor values; **(b)** Behavior for the distal learning task, Eq. (5.15). The plot shows the percentage of rotation around each of the axes for different values of the guidance factor γ_D (no teaching for $\gamma_D = 0$). The rotation around the red axis is clearly preferred for non-zero γ_D. The mean and standard deviation of 10 runs each 60 min long, excluding the first 10 min (initial transient, no teaching). Parameters: $\epsilon_C = \epsilon_A = 0.1$, update rate 100 Hz.

Experiment

In this experiment we use the SPHERICAL robot, as described in Section 2.2.5. For each axis we have one sensor value that is the z-component of the axis vector in world coordinates which is illustrated in Fig. 5.2(a). We use the world model extension as proposed in Section 4.8 with the bias towards self-induced interpretation of sensor values, see Section 4.8.4.

The objective of this experiment is to show that a simple teaching signal in terms of sensor values can be used to effectively guide the behavior of the SPHERICAL robot towards rotations around the first internal axis. To achieve that, we use the distal learning task that requests a low value of the first sensor. More precisely,

$$x_t^D = \begin{pmatrix} 0 \\ (x_t)_2 \\ (x_t)_3 \end{pmatrix} , \tag{5.15}$$

thus only the first component of the sensor value produces an error signal. We expect that the robot will preferably rotate around the first axis since this keeps the first sensor value low, see Fig. 5.2(a). To evaluate, we performed for different values of the guidance factor 10 runs each with a duration of 60 min and the robot on the level ground. The distal learning setup requires a well trained world model. Therefore no teaching signal was provided during the first 10 min of the experiment. As a descriptive measure of the behavior, we used the index of the internal axis around which the highest rotational velocity

was measured at each moment of time. Figure 5.2(b) displays for different values of the guidance factor (γ_D) and for each of the axes the percentage of time it was the major axis of rotation. Without teaching there is no preferred axis of rotation. With distal learning the robot shows a significant preference for a rotation around the first axis up to 75 %. For overly strong teaching, a large variance in the performance occurs. This is caused by a too strong influence of the teaching signal on the learning dynamics. Remember that the rolling modes can emerge due to the fine regulation of the sensorimotor loop to the working regime of the homeokinetic controller, which cannot be maintained for large values of γ_D.

Why is it not possible with this method to force the controller to stay in the rotational mode around the first axis? The answer is rather intuitive: When the robot is in this rotational mode the teaching signal is negligible. However, the controller's sensitization will increase the impact of the first sensor, such that the mode becomes unstable again.

To summarize, the direct teaching mechanism proposed in Section 5.1.1 allows us to specify motor patterns that are more or less closely followed, depending on the strength of integration of the additional force into the learning dynamics. In this section we considered sensor teaching signals that were transformed into motor teaching signals using the internal world model. We have shown that the Spherical robot with the homeokinetic controller can be guided to locomote mostly around one particular axis, by specifying a constant sensor teaching signal at one of the sensors. The supervised learning in terms of sensor signals is one step in the direction of imitation learning. Imitation learning [142] deals with how an autonomous robot can acquire behaviors from other agents or from humans. In this setting the robot can typically perceive the movement via a camera or perhaps via its joint sensors if the demonstrator was moving the parts of the robot's body. In the latter case the methods proposed here can be directly used. If only visual information is available the correspondence problem concerning the mismatch between the teacher's body and the robot's body needs to be resolved [34, 49, 134], which is not discussed here.

5.2 Guiding with Cross-Motor Teaching

Many robotic systems and animals have a symmetric structure, e. g. a sagittal symmetry between the left and right sides of the body or a number of identical legs and so forth. We will now discuss how to provide the control system with information about the body morphology and symmetries of the desired behaviors to influence the self-organization process. For that we will use soft constraints which reduce the effective dimension of the sensorimotor dynamics and thus guide the self-organization along a sub-space of the original control problem. In biological systems these kinds of constraints are often implemented on a low level of neuronal circuitry, e. g. pairs of antagonistic muscles are connected such that activity of one inhibits activity of the other on the level of α-motor neurons and inter-neurons in the spinal cord [118].

Inspired by this regulation we use the motor values of one controller neuron as a teaching signal for another controller neuron. We call this method *cross-motor teaching*. To describe which controller neuron receives a teaching signal from which we use *cross-motor connections*. Note that despite the use of 'teaching signals' the algorithms is completely unsupervised, because the signals are generated internally.

To introduce this method we will first consider pairwise symmetries and then generalize to permutation relations. Finally, we will discuss how arbitrary cross-motor teaching can be implemented to reduce the effective dimensionality of the search space.

5.2.1 Enforcing Pairwise Symmetries

First we want to influence the controller to prefer a mirror-symmetry in the motor patterns. For that, pairs of motors are selected which are supposed to be synchronous. Let us consider a particular pair of motors (r, s). We place a cross-motor connection from r to s and back. Thus, the motor r is driven towards the value of motor s and vice versa. Hence, the (internal) teaching signal is

$$(y_t^S)_r = (y_t)_s \qquad \text{and} \qquad (y_t^S)_s = (y_t)_r \,, \tag{5.16}$$

which is used in Eqs. (5.2–5.6).

Likewise, an anti-symmetric relation can be expressed with $(y_t^S)_r = -(y_t)_s$ and vice versa. In this simple setup the cross-motor connections have either a positive or negative sign. For those motors i that are not part of a pair we need to set $(y_t^S)_i = (y_t)_i$, in order to produce no error signal in the framework of direct motor teaching, Section 5.1.1. More complex connection types are discussed later.

Experiment

To illustrate the concept we will consider the TwoWheeled robot, see Section 2.2.1. The robot has two motors actuated according to y_1 and y_2. Let the preferred behavior be to drive straight. This can be obtained by defining a symmetric relation between both motors following Eq. (5.16), i. e.

$$(y_t^S)_1 = (y_t)_2 \qquad \text{and} \qquad (y_t^S)_2 = (y_t)_1 \,. \tag{5.17}$$

For experimental evaluation we placed the robot in an environment cluttered with obstacles. We performed for different values of the factor γ_S five runs of 20 min length. In order to quantify the influence of the cross-motor teaching we record the trajectory, the linear velocity, and the angular velocity of the robot. We expect an increase in linear velocity because the robot is to move straight instead of turning. For the same reason the angular

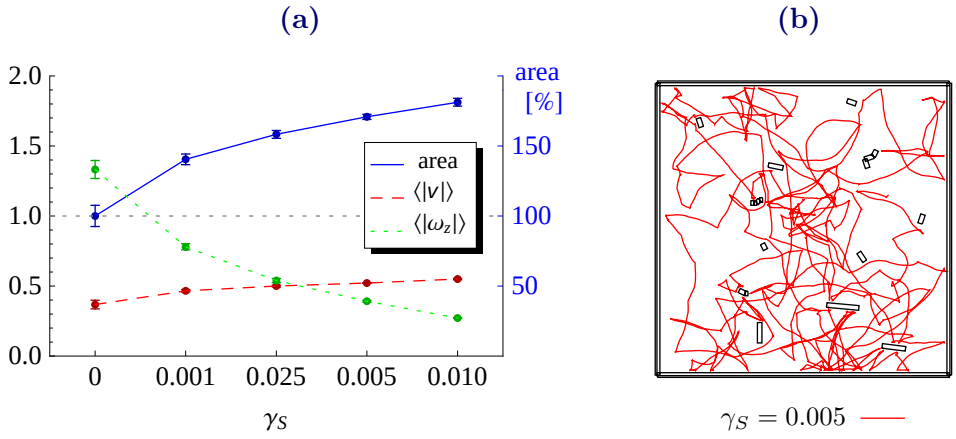


Figure 5.3: Behavior of the TwoWheeled robot with enforced symmetry of the two motors. **(a)** Mean and standard deviation (of 5 runs each 20 min) of the area coverage (**area**), the average velocity ($\langle \boldsymbol{v} \rangle$), and the average turning velocity ($\langle |\omega_z| \rangle$) for different values of the guidance factor γ_S. Area coverage (box counting method) is given in percent of the case without influence ($\gamma_S = 0$) (**right axis**). The robot is driving straighter and its trajectory covers more area for larger γ_S; **(b)** An example trajectory of the robot with guidance factor $\gamma_S = 0.005$. Parameters: $\epsilon_C = \epsilon_A = 0.01$, update rate 100 Hz.

velocity should go down. In Fig. 5.3 two sample trajectories and the behavioral quantifications are plotted. Additionally, we plot the relative area coverage[2], which reflects how much more area of the environment was covered by the robot with cross-motor teaching compared to the original homeokinetic controller. As expected, the robot shows a distinct decrease in mean turning velocity and a higher area coverage with increasing values of the guidance factor. Note that the robot is still performing turns and drives both backwards and forwards and does not get stuck at the walls, as seen in the trajectory in Fig. 5.3(b). The properties of the homeokinetic controller, such as sensitivity and exploration, remain.

We have seen that a pairwise cross-motor teaching can be used to guide the self-organizing control to drive mostly straight in the TwoWheeled robot. The strength of this preference can be adjusted by the guidance factor. The algorithm is unsupervised and the only specific information that is given is the pair of motors to be synchronized. Let us now proceed to a more flexible specification of cross-motor connections in order to deal with more complex situations.

5.2.2 Permutation Relations

Now we want to consider a more general cross-motor connection setup where each motor has one incoming and one outgoing connection, such that there is still only one teaching

[2]The area coverage of the trajectory is calculated using a box-counting method.

signal per motor. The cross-motor connections can be described by a permutation π_m of the m motors that assigns each motor a source of teaching input. Additionally a sign function $\chi : \{1, \ldots, m\} \to \{-1, 1\}$ defines whether the motors are supposed to be symmetric or antisymmetric. The teaching signal is then given by

$$\left(y_t^S\right)_i = \chi\left(\pi_m(i)\right)\left(y_t\right)_{\pi_m(i)} \qquad \text{for } i = 1, \ldots, m. \tag{5.18}$$

The symmetric setup (Eq. (5.16)) is of course a special case of this notation. With a cyclic schema of connections also a group of motors can be synchronized. In the following experiment we use a rotation-symmetric configuration to show that a high-dimensional chain-like robot can quickly develop a locomotion behavior.

Experiment

Let us now consider a more complex robot – the Armband. This robot consists of a sequence of flat segments placed in a ring-like configuration, where subsequent segments are connected by motor-operated hinge joints. As a result we obtain a robot with the appearance of a bracelet or chain, see Fig. 5.4(a). A detailed description of the simulated hardware is located in Section 2.2.8. Since the robot is symmetric there is by construction no preferred direction of motion, meaning that the robot controlled by the homeokinetic controller will equally probable move forward and backward. The robot cannot turn or move sideways, but it can produce a variety of postures and locomotion patterns.

With the method of cross-motor teaching we can select different symmetries, such that the robot is more likely to perform a directed motion. For that we define the permutation, (used in Eq. (5.18)), as

$$\pi_m(i) = (i + k + \lfloor m/2 \rfloor) \mod m, \tag{5.19}$$

where $k \in \{-1, 0, 1\}$. We will only use positive connections, such that the sign function is not required. Thus, the teaching signals are

$$\left(y_t^S\right)_i = \left(y_t\right)_{(i+k+\lfloor m/2 \rfloor) \bmod m} \qquad \text{for } i = 1, \ldots, m. \tag{5.20}$$

The choice of k depends on the desired direction of motion and on whether the number of joints m is even or odd. If m is even then $k = -1$ and $k = 1$ are used for both directions (forward or backward) and $k = 0$ represents a point symmetric connection setup. In the latter case the robot will not prefer a direction of motion and the behavior is similar to the one without guidance. For an odd value of m, which is also used here, $k = 0$ and $k = 1$ can be used for backward and forward motion. In the following experiments the robot has $m = 13$ motors. The motor connections for $k = 1$ are illustrated in Fig. 5.4(b). Each motor connection is displayed by an arrow pointing to the receiving motor. Note that the connections are directed and a motor is not teaching the same motor from which it is receiving teaching signals. For $k = 0$ all arrows in Fig. 5.4(b) are inverted, meaning that for each connection the sending and receiving motors would swap roles.

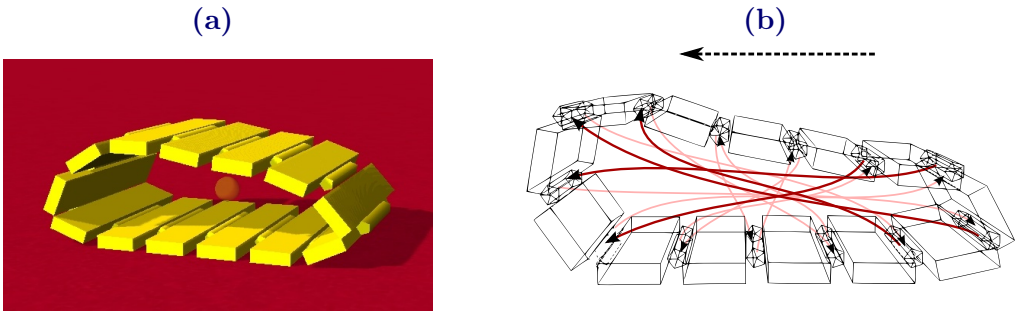


Figure 5.4: Armband robot with cross-motor connections. **(a)** Screenshot from the simulation. The transparent sphere in the center marks the center of mass of the robot; **(b)** Schematic view of the robot. The prismatic structures are hinge joints actuated by servo motors. The arrows indicate unidirectional cross-motor connections, where the head points to the receiving unit. All links are equal, but for visibility reasons only four links are drawn boldly. For this connection setup the robot preferably moves leftwards.

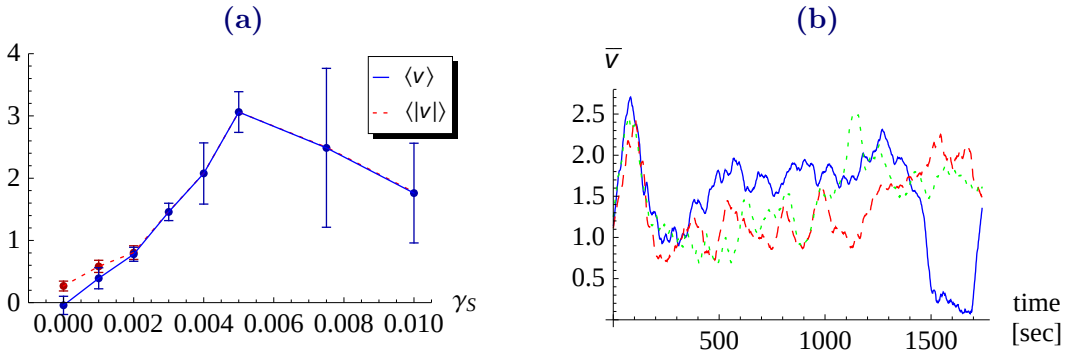


Figure 5.5: Performance of the Armband robot with constant cross-motor teaching. **(a)** Mean and standard deviation of the average velocity $\langle v \rangle$ and the average absolute velocity $\langle |v| \rangle$ of 5 runs for different value of the guidance factor γ_S; **(b)** Velocity of the robot $\overline{v}$ (averages over 1 minute sliding window) for 3 runs at $\gamma_S = 0.003$. Parameters: $k = 1$, $\epsilon_C = \epsilon_A = 0.1$, $\sqrt{E}$ (Section 4.2.4), update rate 100 Hz.

To evaluate the performance we conducted for different values of the guidance factor γ_S 5 trials each 30 min long. In a first setting the cross-motor connections were fixed ($k = 1$) for the entire duration of the experiment. We observed the formation of a locomotion behavior after a very short time. Note that this behavior requires all joints of the robot to be highly coordinated. As a quantitative measure of the performance we calculate the horizontal velocity v using the center of mass of the robot, see Fig. 5.4(a). Thus, the velocity is a scalar number and we define forward motion if $v > 0$ and backward motion if $v < 0$. In this experiment we expect the robot to move only forward, because a fixed cross-motor connection setup was used. The average velocity of the robot increased distinctively with raising guidance factors, see Fig. 5.5(a). For excessively large values of the guidance factor γ_S the velocity goes down again. This occurs for two reasons: First, the cross-motor teaching has a too strong influence on the working regime of the homeokinetic controller and second the actual motor pattern of the locomotion behavior does not perfectly obey the relations between the motor values as specified by Eq. (5.20). In order to satisfy the constraints imposed by Eq. (5.20) all motor values need to be equal, which is of course not the case in the locomotion behavior.

Without guidance the robot moves equally to both directions but with comparably low velocity. This can be seen at the mean of the absolute velocity in Fig. 5.5(a). If the value of the guidance factor is chosen conveniently, the robot moves in one direction with varying speed see Fig. 5.5(b) where 3 velocity traces are displayed. The locomotive behavior can also be seen in [Video 10] for a low value of guidance factor ($\gamma_s = 0.001$) and in [Video 11] for a medium value of guidance factor ($\gamma_s = 0.003$). We also observe a peak of high velocity after the first few minutes, which is followed by a dip before a more steady regime is attained. During this time the controller is going from a subcritical regime (at $t = 0$) to a slightly supercritical regime.

In a second setup we changed the cross-motor connections every 5 min, i. e. k was changed from 0 to 1 and back. A value of $k = 0$ should lead to a negative velocity and a $k = 1$ to a positive velocity. To study the dependence on the guidance factor and to measure the performance we use the average absolute velocity ($\langle |v| \rangle$) and the correlation of the velocity with the configuration of the cross-motor teaching ($\rho(v, k)$), see Fig. 5.6(a). Without guidance ($\gamma_S = 0$) there is, as expected, no correlation with the supposed direction of locomotion. For a range of values of the guidance factor we find a high total locomotion speed with a strong correlation to the supposed direction of motion. Note that the size of the correlation depends on the length of the intervals of one connection setting. For long intervals the correlation will approach one. In Fig. 5.6(b) the velocity of the robot is plotted for different runs with the same value of the guidance factor that was used in the previous experiment ($\gamma_S = 0.003$). We observe that the robot changes the direction of motion shortly after the configuration of couplings was changed, cf. [Video 12].

The experiment illustrates that we can achieve a specific behavior in a high-dimensional robot (here 13 DoF) by using cross-motor teaching. With a set of motor couplings the symmetry between two directions of motion was broken and a fast locomotion behavior

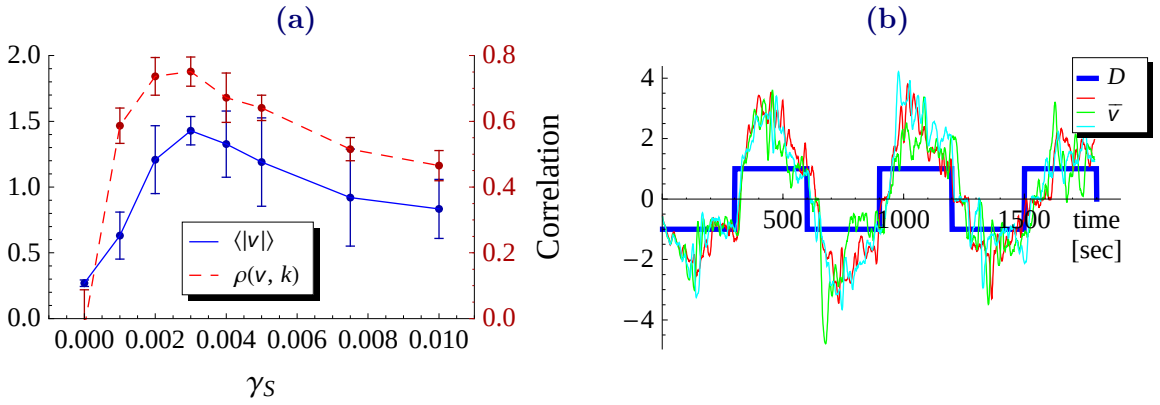


Figure 5.6: Performance of the Armband robot with changing cross-motor teaching. **(a)** Mean and standard deviation of 5 runs of the average absolute velocity $\langle|v|\rangle$ and the correlation $\rho(v,k)$ of the velocity with the configuration of the couplings for different values of the guidance factor γ_S; **(b)** Velocity (averages over 10 sec sliding windows) of the robot for 3 runs at $\gamma_S = 0.003$ and the supposed direction of motion $D = 2k - 1$ for better visibility. Parameters: $\epsilon_C = \epsilon_A = 0.1$, $\sqrt{E}$ (Section 4.2.4), update rate 100 Hz.

set in. When the connections are changed during the runtime the behavior of the robot changes reliably. In the next section we will discuss methods to implement other types of motor couplings that are not limited to a permutation setup.

5.2.3 Arbitrary Cross-Motor Teaching Configurations

In a more general form, the connection setup between motors can be expressed by a graph. If we keep the restriction of the couplings to a symmetric or antisymmetric relation between the motors then the graph can be expressed by an $(m \times m)$ connection matrix[3] $\mathcal{M}$ with elements from $\{-1, 0, 1\}$. We define the teaching signal as the average of all incoming connections, thus

$$(y_t^S)_i = \frac{1}{\sum_{j=1}^{m} |\mathcal{M}_{ij}|} \sum_{j=1}^{m} \mathcal{M}_{ij}(y_t)_j \qquad \text{for } i = 1 \ldots m. \tag{5.21}$$

The number of non-zero entries in each row must be larger than zero, i. e. $\sum_{j=1}^{m} |\mathcal{M}_{ij}| > 0$. A unit matrix for $\mathcal{M}$ represents no cross-motor teaching, since all motors receive only their own motor value. Note that using the average value of several motor values as a teaching signal has the disadvantage of reducing the activity in the sensorimotor loop.

[3]The connection matrix is also known as weight matrix and an entry in column i and row j represents the connection from motor neuron i to j. In our case a value of 0 means no connection and values of -1 and 1 stand for a negative or positive teaching connection.

This is because the average value often has usually a small magnitude, e. g. if positive and negative motor values are combined.

Another interesting way of coupling would be to enforce a certain phase-difference between two motors. However, this requires a suitable definition of a phase for each controller output, which will not be studied here.

The proposed coupling graph requires rather detailed information about the robot's morphology and its dynamics. Nevertheless, we see a large potential for this method, because the graph of connections could be learned by a higher level learning procedure. One possible way would be to use a Hebbian type learning to capture the correlations between motor channels for a particular behavior that is produced by the homeokinetic controller from time to time. Strong correlations between motor channels would suggest a cross-motor connection between them (likewise a strong anti-correlation would suggest a connection with negative sign).

The mechanism proposed here can also be transferred to sensor space using the direct sensor teaching (Section 5.1.2). The configuration is then defined in terms of cross-sensor coupling analogously to the definitions given above. This can become useful, for example if a certain behavior is demonstrated by a human operator by passively moving the robot. In the case of the Armband robot, one can easily imagine that the robot is pushed along the ground such that a locomotion pattern is formed. Based on the sensor readings, the correlations between the sensor channels can be determined and serve as a basis for the construction of a specific cross-sensor couplings.

Let us summarize: Starting from the guidance by teaching we introduced the concept of cross-motor teaching that allows us to specify abstract relations between motor channels. There are no external teaching signals required, because the motor values are used mutually as teaching signals. The only specific information put into the system is mutual teaching configuration. First we studied simple pairwise symmetric relations and shaped the behavior of the TwoWheeled robot to drive mostly straight through the cross-motor teaching between both motors. This teaching method introduces soft constraints that guide the self-organization process to a subspace of the entire sensorimotor space and therewith the effective dimension of the search space for behaviors is reduced. This was demonstrated using a high-dimensional robot – the Armband. With a simple cross-motor teaching configuration the robot developed within a short time fast locomotion behaviors from scratch. The direction of motion was altered by a change in the connection setup.

5.3 Guiding with Reward

In this section we investigate how to guide the self-organization process by providing an online reward or punishment. The starting point for the following considerations is the observation that the homeokinetic controller explores the behavioral space of the controlled

system, cf. Section 4.4. For instance, the behavior of the Barrel robot showed a systematic sweeping through the accessible frequencies of the sensor state reflected by rolling modes with different velocities (Section 4.4.1). In the case of the Spherical robot with its three dimensional motor and sensor space we also observed a sweeping through a large set of possible behaviors (Section 4.4.2). In a setup where the robot can move freely, it will exhibit different slow and fast rolling modes around different axes. An important observation is that behaviors which are well predictable will persist longer than others, cf. Section 3.5. Nevertheless, due to the exploratory character of the controller all behavioral modes are transient in their nature. In order to maintain a currently active behavior the adaptation rate of the homeokinetic controller has to be low, because otherwise it will continue to search for new behaviors. At first glance it seems to be counterintuitive that we have to reduce learning speed in order to keep a behavior, but the point is that the controller has already learned to produce the behavior when it is exhibited by the robot. It becomes also clear if we think of the transient lifetimes of well and less predictable behaviors. The homeokinetic controller gives the well predictable behavior a longer persistence, which is indeed sound. Note that this reflects a change of adaptation speed, namely that for predictable behaviors a smaller gradient of the error function occurs, which implies smaller learning steps. The idea is now to exploit this effect and regulate the speed of learning based on reward or punishment. We can expect that rewarded behaviors persists longer and punishment behaviors are left quicker.

Let us define a reward signal $r(t) \in \mathbb{R}$ for each time t, which should be considered as a punishment for negative values and as a reward for and positive values. To incorporate the reward signal we define a new error function in the following way

$$E^r = (1 - \tanh(r(t)))E, \tag{5.22}$$

where E is the usual TLE function defined in Eq. (4.12). The factor $(1 - \tanh(r(t))$ approaches zero for large positive rewards and 2 for large negative rewards as depicted in Fig. 5.7(a). The multiplication of the error function is the same as multiplying the learning rate. Thus, we could similarly write $\epsilon^r_{C,t} = (1 - \tanh(r(t)))\epsilon_C$. The construction of the new error function Eq. (5.22) follows our considerations above. For a reward $(r > 0)$ the factor $(1 - \tanh(r(t))$ is smaller than 1 and thus decreases the learning speed. For a punishment $(r < 0)$ the learning speed is increased. The hyperbolic tangent imposes constraints on effective values of $r(t)$, namely they should be in the interval $(-3, 3)$ to have a differentiable effect on the learning dynamics. However, it is not required that the interval is being utilized by a particular reward function, nor does it do harm to exceed the range.

Before we evaluate this method of guidance let us wrap up the mechanism. All behavioral modes of the non-guided homeokinetic controller are transient. The lifetime of a behavior normally depends on the size of prediction error ξ (and on the sensitivity) so that well predictable behavior remain longer. We used this to modulate the lifetimes based on reward and punishment using an additional factor in error function (Eq. (5.22)) . Behaviors

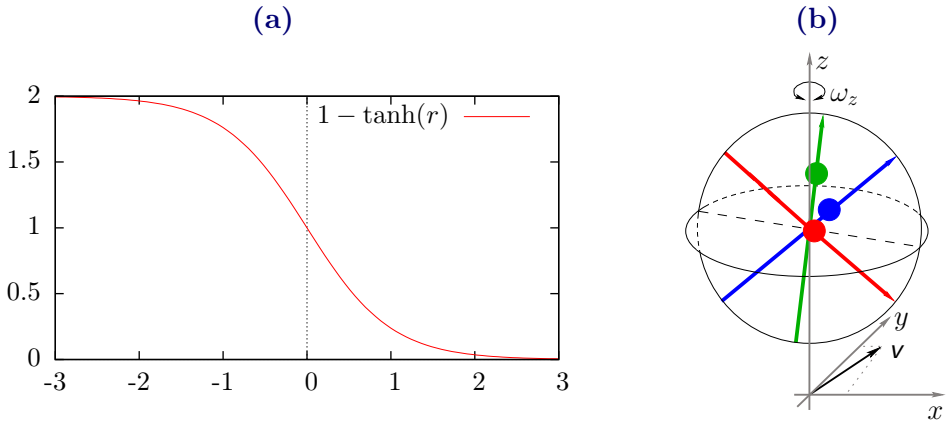


Figure 5.7: Reward dependent factor and the SPHERICAL robot in world coordinates. **(a)** Factor for the modified error function, Eq. (5.22); **(b)** SPHERICAL robot with angular velocity (ω_z) around the z-axis and velocity vector v in x-y-plane. Note that x, y, z refer here to the axes of the world coordinate system.

with small or even negative reinforcement should be left rapidly, whereas large positive reinforcement tends to increase the lifetimes, which is maximal for behaviors that are both rewarded and well predictable.

Let us now apply the mechanism to shape the behaviors of the SPHERICAL robot. In the following sections we will demonstrate two different behaviors which can be effectively guided.

5.3.1 Reinforcing Speed

In the following experiment we will use the SPHERICAL robot, as described in Section 2.2.5. One of the simplest possible desired behaviors of this robot is to move fast. Let us construct the reward function for this goal. For small velocities the reward should be negative, thus causing a stronger change of behavior, whereas larger velocities should result in a positive reward. To achieve that, the reinforcement signal can be expressed as

$$r(t) = \frac{1}{3}\|\mathsf{v}_t\| - 1\,, \tag{5.23}$$

where v_t is the velocity vector of the robot, see Fig. 5.7(b). In order to compare the results with the unguided case the reward is shifted, such that it is zero for the average velocity of normal runs. The scaling is done to keep the reward within the effective range[4].

We conducted 20 trials with the SPHERICAL robot with reinforcement and 20 trials without reinforcement, all with random initial conditions, each for 60 min in simulated real time

[4]Note that here the minimal reward is -1, so we do not use the entire range of punishment.

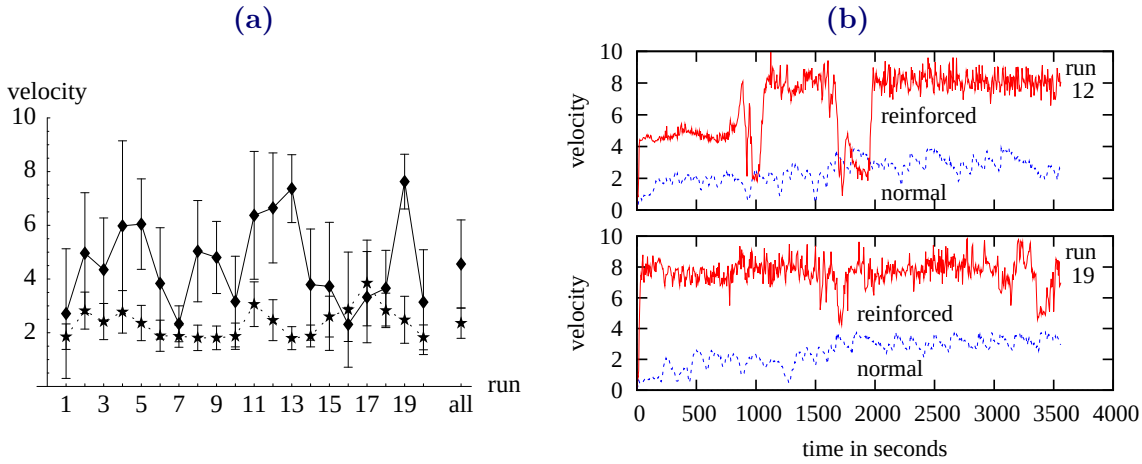


Figure 5.8: Performance of the Spherical robot rewarded for speed. **(a)** Mean and std. deviation of the velocity of the Spherical robot for 20 runs each 60 min long with (**diamonds**/**solid** line) and without (**stars**/**dotted** line) speed reinforcement. The label '**all**' denotes the mean and std. deviation over the means of all runs which is significantly ($p < 0.001$) higher for the reinforced runs; **(b)** Time course of the robot's velocity for run number 12 and 19, where **blue**/**dotted** shows the normal case and **red**/**solid** line shows the reinforced case.

on a flat surface without obstacles. The robot also experiences rolling friction, so that fast rolling really requires constant acceleration. In Fig. 5.8 the mean velocity (measured at the center of the robot) for each simulation is plotted and the velocity trace of the robot for two reinforced and two normal runs are displayed as well. The simulations are enumerated and plotted pairwise in comparison, although the pairing is random. The mean velocities of the reinforced runs are larger than the ones of the normal runs. This is especially evident in the overall mean (mean of means marked with '**all**' in Fig. 5.8(a)), which is significantly different. The null hypothesis that the set of means of the reinforced runs and of the normals runs have an indistinguishable mean was rejected with $p < 0.001$ using the t-test. However, since straight and also fast rolling modes are easily predictable and active they are also exhibited without reinforcement for a long time. The traces illustrate that the robot with reinforcement reaches quicker a faster rolling motion and also stays longer in these behaviors. It is important to note that the fast rolling modes are also found again, after the robot is moving slower, see Fig. 5.8(b).

The guidance of the homeokinetic controller using a reward for fast motion has shown to increase the average speed of the robot significantly. Nevertheless there are trials where no increased speed was found.

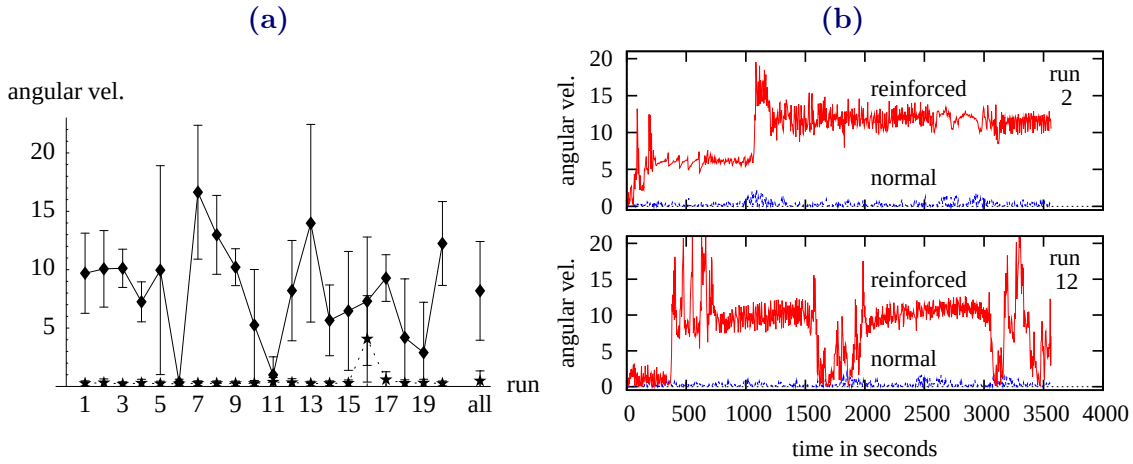


Figure 5.9: Performance of the SPHERICAL robot rewarded for spin. **(a)** Mean and std. deviation of the angular velocity ω_z of the SPHERICAL robot for 20 runs each 60 min long with (**diamonds**/**solid** line) and without (**stars**/**dotted** line) spin reinforcement. The label '**all**' denotes the mean and std. deviation over the means of all runs; **(b)** Time course of the velocity for run number 12 and 19, where **blue**/**dotted** shows the normal case and **red**/**solid** line shows the reinforced case.

5.3.2 Reinforcing Spin

In a different setup we want the robot to follow curves and spin in place. We use the angular velocity ω_z around the z-axis of the world coordinates system, which is perpendicular to the ground plane, as depicted in Fig. 5.7(b). The reward function is now given by

$$r(t) = \frac{1}{3}\|\omega_z\| - 1\,. \tag{5.24}$$

Again the reward is scaled and shifted to be zero for normal runs and to be in an appropriate interval. Positive reward can be obtained by rolling in a curved fashion or by entering a pirouette mode. The latter can be compared to a pirouette done by figure-skaters – with some initial rotation the masses are moved towards the center, so that the robot spins fast in place. The robot also experiences rolling friction, so that fast pirouettes are not persistent. Again, we conducted 20 trials with reinforcement and 20 trials without reinforcement, each for 60 min simulated real time on a flat surface without obstacles. In Fig. 5.9(a) the mean angular velocity ω_z for each simulation is plotted. The angular velocities traces of the robot in two reinforced and two normal runs are displayed in Fig. 5.9(b). In this scenario the difference between the normal runs and the reinforced runs is indistinguishable. Nearly all reinforced runs show a large mean angular velocity. The reason for this drastic difference is that these spinning modes are less predictable and therefore quickly abandoned in the non-reinforced setup. The traces show that the robot in a normal setup rarely performs spinning motion, whereas the reinforced robot, performs after some time of exploration

very fast spinning motions, which are persistent for several minutes. In this setup it can also be seen that the rewarded behaviors are found again after they are lost, see Fig. 5.9(b).

In this scenario the mechanism to modulate the learning speed by a reward signal showed a strong effect on the behavior of the Spherical robot. When controlled by the homeokinetic controller without guidance the robot rarely exhibits narrow curves or spinning behavior. In contrast the guided controller engaged the system into curved motion most of the time. One might wonder how it is possible that this technique is able to reach a behavior that is normally not exhibited. The reason is that when the robot is starting to follow a curve, then the learning rate of the controller goes down, although the world model is still learning normally. In the unguided case the prediction error will rise (because it is a new behavior) and thus the controller will quickly leave this behavior. This actually happens before a fast winding is reached. In the rewarded case the world model is able to capture the behavior before it is left and thus also influences the control network, which in turn enables the control system to enter modes of more narrow curves.

5.4 Discussion

In the first part of this chapter we proposed two methods to guide the otherwise freely self-organized behavior based on teaching signals. We defined an appropriate error function that allows one to directly specify a desired motor pattern and verified its functioning with a simple robot experiment. The balance between self-organized behavior and taught behavior can be adjusted with a single parameter. Using the internal world model we are able to transform sensor teaching signals into motor teaching signals. In this framework, the Spherical robot was taught to prefer the rotation around one particular axis. This was achieved solely by requesting a zero value of the sensor value measuring the axis orientation of that particular axis.

The direct teaching mechanisms form the basis for further investigations that aim at higher level guiding mechanisms, which we consider in the second part of this chapter (Section 5.2). We introduced the method of cross-motor teaching with which abstract relations between motor channels can be specified, such as symmetric or anti-symmetric activity or more elaborate connection patterns. This induces soft constraints and therewith reduces the effective dimensionality of the system. An example with the TwoWheeled robot showed that by enforcing the symmetry between the left and the right wheel the behavior changes qualitatively to straight motion. Nevertheless, the explorative character of the controller is not destroyed. The resulting behaviors are less specific than in the case of direct teaching. For example the TwoWheeled robot can drive forward or backward at its own choice, whereas in the direct teaching setup the direction of driving is specified externally. It is also interesting that the robot remains sensitive to perturbations and changes its behavior from time to time so that the available environment is thoroughly scanned. Especially visible is the effectiveness of guidance with the high-dimensional Armband robot. A cross-motor

teaching configuration with only one connection per joint leads to a fast and coordinated locomotion behavior. A change in the connections rapidly and reliably causes a change in the direction of motion. This method offers a good way for higher level control structures to direct the behavior of the robot.

In the third part of this chapter (Section 5.3) we proposed a simple method to guide the self-organizing behavior using online reward signals. In essence, the original time-loop error is multiplied by a strength factor, obtained from the reward signal. The approach was applied to the Spherical robot with two goals, fast motion and curved rolling. In both cases the performance was significantly increased, although the reinforcement of fast motion showed only a small effect, because the homeokinetic controller achieves fast rolling modes already without guidance. When reinforcing curved rolling the results are distinctive. In this case we find fast spinning modes, which are not observed otherwise. Nevertheless, the exploratory character of the paradigm still remains intact. These results have been published in [90].

Let us compare the guided self-organization using cross-motor teaching and rewards with other approaches to autonomous robot control, namely evolutionary algorithms [104] and reinforcement learning [161]. We will consider several dimensions along which they are compared: *(i)* the achievement of the specified goal, *(ii)* the learning time, *(iii)* the flexibility of the solution, and *(iv)* the universality of the method. Evolutionary algorithms can optimize the parameters of the controlling neural network and can possibly achieve the behaviors demonstrated here. Similar systems in terms of dynamical complexity have been considered in [36, 71, 93]. However, for high-dimensional systems, identical subcomponents are often used and in any case it requires a long learning time (many generations with many individuals). The final controller is mostly static, such that it only works in the conditions it was evolved in. Evolutionary algorithms are universal, but the representation of the problem in the artificial genome requires great care. When considering reinforcement learning (RL) we have to distinguish how the control is organized. Traditionally RL uses a set of discrete actions, which would be an intractably large set in the case of the 13-DoF Armband robot. RL would not yield a good performance in this case. In order to achieve a good performance the space of actions has to be structured first. A very powerful RL method is the natural actor-critics [121]. This allows for a continuous state and action space and is currently the most efficient RL algorithm for high-dimensional systems [1]. Nevertheless, for example a seven DoF robotic arm requires initial supervised learning (by human demonstration) before it can successfully learn to perform the task [119]. Even with sophisticated methods like natural actor-critics the learning times are very long in comparison to the approaches proposed here. RL can deal with delayed rewards since it solves the credit assignment problem, whereas the reward schema used here (Section 5.3) require online rewards. In the case of the cross-motor teaching the specification of the desired behavior is indirect and limited in comparison to the possibilities of reward function in RL and the fitness function in evolutionary algorithms. The methods proposed here for guided self-organization are specific methods to shape the behaviors of the homeokinetic controller. As a matter of fact, the desired behaviors are found very fast even in high-

dimensional and dynamically complex systems. However, the desired behaviors are only partially followed. This is due to the underlying self-organization which causes an ongoing exploration around the desired behaviors. Concerning the flexibility, the resulting controllers are highly adaptive and can change quickly to new situations, which is one of the major strengths of the self-organizing system. To conclude, the guided self-organization methods offer a fast development of goal-oriented behaviors in high-dimensional continuous robotic systems from scratch, which cannot be achieved with other learning control systems so far. However, the implementation of goals is comparably limited. The reward-based guidance allows any reward signals, but no time delays are tolerated. The cross-motor teaching method is suitable to select a subset of behaviors, but cannot be generalized to all behaviors. A combination of both methods is also possible, namely using cross-motor teaching to be very effective in high-dimensional systems and additionally using rewards to give a fine grain control over the behavior.

In the current setup the controller and the internal world model will forget past behaviors, so that there is no long term effect of the reinforcement. Another problem for practical applications is that we achieve only a tendency to a certain behavior and cannot guarantee its successful execution. These problems will be tackled in the next chapter, where we will look into the acquisition of behavioral primitives using a set of competing expert networks. The presented guiding mechanisms are then usable to shape the development of behavioral primitives.

Chapter 6

Goal-Oriented Behavior from Self-Organized Behavioral Primitives

An expert is a person
who has made all the mistakes
that can be made in a very narrow field.

Niels Bohr (1885 - 1962)

The homeokinetic control is a very powerful method to obtain coordinated behavior in complex robot systems (Chapter 4), but it can not really be used in practical applications, because one cannot directly accomplish a task with it. In the present chapter we will solve this shortcoming in a surprisingly intuitive way.

A commonly used technique for learning a particular task is reinforcement learning[1] [161]. A reinforcement learning algorithm learns to choose the right sequence of actions to maximize a long-term reward. The latter has to be designed according to the current task. Unfortunately, in real world applications the dimensionality of the systems is so large that it is impossible to learn the right sequence of ad-hoc actions within a feasible amount of time. Consider for example a robot with about 20 degrees-of-freedom (DoF), e. g. the SNAKE robot as used in the present work (Section 2.2.7) or the Honda humanoid robot ASIMO [173] (with 26 DoF), where each DoF needs a motor command at every instant in time. Even if only three different commands per DoF are used then there are $3^{20} > 10^9$ different actions to choose from. As it is impossible to search such a huge space for useful actions for a particular situation, it is necessary to find a more compact representation by coarse grain building blocks, namely behavioral primitives.

[1] Reinforcement learning refers here to the maximization of a future reward, which is possibly given temporally delayed. This is in contrast to the immediate reinforcement signals that were used in Section 5.3.

As the concept of *behavioral primitives* is studied in fields of human movement science and also in robotics, we find in the literature a variety of synonyms, such as 'movement primitives' [141], 'motor schemas' [4], and 'units of action' [154]. It is believed that the existence of behavioral primitives is essential for complex and effective skill learning and movement generation [141]. Behavioral primitives are defined as sequences of actions or as a sensorimotor dynamics that accomplish a complete goal-oriented behavior. They can be as simple as an elementary action, e. g. 'extending a leg', 'waving a hand', etc., but can also comprise more complex movements. For the same reasons as given above, in [140] it was stated that too simple low-level representations do not scale well to systems with many degrees of freedom. Thus, it was suggested that behavioral primitives code complete temporal behaviors, like 'grasping a cup', 'walking' or 'shooting a ball'. Since they are to be used as building blocks for more complex behaviors it is important that the primitives can be easily sequenced and combined. An appropriate storage of the primitives can be done in closed-loop controllers, much like the homeokinetic controller itself. This yields a good generalization performance and smooths the transients between primitives, see also [162].

Let us now consider the behaviors that are performed by the homeokinetic controller with some temporal persistence. Even though they do not have a specific goal [2], unless shaped by guided self-organization, see Chapter 5, we argue that an observer can attribute a goal to most of the particular behaviors, for example the different rolling modes of the Spherical robot appear to follow the goal of directed motion. Therefore these behaviors are likely to be useful for the construction of more complex goal-oriented behaviors. The sensorimotor loops established by the homeokinetic controller do not produce merely elementary actions but 'complete temporal behaviors'. We will see the large difference in the performance of learning a task when behavioral primitives are used instead of elementary actions even in low-dimensional systems.

In the first part of this chapter we will focus on the acquisition of behavioral primitives, and in the second part we will use them in conjunction with reinforcement learning to create composite goal-oriented behaviors.

6.1 Acquisition of Behavioral Primitives

If the robot re-encounters a situation in the environment it should be able to reuse behaviors that have been acquired earlier. The extraction of behaviors from a continuous stream of sensorimotor information involves clustering or segmentation, generalization, and suitable storage. The segmentation is necessary to chop the data into chunks that belong to one behavior. In the case of the self-organized motor control the behavior changes permanently, such that the system is always on a transient from one behavior to another. However, the lifetimes of these transients differ depending on the TLE, see Section 3.6. Highly

[2] Since the homeokinetic controller optimized the time-loop error Eq. (4.12) there is always the general goal of self-exploration and active interaction but no specific goal per se.

predictable and sensitive behaviors will remain longer than others, which makes them easier to be extracted. Nevertheless, there is no typical timescale of a behavior. Beside the appropriate segmentation, an important requirement for the behavioral elements is the stability against perturbations and a wide generalization capability. This is advantageous in order to reach and stabilize the belonging behaviors from different starting points in the behavior-space. We will see that both properties are fulfilled by the behavioral primitives used here.

We approach the challenge of behavior acquisition with multiple local models/controllers, which follows the paradigm of 'divide and conquer', and is a common way to tackle complex non-linear systems [99]. The mixture of local experts [72] is a method performing the segmentation and modeling online. There, a gating network and a set of expert networks are trained. The gating network learns to map the inputs to a weighted sum of the outputs of a set of experts, whereas each expert learns the input to output mapping, depending on the strength of its participation in the total output. Let us assume the data originates from a set of different dynamics (e. g. different behaviors in the case of the robot) and the system switches between them. In this scenario a problem arises when the gating is based on the inputs alone. Namely that the underlying dynamics will have overlapping input domains and thus the gating cannot easily distinguish between the dynamics. Neither does the gating network encourage the expert to compete, but rather to cooperate. The latter makes the representation more prone to the loss of information through interference with previously learned relationships. A method that overcomes these problems is the competing experts schema [117], which we will introduce in more detail below. The competition between the experts is based on a single quantity, namely on their ability to predict the current data. This approach was successfully applied e. g. to the vowels recognition in speech data [106] and the analysis of wake/sleep EEG signals [75]. In our case the data is provided by the behaving robot, with each expert representing one behavior. The segmentation between behaviors happens naturally at borders of the Voronoi cells [7] defined by the areas where one expert is predicting best. Thus, in contrast to the setup with a gating network, the segmentation is now not directly dependent on the inputs (sensor values). Moreover, a good generalization is brought about by the neural networks which store the behaviors in their synaptic weights.

The next section explains the competing expert schema and the softmax competition rule. After that, a new winner-takes-all competition rule is proposed that is more suitable for the data obtained from the behaving robot. Using two examples we demonstrate the partitioning of behavior-space by the experts. Finally, we investigate how the experts can be used to control the robot and to find flexible and combinable behavioral primitives.

6.1.1 Competing Experts

Let us now focus on the competing expert setup with softmax competition and annealing as introduced by Pawelzik et al. [117]. Consider the time series $\{a_t, b_t\}$ with $t = 1, 2, \ldots$

of input-output pairs. The underlying dynamics of the time series is assumed to switch on a rather slow timescale, which is also a valid assumption for the robot behaviors. We have an ensemble of experts

$$F_i : a_t \mapsto \tilde{b}_t^i \qquad i = 1, \ldots, r\,, \tag{6.1}$$

mapping the input a_t to the predicted output $\tilde{b}_t^i$ and we define for each expert the square prediction error as

$$\Xi_t^i = \left| b_t - F_i(a_t) \right|^2 . \tag{6.2}$$

Let the experts F_i consist of neural networks with the parameter matrix W^i that is adapted to minimize the prediction error via gradient descent as

$$\Delta W^i = -\epsilon_F p_t^i \frac{\partial \Xi_t^i}{\partial W^i}\,. \tag{6.3}$$

The learning rates of the experts are composed of a maximal learning rate ϵ_F and a weighting coefficient p_t^i determined by the competition function called softmax:

$$p_t^i = \frac{e^{-\beta \sum_{\tau=-\delta}^{\delta} \Xi_{t-\tau}^i}}{\sum_{j=0}^{r} e^{-\beta \sum_{\tau=-\delta}^{\delta} \Xi_{t-\tau}^j}}\,, \tag{6.4}$$

where β is the strength of the competition and δ is a time-horizon. For $\beta = 0$ all experts learn equally and for $\beta = \infty$ there is a hard competition in the winner-takes-all fashion. The weighting coefficients are intuitively the relative probability of contribution of each expert to the current dynamics. Note that the weighting coefficients are normalized i. e. $\sum_{j=1}^{n} p_t^j = 1$ and depend on δ future prediction errors. These are, however, not known at the time of the competition and thus one can use instead of $\sum_{\tau=-\delta}^{\delta} \Xi_{t-\tau}^i$ a sliding average $\overline{\Xi}_t$ which is iteratively defined as $\overline{\Xi}_t^i = 1/\delta \Xi_t^i + (1 - 1/\delta)\overline{\Xi}_{t-1}^i$. In order to ensure a good distribution of experts, the competition strength β is adiabatically increased during the learning, which is called annealing process. It must take place at a sufficiently long timescale, because the diversification occurs at particular "temperatures" ($T = \frac{1}{\beta}$) where the network parameters separate abruptly [137].

6.1.2 Framework

Ultimately the learned experts networks should serve as controllers for a particular behavior. In order to guarantee a smooth transition between these behaviors, as well as a robust and coherent activation of the appropriate behaviors, it is necessary to provide the robot with a flexible representation of the behavioral primitives. We suggest using an adaptive input-output mapping realized by a feed-forward neural network.

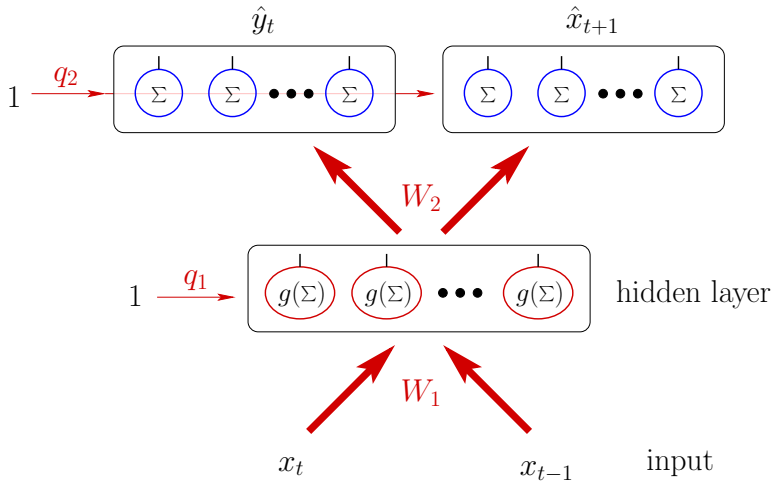


Figure 6.1: Topology of an expert neural network. Inputs are the vectors of sensor values x_t and x_{t-1}. Red arrows denote all-to-all connections. The hidden layer consists of neurons with hyperbolic tangent activation functions. The output layers have linear neurons. See Eq. (6.6).

The design of the experts has to be done with some care, because they should have the right complexity. On the one hand, too simple experts are not able to capture a single behavior, such as a particular rolling mode of the Spherical robot. In this case several experts would be required for one behavior, which is undesirable because eventually the experts are to be controllers for behavioral primitives. On the other hand, too complex experts learn slower and most importantly will possibly capture multiple behaviors. Why is this undesirable? Because if the experts are used as controllers for the robot, then it is more difficult to predict which behavior will be exhibited. This raises the question of the basin of attraction of the stored behaviors that will be discussed in Section 6.1.5 below.

The input-output configuration of the experts is similar to the one of the homeokinetic controller, such that they can be used almost directly as a drop-in replacement for the robot controller. However, the only difference is that they also receive delayed sensor values, which enables them to capture behaviors that are based on the internal parameters dynamics of the homeokinetic controller. Hence, we define

$$\begin{pmatrix} \tilde{x}_{t+1} \\ \tilde{y}_t \end{pmatrix} = F_i(x_t, x_{t-1}), \tag{6.5}$$

for $i = 1, \ldots, r$ experts. Note that $\binom{x}{y}$ denotes an $n+m$ dimensional column vector with all entries from x above entries of y. The r expert networks have to learn the mapping from sensor values (present and past) to motor values and future sensor values. In motor control theory, these models are called forward and inverse models [177], which are jointly comprised in the here-proposed experts. The architecture of an expert neural network is

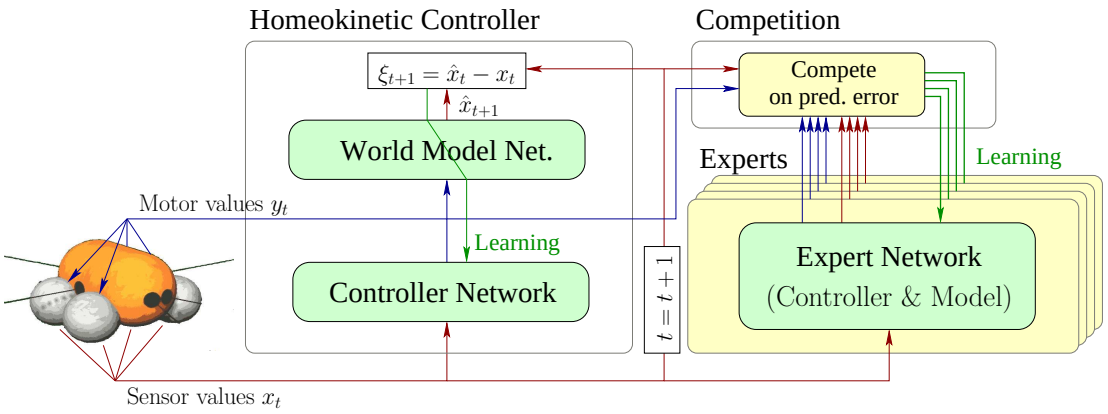


Figure 6.2: Overall architecture of homeokinetic controller with the competing experts long-term memory.

depicted in Fig. 6.1 and is given by

$$F_i(x_t, x_{t-1}) = W_2^i \, g\left[W_1^i \begin{pmatrix} x_t \\ x_{t-1} \end{pmatrix} + q_1^i\right] + q_2^i, \tag{6.6}$$

where W_1^i, W_2^i are synaptic weight matrices and q_1^i, q_2^i are offset vectors of expert i. The offsets are required to produce a non-zero output for zero inputs or to account for sensor or motor values which have non-zero middle positions. The neurons in the hidden layer have a non-linear activation function $g(\cdot) = \tanh(\cdot)$ (component-wise). Their number (k) determines the dimension of the weight matrices and offset vector, i. e. $W_1 \in \mathbb{R}^{k \times 2n}$, $W_2 \in \mathbb{R}^{(n+m) \times k}$, $q_1 \in \mathbb{R}^k$, and $q_2 \in \mathbb{R}^{n+m}$. The interesting point is that we designed the hidden layer as a bottleneck, i. e. $k < n + m$, such that an intermediate low-dimensional representation has to be found, thus improving the generalization properties of the experts. The entire setup with the homeokinetic controller is illustrated in Fig. 6.2. In the current setup there is no information flow back to the homeokinetic controller, which presents an interesting possibility for further improvements.

Analogously to Eq. (6.2) we define the prediction error ξ_t^i of expert i as the mismatch between observed and predicted values, i. e.

$$\xi_t^i = \begin{pmatrix} x_t \\ y_{t-1} \end{pmatrix} - F_i(x_{t-1}, x_{t-2}), \tag{6.7}$$

$$\Xi_t^i = {\xi_t^i}^\top \xi_t^i. \tag{6.8}$$

The parameters of F_i are adapted to minimize Ξ_t^i by gradient descent. Since the network has multiple layers the backpropagation learning rule [170] is used, which is a generalization of the delta rule. For that the output of the hidden layer is required, which we denote by

$$o^i = g\left[W_1^i \begin{pmatrix} x_{t-1} \\ x_{t-2} \end{pmatrix} + q_1^i\right]. \tag{6.9}$$

Using Eq. (6.9) the updates of the parameters W_2^i and q_2^i are simply given by delta rule as

$$\Delta W_2^i = \epsilon_t^{F_i} \xi_t^i {o^i}^\top, \tag{6.10}$$

$$\Delta q_2^i = \epsilon_t^{F_i} \xi_t^i. \tag{6.11}$$

For the updates of W_1^i and q_1^i the mismatch ξ_t^i at the output layer is back-propagated to a mismatch at the hidden layer $\rho_t^i \in \mathbb{R}^k$ as

$$\rho_t^i = {W_2^i}^\top \xi_t^i. \tag{6.12}$$

Then we find

$$\Delta W_1^i = \epsilon_t^{F_i} (\rho_t^i \circ g') \begin{pmatrix} x_{t-1} \\ x_{t-2} \end{pmatrix}^\top, \tag{6.13}$$

$$\Delta q_1^i = \epsilon_t^{F_i} \rho_t^i, \tag{6.14}$$

where g' is taken at $W_1^i \binom{x_{t-1}}{x_{t-2}} + q_1^i$, see Eq. (6.9), and $\circ$ denotes the component-wise multiplication, see Section 4.1.3. The variable learning rate $\epsilon_t^{F_i}$ is determined by the competition algorithm. In the case of the annealed softmax competition we have $\epsilon_t^{F_i} = \epsilon_F p_t^i$, where ϵ_F is a uniform learning rate for all experts and p_t^i is the weighting coefficients, cf. Eq. (6.4).

6.1.3 Winner-Takes-All with Suboptimality Penalty and Annealing

Although the competing experts algorithm is particularly suitable for non-stationary time series with an underlying switching dynamics we encounter a fundamental problem when applying it to the behavior extraction task. The assumptions that are made for the annealing process, is that the switching between different dynamics occurs on a much faster timescale than the annealing process. For the behaving robot controlled with the homeokinetic controller, however, the switching between behaviors has a statistics that would imply an extremely long annealing process. In our robotic systems we find often long periods of one behavior and a very inhomogeneous switching dynamics where certain behaviors are very rarely exhibited. Remember that the parameter dynamics of the homeokinetic controller has no fixed point, but is rather on a transient all the time. The length of time that a particular behavior is exhibited depends strongly on the size of the TLE, see Section 3.6, and on the physical stability of the behavior.

Ideally a behavior is captured by an expert when it is exhibited for a sufficiently long time in total, independent of the amount of intermediate switches and the total run-time. We also want a high degree of preservation in the once learned behavioral representations. The latter implies a winner-takes-all learning scheme, because otherwise a strong interference with already learned behaviors occurs. The most naive solution would be to always select

the best expert to learn the current pattern. However, in this case we observe that typically one and the same expert always wins the competition and the remaining experts are not allowed to learn anything. We propose a solution for this problem that we call *suboptimality penalty with annealing*, that penalizes experts performing suboptimally with respect to their past performance while individually annealing the learning rate. We introduce a minimal achieved error for each expert and define the positive deviation from this minimum as a measure of suboptimality. The reasoning is that when an expert leaves its domain of competence then its performance drops, whereas uncommitted experts have a similar prediction error on all behaviors and are thus less penalized.

The prediction error, Eq. (6.8), of the experts is noisy. Thus, we redefine it as

$$\Xi_t^i = \frac{1}{\tau_E} {\xi_t^i}^\top \xi_t^i + \left(1 - \frac{1}{\tau_E}\right) \Xi_{t-1}^i \,, \tag{6.15}$$

which is simply the sliding average of the square misfit ${\xi_t^i}^\top \xi_t^i$ with the time constant τ_E. To determine the best performance we define a second, longer averaged prediction error

$$\overline{\Xi}_t^i = \frac{1}{\tau_{E'}} \Xi_t^i + \left(1 - \frac{1}{\tau_{E'}}\right) \overline{\Xi}_{t-1}^i \,, \tag{6.16}$$

where the new time constant $\tau_{E'}$ is usually chosen to be $\tau_{E'} = 10\tau_E$. The best obtained performance of an expert is measured by the minimal achieved (smoothed) prediction error and is therefore given by

$$\breve{\Xi}_t^i = \min_{i \le t} \overline{\Xi}_t^i \,. \tag{6.17}$$

Using this, we define the suboptimality as the positive derivation from this minimum and obtain the penalized prediction error as

$$\tilde{\Xi}_t^i = \Xi_t^i + p \cdot \max(0, \Xi_t^i - \breve{\Xi}_t^i)^2 \,, \tag{6.18}$$

where p is the penalty factor. The winning expert w_t is the one with the lowest penalized prediction error, thus

$$w_t = \arg\min_i \tilde{\Xi}_t^i \,. \tag{6.19}$$

In essence, the competition is based on the penalized prediction error, where the penalty is given by the deviation from the minimal achieved error. Only the winning expert is trained using the patterns observed from the robot. For that Eqs. (6.10–6.14) are applied for $i = w_t$. An illustration of the method is provided in Fig. 6.3. For two behaviors and two experts the typical development of the prediction errors and the derived quantities are shown. At the beginning, expert number two wins the competition and improves its performance for this behavior. After the behavior changes, the prediction error of expert two is still better than the prediction error of the yet uncommitted expert (number one). However, the

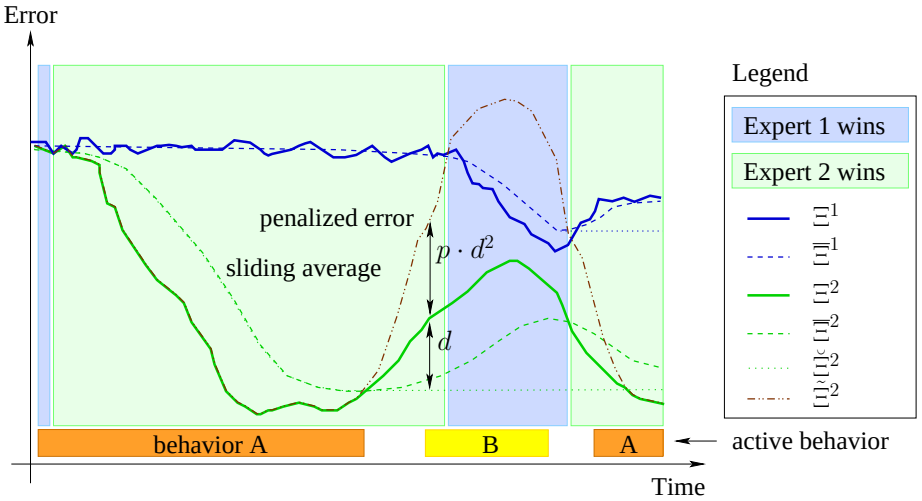


Figure 6.3: Illustration of the competition of two experts with the penalty for suboptimality algorithm. The **orange** and **yellow** bars indicate the currently active behavior. In between a transient occurs. Expert 2 wins by chance at the beginning and learns to predict behavior A, as seen at the drop of the prediction error (**green** lines). When the behavior changes the prediction error of expert 2 increases and the penalized error (**brown dashed dotted** line) becomes worse than the prediction error of expert 1 (**blue** line), thus expert 1 wins, and so forth. Note that due to the sliding average of the prediction error (Eq. (6.15)) there is a time shift between the change of behavior and the change of the winning expert.

suboptimality penalty shows its effect and so expert one wins and can therefore learn the second behavior. Note that the switch to a new winning expert occurs temporally after the change of behavior, which is due to the sliding averages of the errors. For that reason it is advisable to perform the training with delayed training patterns, where the delay is half of the averaging time constant τ_E. More formally, the update (Eqs. (6.10–6.14)) of the parameters of the current winning expert ($i = w_t$) are calculated using input-output pairs from $t' = t - \lfloor \tau_E/2 \rfloor$.

To ensure convergence the learning rate of the winning expert is annealed. Thus, the learning rate $\epsilon_t^{F_i}$ of expert i is given by

$$\epsilon_t^{F_i} = \begin{cases} \left(1 - \frac{1}{\tau_\epsilon}\right) \epsilon_{t-1}^{F_i} & i = w \\ \epsilon_{t-1}^{F_i} & \text{otherwise.} \end{cases} \tag{6.20}$$

The described algorithm can be successfully used to teach experts the robot behaviors produced by the self-organizing controller. However, there are some useful extensions which help to ensure convergence and enhanced the long-term plasticity of the system, which are described now. The annealing of the learning rate of the winner, Eq. (6.20), follows an exponential decay with time constant τ_ϵ. Such an exponential decay is too fast to ensure convergence in general. However, we introduce a rate of "forgetting" (τ_F), at which the learning rates of all experts are "warmed up" again. Instead of an exponential increase we use a linear increase, thus, the learning rate $\epsilon_t^{F_i}$ of expert i is given by

$$\epsilon_t^{F_i} = \begin{cases} \left(1 - \frac{1}{\tau_\epsilon}\right) \epsilon_{t-1}^{F_i} & i = w \\ \epsilon_{t-1}^{F_i} + \frac{1}{\tau_F} & \text{otherwise.} \end{cases} \tag{6.21}$$

Eq. (6.21) comprises an exponential decay for the learning rate of the winning expert and at the same time a linear increase for non-winning experts using a much longer time constant $\tau_F \gg \tau_\epsilon$. Assuming expert i wins with probability $p(i)$ we can estimate the asymptotic learning rate as $\epsilon_t^{F_i} = \frac{\tau_\epsilon}{\tau_F p(i)}$. This maintains a certain flexibility in the system of experts and allows for a possible reuse of already specialized experts in case they are unused for a long time.

In order to improve the speed of learning and ensure that uncommitted experts are able to quickly capture the observed behavior, it is advisable to have an initial phase where all experts receive training patterns. Thus, Eqs. (6.10–6.14) are applied for $i = 1, \ldots, r$ with $\epsilon_t^{F_i} = \epsilon_F \cdot e^{-t/\tau_I - 1}$ independently of the competition, where τ_I is a time constant defining the length of the initial phase.

The second quantity that should be affected by the "forgetting" process is the minimal error $\breve{\Xi}^i$. A slow increase of the minimal errors for non-winning experts ensures that an expert that is not selected as a winner for a very long time gets a higher chance to be trained on a different behavior. Additionally, we only decrease the minimum error for the winning expert, because there might be periods of trivial behavior that are predicted well

by all experts, e. g. if all sensors and motor values are zero. The minimal prediction error is thus given iteratively by

$$\breve{\Xi}_t^i = \begin{cases} \min(\breve{\Xi}_{t-1}^i, \Xi_t) & i = w\,, \\ \breve{\Xi}_{t-1}^i \cdot \left(1 + \frac{1}{\tau_F}\right) & \text{otherwise}\,, \end{cases} \tag{6.22}$$

where τ_F is again the "forgetting" time constant, see Eq. (6.21).

To summarize, we use a winner-takes-all competition with annealing learning rates and a penalty term for suboptimality, which is defined using the minimal prediction error as a measure for optimal performance. What are the conditions that need to be fulfilled by the time series such that this algorithm works? Without going into much detail, the first assumption is that the underlying source of the time series is multifold and exhibits a certain switching between its different sub-dynamics (behaviors in our case). The switching is assumed to occur on a slower timescale than the system cycle. While the switching from one sub-dynamics to the next need not be abrupt, the transient period is assumed to be shorter than the duration of the switching intervals. These assumptions are also made for the original competing expert approach [117]. However, in contrast to the original algorithm our approach does not require an equal sampling of all sub-dynamics and a frequent switching between them.

Parameter selection

Let us now discuss the parameter selection. The learning rate ϵ_F of the expert networks must be chosen to fit the network size and the typical length of a period of oscillatory behaviors. It should not be too large, to avoid local overfitting. In our applications we use $\epsilon_F \in [0.001, 0.01]$.

The fine tuning of the penalty factor p was found to be unnecessary. In our applications we use a value between 1 and 10. The more experts competing, the higher the values of p are recommended.

The remaining parameters are timescales. The shortest one is the error smoothing horizon τ_E. This should be rather small and should be adjusted to filter the noise in the prediction error. It should be certainly not larger than a 10th of the expected behavior switching time. The constant for annealing of the learning rate τ_ϵ and the length of initial learning phase τ_I are rather uncritical and we may chose $\tau_\epsilon = \tau_I = 10/\epsilon_F$. The time to forget the best performance τ_F must be larger than the maximal expected time to re-encounter all behaviors. In the following robot experiments it is chosen to be $\tau_F = 3 \cdot 10^5$ which is equivalent to about one hour of simulated real time. Since most of the experiments we report do not exceed this time, the forgetting is more of a conceptual term, ensuring long-term adaptivity. We expect that it will become more important for long experiments, especially with changing environments.

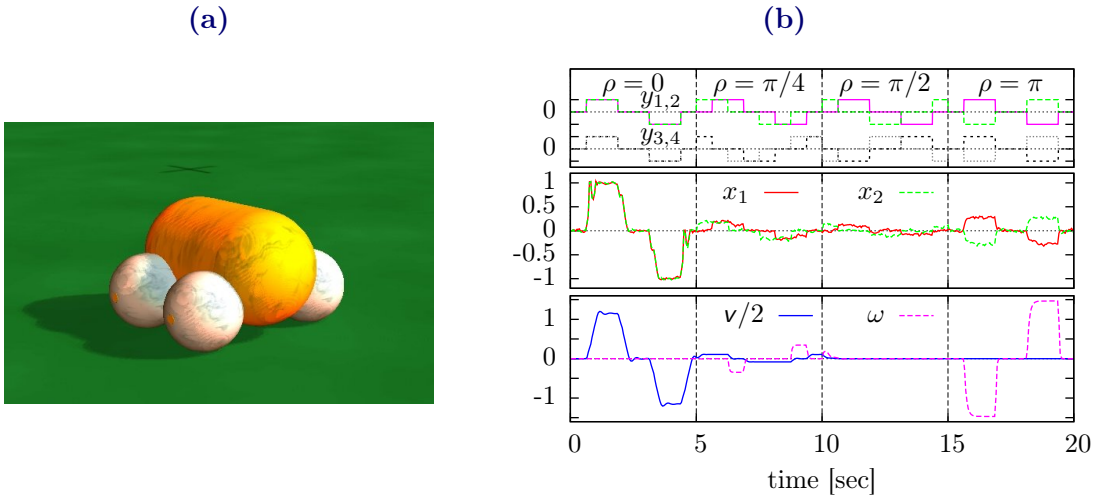


Figure 6.4: Simulated FourWheeled robot. The robot has four independently powered wheels and four wheel velocity sensors. **(a)** Screenshot of the simulation; **(b)** Response of the robot to different activations of the motors. The motor values $y_{1,\dots,4}$, Eq. (6.23), follow an impulse function with different phase shifts ρ, **top** panel. The sensor values $x_{1,2}$ of the two front wheels show only a response in the case $\rho = 0$ and $\rho = \pi$, **middle** panel. The driving velocity v and the turning velocity ω are shown in the **bottom** panel.

6.1.4 Extraction in Action

Let us now apply the competing experts architecture to the behavior extraction task at the example of two robots. We start with a wheeled robot and continue with the Spherical robot.

Application to the FourWheeled Robot

The first robot we consider is the FourWheeled robot, as depicted in Fig. 6.4(a) and described in Section 2.2.2. The robot has four independently driven wheels. In order to move the robot reasonably a certain degree of coordination is required. To illustrate the physical properties of the robot we use predefined motor values that are described by a periodic impulse function as

$$y_i = \begin{cases} 1 & \sin(0.2\,t + i\rho) > 0.5, \\ -1 & \sin(0.2\,t + i\rho) < -0.5, \\ 0 & \text{otherwise}, \end{cases} \tag{6.23}$$

with the phase shift ρ. The responses of the robot for different values of ρ are plotted in Fig. 6.4(b). Essentially the robot only moves if the wheels on one side rotate in the same direction, i. e. for $\rho = d\pi$ with $d = 0, 1, \dots$.

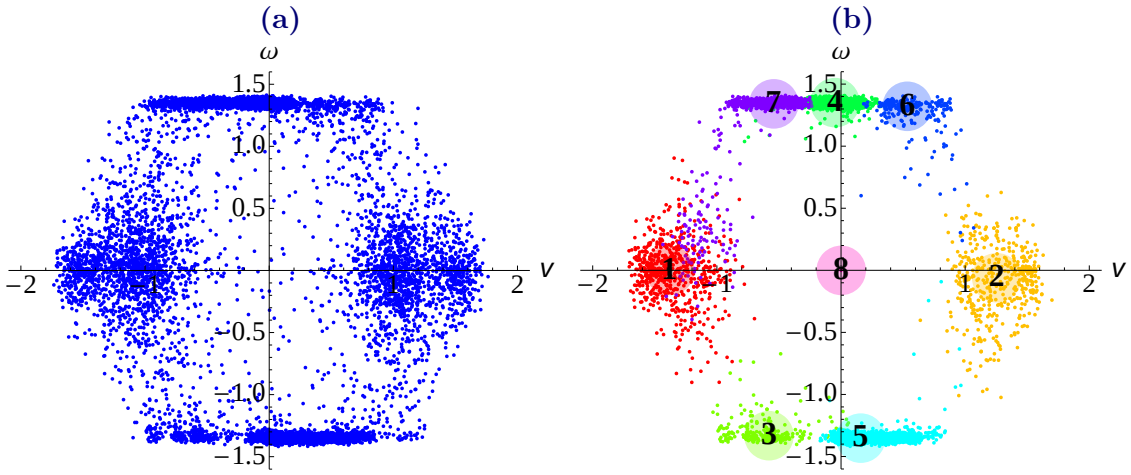


Figure 6.5: Distribution of 8 experts on the behavior space during a 30 minute experiment with the FourWheeled robot. Each point represents the state of the robot in terms of driving velocity v and rotation velocity ω at a certain time. **(a)** Data points as produced by the robot; **(b)** Clustering with experts. The color represents the number of the winning expert (w). Only points with a low prediction error of the winning expert are plotted. The center of the point-cloud for each expert is marked with a numbered disk. Parameters of homeokinetic controller: $\epsilon_C = \epsilon_A = 0.05$, 50 Hz update rate; for competition: $\epsilon_F = 0.01$, $p = 10$, $\tau_E = 20$.

When controlled with the homeokinetic controller a coordinated behavior develops very quickly, such that the robot shows straight driving and curved driving with different radii. For the behavior extraction we use the following parameters: There are $r = 8$ expert networks with $k = 2$ hidden units (see Fig. 6.1). The learning rate for the experts is $\epsilon_F = 0.01$ and the penalty factor $p = 10$. The timescale for the averaging of the prediction error is $\tau_E = 20$. For the remaining parameters the default values are used, see Section 6.1.3. We let the robot drive for 30 min and recorded the translational and rotational velocity as well as the index of the winning expert. The behavior of the robot in terms of translational and rotational velocities is displayed in Fig. 6.5(a). The resulting clustering is presented in Fig. 6.5(b), where a clear partitioning of the behavior space is observed. There are two experts for forward and backward driving (#1,#2) and two for rotation in place in both directions (#4,#5). Experts #3, #6, and #7 represent curved driving with different radii and expert #8 remains unused. The histogram of winning frequencies of the experts is plotted in Fig. 6.6. The experts do not win with the same probability. This shows that the extraction is not based on the duration and frequency of a behavior but rather its qualitative properties. The trajectory of the robot is displayed in Fig. 6.7, where each part is colored according to the winning expert. A clear and stable segmentation of the different behaviors is seen from the start of the experiment on.

If there are many experts available then not all of them receive sufficient training data to learn a behavior. We define a threshold at 1% of the learning time and those experts

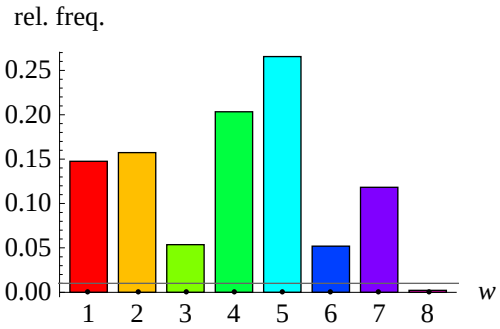


Figure 6.6: Histogram of winning. For each expert the relative frequency to win the competition is plotted. The color-code is the same as in Fig. 6.5(b). The gray line at 0.01 marks the threshold under which the experts are considered to be uncommitted, here #8.

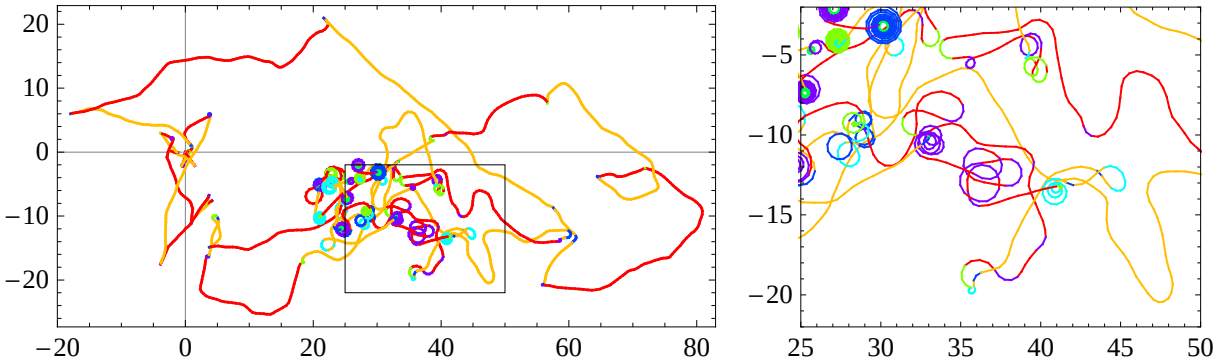


Figure 6.7: Trajectory of the robot in physical space, colored according to the active expert. The **right** graph shows a magnified view of the rectangular area marked in the **left** graph. The color code is identical to Fig. 6.5.

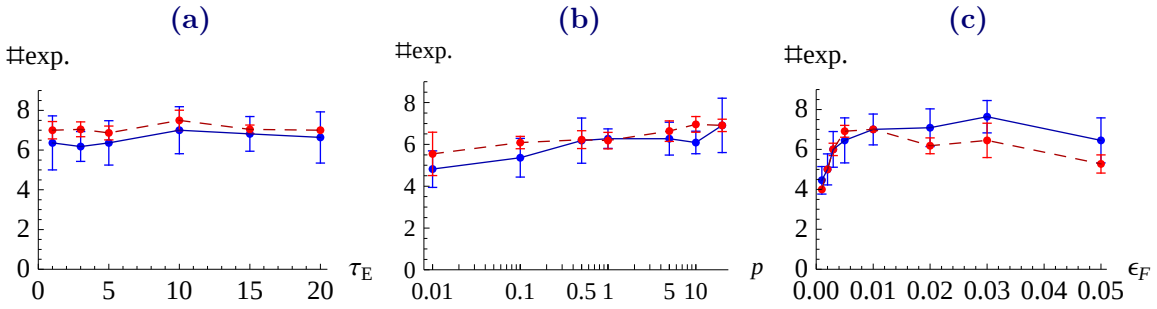


Figure 6.8: Parameter dependence of the number of committed experts. The **blue** lines stand for the setup with independent robot behaviors and **red dotted** lines stand for the case of a single recorded robot. For each parameter setting 10 independent simulations have been performed, where the initial weights of the expert networks are randomly chosen. The graphs show the dependence of the number of committed experts on: **(a)** the averaging timescale τ_E; **(b)** the penalty p (in log-linear scale); and **(c)** the learning rate ϵ_F. Parameters (if not varied): $\tau_E = 5$, $p = 10$, $\epsilon_F = 0.005$.

that win less than 1% are considered to be uncommitted, see also Fig. 6.6. Let us study how the number of committed experts is influenced by the system parameters. Essentially, there are three parameters to check: the learning rate of the experts ϵ_F, the averaging constant τ_E and the penalty p. We use $r = 12$ experts in order to always have room for uncommitted ones. We use two setups, one with a differently behaving robot[3] for each trial and one with a single recorded robot behavior. In the first case the particular sensorimotor data differs in the sequence and duration of behaviors. However, the number of actually shown behaviors is roughly constant. In case of the recorded robot behavior the only source of non-determinism is the random initialization of the synaptic weights (W^i, q^i) of the experts. We found that the extraction algorithm has little dependence on the choice of τ_E and p, see Fig. 6.8(a),(b). Unsurprisingly, for a higher penalty we obtain a higher number of experts. The learning rate ϵ_F, however, has a significant impact on the number of committed experts, see Fig. 6.8(c). For very low learning rates few experts are committed, which is due to a slow learning progress. Thus, the minimal errors decrease slowly and the penalty for suboptimality is less effective. Nevertheless, a wide range of values of the learning rate results in a high number of committed experts, such that ϵ_F does not require fine tuning. In general, we find no over-specialization, i. e. that behaviors are not split into many small sub-behaviors, independent of the parameter choice, such that uncommitted and highly adaptive experts remain available for new behaviors.

Let us now have a closer look at the clustering with different numbers of committed experts. For better comparison we use the same recorded robot behavior as above (Fig. 6.8). In Fig. 6.9 the winning statistics and the clustering of the behavior space for three different values of the learning rate are displayed. When only 4 experts are committed we find a

[3]The robot was controlled with an independently initialized homeokinetic controller with a different instance of the noise process, thus the behavior is not identical.

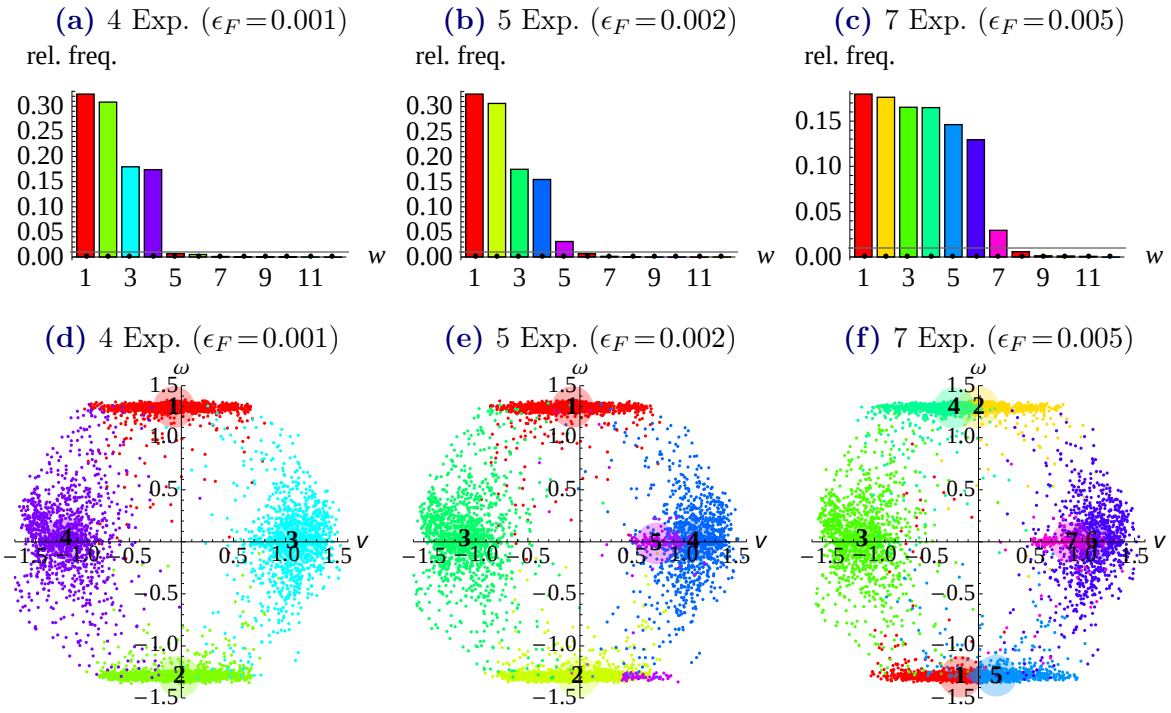


Figure 6.9: Different number of committed experts depending on learning rate. The 12 experts are sorted according to their winning frequency. The winning statistics **(a-c)** and the clustering of the behavioral space **(d-f)** for simulations with $\epsilon_F = 0.001, 0.002, 0.005$ are displayed. For $\epsilon_F = 0.001$ **(a,d)** only 4 experts are committed, whereas for $\epsilon_F = 0.002$ we find 5 and for $\epsilon_F = 0.005$ there are 7 experts committed. See also Fig. 6.5 and Fig. 6.8. The underlying robot behavior was identical (a recorded run was played back). Parameters: $\tau_E = 5$, $p = 10$.

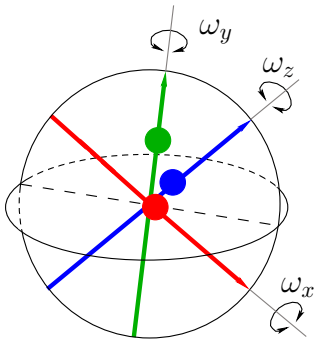


Figure 6.10: Spherical robot with angular velocities around internal axes.

symmetric arrangement of experts covering the most prominent behaviors, Fig. 6.9(a),(d). When more experts are committed, a more fine grain clustering occurs. For example, with 5 experts there are two experts for forward driving, one for slower (#5) and one for faster (#4) motion, Fig. 6.9(b),(e). With 7 experts we get different representations for curved behaviors, e. g. left curves with slow forward or backwards speed covered by experts #1 and #5 in Fig. 6.9(c),(f).

Before we turn to the Spherical robot, let us summarize. The clustering of the behavioral space of the FourWheeled robot was successfully performed by the proposed competing expert algorithm. The parameter dependence turned out to be graceful and only the learning rate had to be chosen appropriately. It should be emphasized that no over-specialization was observed, such that uncommitted experts are available for future novel behaviors.

Application to the Spherical Robot

A more interesting robot in terms of behavior is the Spherical robot, which was the subject of several experiments, e. g. in Sections 4.8.4 and 5.3.1. We will now apply the competing experts setup to this robot and extract primitive behaviors. In the following experiments we use the three axis-orientation sensors described in 2.2.5. Thus the robot has three motors and three sensors, which results in six inputs and six outputs for the expert networks, Eq. (6.5). As the number of hidden units we have chosen $k = 4$, instead of 2 in the case of the FourWheeled robot, because the behaviors have a more complex dynamical structure. For example, when the robot rolls on a flat surface, the sensor values perform a harmonic oscillation with different amplitudes. The frequency of this oscillation depends on the velocity of the robot. If the rotation axis matches one of the internal axes, (as in Fig. 4.28 (p. 107), **A-C**), then one sensor value has a zero amplitude. The internal masses must perform an oscillation with a suitable frequency and phase-shift in order to produce a coherent rolling behavior. In this way each particular behavior is characterized by an orbit in the sensorimotor space, and the orbits of different behaviors are also partially overlapping. In order to obtain a suitable visualization, we consider the angular velocities

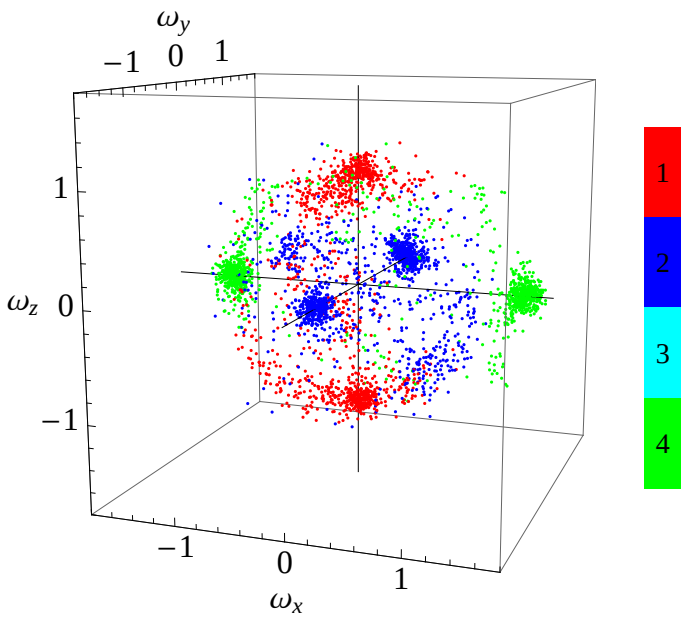


Figure 6.11: Partition of the behavior-space of the Spherical robot with 4 experts. Each point represents the state of the robot in terms of angular velocities around the three internal axes of the robot at a certain time. Note that the angular velocities are not directly accessible by the controller and by the experts. Nevertheless, a clear partition is observed, where only three experts are committed.
Parameters of homeokinetic controller: $\epsilon_C = \epsilon_A = 0.1$, update rate 100 Hz, extended world model (Section 4.8.4); for competition: $\epsilon_F = 0.001$, $p = 1$, $\tau_E = 50$.

$\omega_{x,y,z}$ around the three internal axes, as depicted in Fig. 6.10. A rolling behavior with fixed rotation axis and fixed velocity is represented by a single point in the space of angular velocities.

In a first experiment we provide only $r = 4$ experts and select a low learning rate and low penalty, such that only 3 experts are committed. The resulting clustering in terms of angular velocities is depicted in Fig. 6.11. Each of the three experts occupies the rotation around one particular internal axis, however, without discrimination of forward and backward motion.

In a longer experiment with $r = 20$ experts and suitably selected parameters we find a much more fine grain partition of the behavior-space, as depicted in Fig. 6.12. Not only is the symmetry broken between forward and backward rolling around the different axes, but also there are experts for different speeds around one and the same axis, e. g. expert numbers #3 and #4. It should be noted that the experts cannot sense the rotation speed directly, because their inputs are only axes orientations. So in terms of inputs and outputs (sensor and motor values) the clusters are actually periodic orbits in a six dimensional space. This demonstrates nicely the potential of the competing expert approach, which does not depend on the input space but only on the prediction performance.

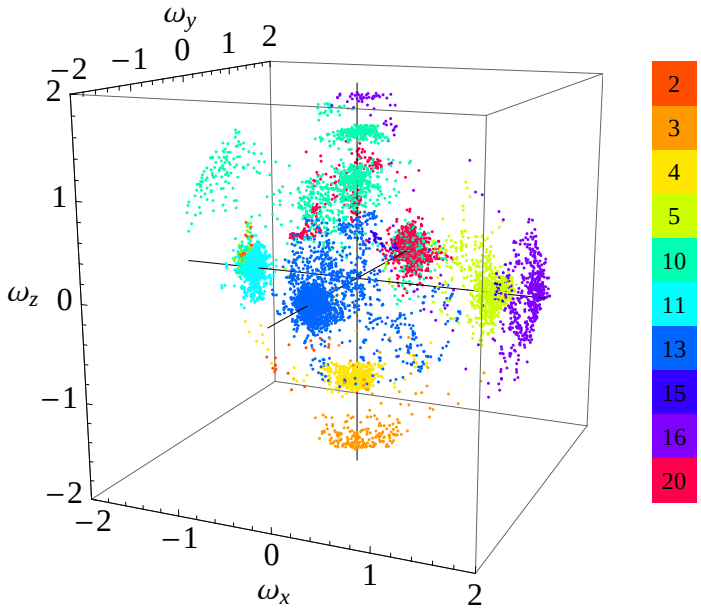


Figure 6.12: Partition of the behavior-space of the Spherical robot with 20 experts. Each point represents the state of the robot in terms of angular velocities around the axes of the robot at a certain time. For clarity only a selection of experts is drawn. In contrast to Fig. 6.11 the experts specialize to have a specific velocity.
Parameters of homeokinetic controller: $\epsilon_C = \epsilon_A = 0.1$, update rate 100 Hz, extended world model (Section 4.8.4); for competition: $\epsilon_F = 0.005$, $p = 10$, $\tau_E = 50$.

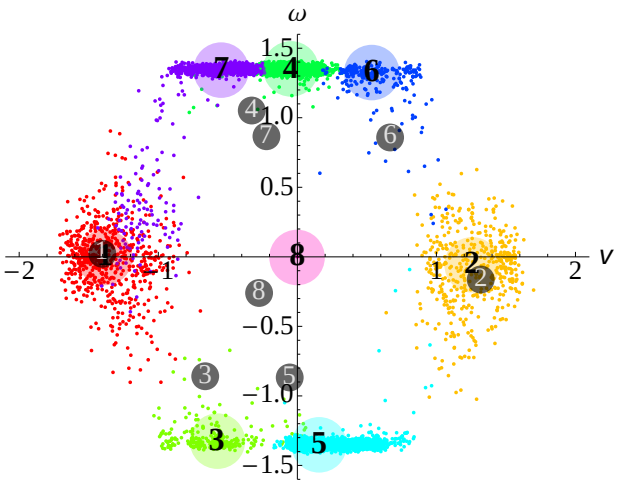


Figure 6.13: Attractor behaviors of the experts for the FourWheeled robot. The colored points and the transparent discs show the training data for the experts (points where the expert won the competition), see Fig. 6.5 (p. 155) for a description. The black disks mark the attractor behavior of each expert.

6.1.5 Experts as Controllers

The extraction of behaviors as discussed above can only be considered successful if the behaviors can be reproduced reliably by the experts. For that reason we will analyze the quality of the acquisition in terms of behaviors exhibited by the robot when the experts are used as controllers. In order to obtain behavioral primitives it is important that the behaviors are combinable and robust. Thus, it is important that there is an attractor behavior obtained when a particular expert network is controlling the robot. Additionally, the basin of attraction of each attractor behavior is of particular interest. This is the region of the behavior-space from which the attractor behavior is reached. If the basin of attraction is large or even spans the entire space of initial configurations then the expert can be activated independently of the state of the robot. This makes the sequencing of experts especially easy. Remember that the experts are closed-loop controller, hence, the transition from one expert to the next will most likely happen on a smooth transient, which we will see at the end of this section.

FourWheeled Robot

In the case of the FourWheeled robot we find for all committed experts stable attractor behaviors with global basins of attraction. This is not very surprising because the control of a behavior consists essentially of constant motor values. We use one by one the developed experts presented in Fig. 6.5 (p. 155) as controllers for the robot. Starting from different initial conditions we measured the behaviors exhibited after a transient phase.

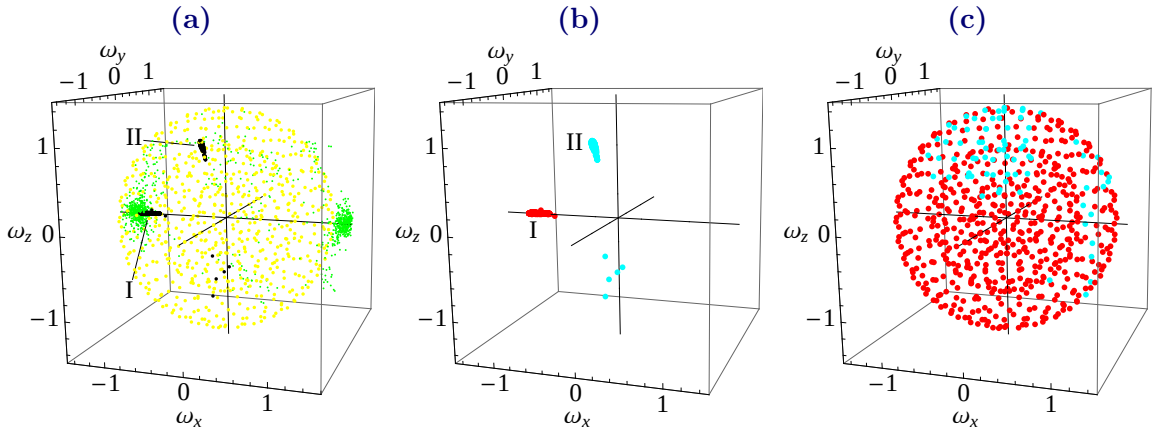


Figure 6.14: Attractor behaviors and their basins of attraction of one expert controlling the Spherical robot in the under-represented case of 4 experts. The expert #4 of Fig. 6.11 is used as an example. **(a)** Initial conditions (**yellow**), original winning points (**green**), and final states of the robot (**black**); **(b)** Final states are partitioned in 2 clusters as marked in **(a)**; **(c)** Initial conditions belonging to the two clusters. The cluster **I** (**red**) has a much larger basin of attraction.

For each expert there is a stable attractor behavior reached. These attractor behaviors are displayed in Fig. 6.13 together with the original partitioning of the behavior space. The shown behaviors are close to the original center of training points for the forward and backward driving expert. The training points are those where the particular expert won the competition and was allowed to learn the sensorimotor correspondence. With the experts controlling curved driving a small deviation from the training data center is observed, but the quality of the behavior is mostly preserved. Since expert #4 and #7 show very similar behavior the acquisition of 6 distinct behavioral primitives has been achieved. Even if the learning of the experts continues theses primitives are not forgotten due to the exponentially decreased learning rate of the experts, Eq. (6.21). Nevertheless, a certain fine tuning of the represented behaviors can still occur.

Spherical Robot

In the case of the Spherical robot, the control of the behaviors is much more complicated than for the wheeled robot, because the weights have to be coordinated with the orientation of the robot. We consider again the developed experts as controller for the robot. Firstly, we want to analyze what happens if too few experts are provided, as in Fig. 6.11 (p. 160), such that multiple behaviors are represented by the same expert. Let us consider the robot behavior when controlled by expert #4 from Fig. 6.11 without additional noise in the sensor values. Starting from a large number of different initial orientations and initial rolling velocities the behavior after a sufficiently long time was recorded, see Fig. 6.14(a),(b).

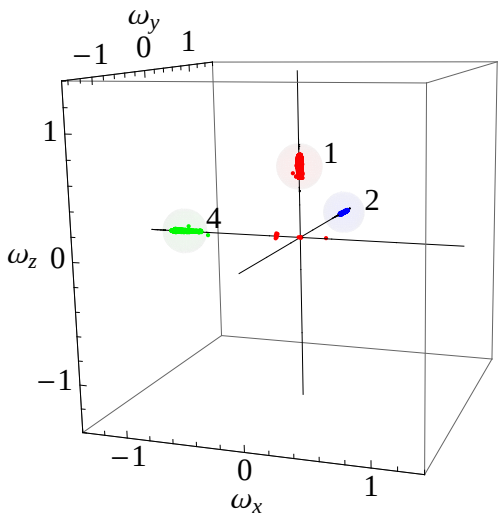


Figure 6.15: The main attractor behaviors of the three committed experts. For each committed expert (#1, #2, and #4) in Fig. 6.11 (p. 160) the attractor behavior, which has the dominant basin of attraction is presented. See also Fig. 6.14.

We observe two attractor behaviors which are not a single point in the space of angular velocities, but appear as small clouds. With closer inspection we find small limit cycles that corresponds to a slight precession[4] of the Spherical robot. The first attractor cloud, marked with **I**, lies inside the original behavior and represents the rolling motion around the first internal axis. The second cluster, marked with **II**, represents a spurious behavior, that is not connected to the training set. To determine the basin of attraction we associate the initial conditions to the two clusters and find that the first cluster has a large basin of attraction, whereas the behavior of the second cluster is only reached from a small subset of the initial conditions, see Fig. 6.14(c).

The remaining two committed experts have a very similar structure of attractors. In Fig. 6.15 the attractor behaviors with a large basin of attraction are displayed for all experts. We find that the main attractor behaviors represent the rotation around all three internal axes in one direction.

Let us now consider the case of many experts, as illustrated in Fig. 6.12 (p. 161). We find mostly one attractor per expert with global basin of attraction. A selection of attractors is displayed in Fig. 6.16. For each of the three internal axes, a behavioral primitive for forward and backward rolling has developed. Beside that, there are behaviors that correspond to a rotation around an axis different from the internal axes, e. g. expert #5 and #6. The attractors of these two experts show particularly well their orbital structure in the space of angular velocities. This usually corresponds to a slow precession movement of the Spherical robot.

[4]Precession refers to a cyclic change in the direction of the axis of a rotating object, e. g. a gyroscope.

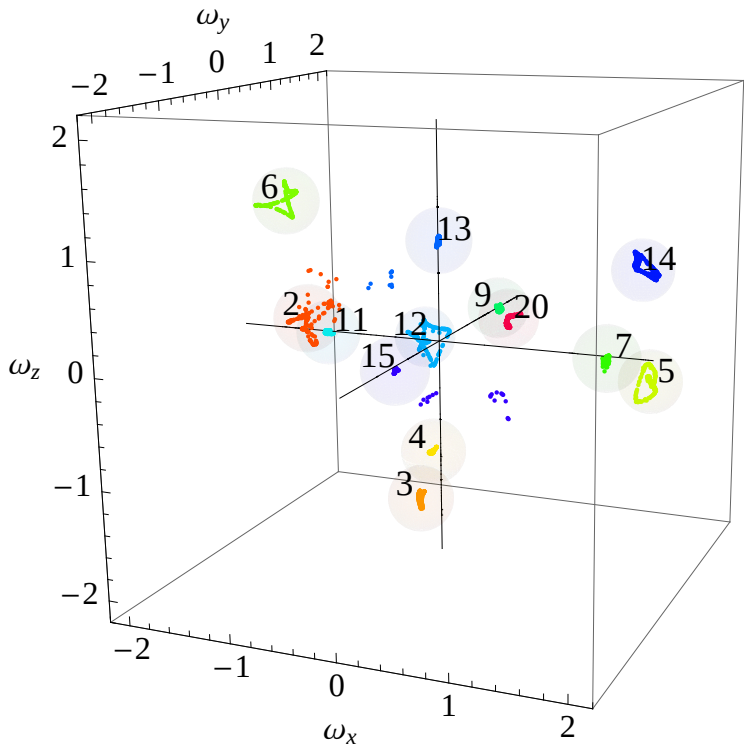


Figure 6.16: Attractor behaviors for the case of 20 experts with the Spherical robot. The colored points show the attractor behavior for each expert. For clarity only a selection of experts is displayed. Each transparent sphere marks the center of the point-cloud belonging to one expert.

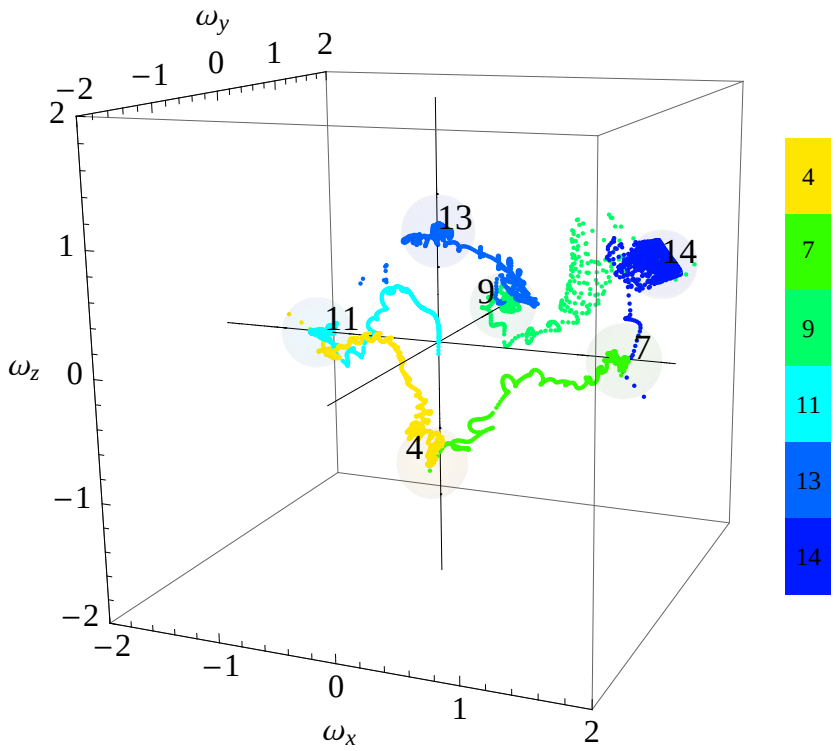


Figure 6.17: Transient behavior when the Spherical robot is controlled by a sequence of experts. Each expert of the sequence #11, #4, #7, #14, #9, and #13 was subsequently controlling the robot for 30 sec. The colors correspond to the controlling expert. The transparent spheres mark the attractor behaviors, see also Fig. 6.16. A smooth transient between the attractor behaviors is observed.

The attractor behaviors represent the repertoire of primitive behaviors that have been acquired during the online learning phase. In the here considered examples, the basins of attraction of these behaviors are global if sufficiently many experts are provided. This qualifies them as behavioral primitives because they can be arbitrarily sequenced to complex behaviors. In order to study the sequencing and the transient behaviors between the primitives we controlled to robot with a sequence of experts. Figure 6.17 shows the trajectory of the Spherical robot in the space of angular velocities for a sequence of 5 experts. Starting from a calm initial position ($\omega_{x,y,z} = 0$), the robot reaches the attractor behavior of the first expert (#11) on a smooth transient. After a certain time the control was switched to the next expert in the sequence and for each transition a smooth transient to the belonging behavior is observed. This shows that the experts, which have been acquired in a self-organized way, serve as combinable behavioral primitives.

6.1.6 Summary

The online acquisition of behavioral primitives from the behaving robot controlled by the homeokinetic controller was achieved with a set of competing experts. We proposed a novel competition algorithm in the winner-takes-all manner with a penalty for suboptimality and an individual learning rate annealing that leads to a suitable distribution of experts in the behavior-space with long-term stability and without overfitting. The extraction of behaviors was shown for the FourWheeled robot and for the Spherical robot with different number of experts. In both cases the method did not require a fine-tuning of parameters and showed a reliable partitioning of the behavior space. In both applications the trained experts have shown their capability to control the robot. Additionally the analysis of the stability of the reproduced behaviors revealed that in most cases the attractor behaviors of the experts have a global basin of attraction. This allows for a simple combination of behaviors by sequencing the experts. Thus, the experts represent behavioral primitives that are flexible and robust building blocks for complex behaviors. A subset of the here presented results has been published in [89].

6.2 Goal-Oriented Behaviors through the Combination of Primitives

In the remaining part of this chapter we will investigate how the behavioral primitives can be used to achieve goal-oriented behaviors. The system should learn to achieve a goal that is specified by rewards and punishments. We will use a higher level of learning that is to choose the right sequence of primitives to maximize the long-term reward. First, we will introduce reinforcement learning, which is the most prominent approach to this kind of learning problem. We will show how to use the behavioral primitives represented by the experts (Section 6.1) as discrete action in the reinforcement learning framework. We then apply the setup to an obstacle avoidance task using the FourWheeled robot and the Spherical robot.

6.2.1 Temporal Difference Learning – Reinforcement Learning

In this section we will formally introduce temporal difference learning and then describe the standard reinforcement learning algorithm named $\mathcal{Q}$-learning. Temporal difference (TD) learning is a method to learn the prediction of a quantity that depends on future values of a given signal, e. g. a reward signal [11]. TD learning is often used in reinforcement learning (RL) to predict the total amount of reward expected in the future. Interestingly, recent studies in neuroscience confirmed that animals must have a similar learning mechanism.

It was found in mammals that the firing rates of dopamine neurons[5] in the midbrain show a convincing correspondence to the reward error function of the temporal difference learning theory [144]. In machine learning many different solutions to the RL task have been proposed based on the TD learning algorithm called TD(λ) introduced by Richard S. Sutton [158]. The most prominent RL algorithms are $\mathcal{Q}$-learning [169], adaptive actor-critics algorithms [79], policy gradient methods [12] and ISO-learning [128]. A short overview of the literature was provided in Section 1.2. For our applications we will use the $\mathcal{Q}$-learning algorithm, because it is simple and is proven to find an optimal solution for a deterministic Markovian system [160].

Let us now go into more detail and formally define TD learning. The name 'temporal difference' comes from the use of differences between predictions over successive time steps. In an iterative manner the prediction is updated to bring it closer to the prediction of the same quantity at the next time step. Suppose the system received at each time step t a reward value $r_t \in \mathbb{R}$. The TD learning tries to predict the quantity

$$\tilde{r}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\,, \tag{6.24}$$

where γ is a discount factor, with $0 \leq \gamma < 1$. Intuitively $\tilde{r}_t$ is the discounted expected reward, where the far future is less important than the immediate future. TD learning exploits that we can write the expected reward iteratively as

$$\tilde{r}_t = r_{t+1} + \gamma \tilde{r}_{t+1}\,. \tag{6.25}$$

To predict the reward an input that characterizes the state of the system is required. Let us denote the input as s_t and the prediction function as $P_t(s_t)$, where the goal is to achieve $P_t(s_t) = \tilde{r}_t$. Note that the expected reward $\tilde{r}_t$ (Eq. (6.24)) implicitly depends on the current and future states via the reward signals $r_{t+\tau}$. The predictor carries a time index because it is gradually improved over time. Rewriting equation Eq. (6.25) using the predictor we obtain

$$P_t(s_t) = r_{t+1} + \gamma P_t(s_{t+1}) - \rho_{t+1}\,, \tag{6.26}$$

where ρ_{t+1} is the misfit or TD error. Note that we use the predictor at time t to predict the reward using the next input. Now, the predictor is updated to minimize the TD-error ρ. This leads to an iterative update rule

$$P_{t+1}(s_t) = P_t(s_t) + \epsilon_P \rho_{t+1}\,, \tag{6.27}$$

where $\rho_{t+1} = r_{t+1} + \gamma P_t(s_{t+1}) - P_t(s_t)$ and ϵ_P is a learning rate. Note that the state space is commonly assumed to be discrete. In the simplest case the predictor can be implemented

[5] The dopamine neurons regulate the release of the neurotransmitter dopamine that regulates the strength of synaptic connections for neurons that use dopamine as neurotransmitters, e. g. the dopaminergic neurons chiefly found in the midbrain.

as a look-up table. In the case of a continuous space the most commonly used variation applies linear function approximators [11]. More sophisticated versions can be found in [51].

TD-learning can thus be used to estimate the value of a certain state with respect to future rewards, however, it does not provide information on how to act in order to maximize this reward. This is the purpose of reinforcement learning algorithms, which add a policy to the system that decides about the actions to take. A simple but powerful RL algorithm is $\mathcal{Q}$-learning [169], that will be introduced in the following. The $\mathcal{Q}$ in its name is derived from the originally used symbol for value table $\mathcal{Q}_t(s,a)$ that assigns a value to each state-action pair (s,a), where s denotes a discrete state and a denotes a discrete action. Using TD learning, especially Eq. (6.27), the value table for the current state-action pair can be updated as follows:

$$\mathcal{Q}_{t+1}(s_t,a_t) = \mathcal{Q}_t(s_t,a_t) + \epsilon_P \left[r_{t+1} + \gamma \max_a \mathcal{Q}_t(s_{t+1},a) - \mathcal{Q}_t(s_t,a_t) \right] . \tag{6.28}$$

The value of the next state is obtained using the maximum over all possible actions, which is independent of the actually executed action. The selection of the action is called policy, and thus this is called off-policy update. Let us denote the policy as $\pi(\cdot)$ with

$$a_{t+1} = \pi(s_{t+1}, Q) . \tag{6.29}$$

The simplest way is to choose the action which promises the maximal reward, i. e. $a_{t+1} = \arg\max_a \mathcal{Q}_t(s_{t+1},a)$ with a high probability and choose a random action otherwise. Note that it is important to select suboptimal actions from time to time in order to explore the value landscape and to find optimal solutions. A more sophisticated policy uses the softmax function (which was also used in Section 6.1.1) to select good actions with a higher probability than actions with a low value.

Let us now consider a variation of the $\mathcal{Q}$-learning algorithm to perform an on-policy update, namely SARSA [138, 159]. The update formula for SARSA reads

$$\mathcal{Q}_{t+1}(s_t,a_t) = \mathcal{Q}_t(s_t,a_t) + \epsilon_P \left[r_{t+1} + \gamma \mathcal{Q}_t(s_{t+1},a_{t+1}) - \mathcal{Q}_t(s_t,a_t) \right] , \tag{6.30}$$

where a_{t+1} is given by the policy (Eq. (6.29)). Here, the actually performed action is used to estimate the value of the next step. Therefore the behavior of the algorithm depends on the policy. In our applications we will use SARSA.

A commonly used extension of TD-learning and reinforcement learning are eligibility traces [147]. The idea is to update not only the prediction for the previous state but also for a number of past states. Eligibility traces are often implemented with exponentially decaying memory with decay parameter λ. To each state-action pair an eligibility value $e_t(s,a)$ is assigned that accounts for how often this state-action pair was visited in the past:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise.} \end{cases} \tag{6.31}$$

The integration of eligibility traced into SARSA is straight forward, it leads to a transformation of Eq. (6.30) to

$$\mathcal{Q}_{t+1}(s,a) = \mathcal{Q}_t(s,a) + \epsilon_P \rho_{t+1} e_t(s,a)\,, \tag{6.32}$$

with

$$\rho_t = r_{t+1} + \gamma \mathcal{Q}_t(s_{t+1}, a_{t+1}) - \mathcal{Q}_t\,. \tag{6.33}$$

For $\lambda = 0$ this is equivalent to the original formulation (Eq. (6.30)), and with $\lambda = 1$ all elements of the table $\mathcal{Q}$ are updated each time. The integration into $\mathcal{Q}$-learning is bit more complicated, because it is an off-policy algorithm. In this case the eligibility traces have to be interrupted when exploratory actions are chosen. More precisely, we set all $e_t(s,a) = 0$ if $a_t \neq \arg\max_a \mathcal{Q}_t(s_{t-1}, a)$.

6.2.2 Experts as Discrete Actions

We propose to use the behavioral primitives, studied in Section 6.1, as discrete actions for the reinforcement learning. Thus we have $a \in \{1, \ldots, r\}$, where r denotes the number of experts. In applications to robots we find two problems with reinforcement learning and behavioral primitives. First, the typical duration of a behavior is unknown, e. g. a turning behavior has an intrinsic timescale, but the behavior to drive straight can have all possible durations. Hence, the selection of an uniform time-interval for RL updates and action selection is difficult, if not impossible. The second problem is that the robotic system is not necessarily Markovian. The Markov property states that the measured state describes the system sufficiently, i. e. there is not past dependence. For example, if a robot has a significant inertia, then the state would need to also capture the impulse of the robot to be Markovian. Therefore the design of the state-space for the RL is crucial. However, the state-space should also be as small as possible to keep the training time within feasible bounds.

We are now going to exploit the modeling capabilities of the experts, to cope with the delayed reaction of the physical body. Remember that the experts are able to predict future sensors values, see Eq. (6.5), which will be most correct if their preferred behavior is currently active. Thus, by comparing the prediction quality of all experts limited to the sensor values, we obtain a good measure of the actually exhibited behavior. To get in intuition, consider a simple driving robot with weak motors. Assume we have two experts, the first one for forwards driving and the second one for backwards driving, see Fig. 6.18. If now the first expert is controlling the robot it will drive forward, which is also reflected in the sensor values. However, if the second expert is activated the robot will not immediately drive backwards because of inertia, thus the sensor values will still report forward driving for a certain time after the switching of control and the prediction error of the first expert will be small. As soon as the sensor value reflects the backwards driving the prediction of

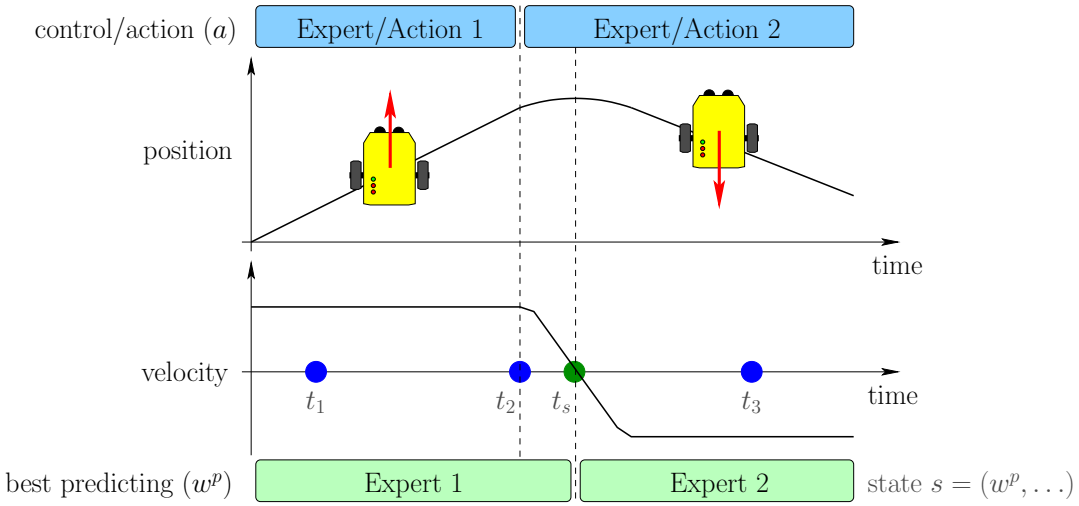


Figure 6.18: Illustration of a robot with large inertia and the best predicting expert as part of the state variable. Periodic RL-steps occur at t_1,t_2,t_3. The controlling expert is displayed on the top (**blue boxes**), which changes at t_2. Expert 1 moves the robot forward and expert 2 backwards. The inertia causes a slow transition period starting at t_2. As long as the velocity is positive expert 1 predicts best the sensor values ($w^p = 1$, **green boxes**). At t_s the velocity of the robot flips sign and expert 2 predicts best. Thus, at t_s a state change occurs and an additional RL-update takes place.

the second expert will be better than the prediction of the first expert. In summary, the index of the expert with the least sensor prediction error provides a generic and discrete measure of the actual behavior, independent of the controlling expert. We define the best sensor predicting expert w^p as

$$w^p = \arg\min_i \left(x_t - F_i(x_{t-1}, y_{t-1})_{(1,\dots,n)}\right)^2 , \tag{6.34}$$

where $F(\cdot)_{(1,\dots,n)}$ denotes the first n components of the output vector, namely only the sensory component, see Eq. (6.5). We propose to use w^p as part of the state variable for reinforcement learning, i. e. $s = (w^p, \dots)$.

Let us focus on the problem of update rate selection. For example, in an obstacle avoidance task the time interval between updates is supposed to be small in order to be able to react on an appearing obstacle. However, if the robot gets stuck in a corner and tries to get away there are many steps required until a reward in the obstacle-free part is received. This is often so long that the learning times become intractable if short update cycles are used. The approach we pursue runs the RL-update and action selection at fixed long time-intervals and at times of state changes. Since the state is a discrete value it is easy to detect a change. Figure 6.18 gives a schematic view of the entire scenario. Note that the change in the state can either originate from a change of the best predicting expert w^p (Eq. (6.34)) or from a change of the context sensor state. The latter can be for example a

discrete infrared sensor configuration. In any case, the change of the state indicates that a distinct situation is reached. This can be a direct consequence of the actions or an external influence, e. g. the encounter of an obstacle. To account for the inhomogeneous update cycles the reward has to be scaled with the interval length since the last update.

The system with w^p in the state variable behaves quasi-Markovian[6] when observed at the state changes. This is necessary because reinforcement learning algorithms require Markovian systems. Thus, with the here-proposed method it is possible to use reinforcement learning with robots that have large inertial effects.

6.2.3 Obstacle Avoidance

Let us now apply reinforcement learning with behavioral primitives to an obstacle avoidance task using the FourWheeled robot and the Spherical robot.

Application to the FourWheeled Robot

First we consider the FourWheeled robot again. Now the 6 infrared (IR) sensors as used, see Section 2.2.2. These sensors actively sense by emitting a beam of light and measure the strength of the reflection. Thus, they measure the distance to an reflective obstacle. If no obstacle is in the range of the sensor, then its value is 0 and if an obstacle is very close then the sensor value is 1. In between we use a linear characteristics. The task for the robot is to drive forward while avoiding obstacles. Hence, the reward is given by

$$r_t = v_t - 2 \max_i \left(x_t^{IR}\right)_i , \tag{6.35}$$

where v is the velocity of the robot (normalized to $[-1, 1]$) and x^{IR} is the vector of IR-sensor values. If the robot bumps into an obstacle the reward is -1, whereas if the robot drives in free space with maximal speed the reward is 1.

We use the SARSA(λ) variation of the $\mathcal{Q}$-learning algorithm, see Section 6.2.1. The selected action (a_t) decide which expert controls the robot at time t. We use the 7 committed experts, see Fig. 6.5 (p. 155), hence the action space contains 7 actions. The inertia of this robot is rather small such that we do not include w^p (Eq. (6.34)) in the state s in this example. Nevertheless, the state must incorporate the infrared sensor values in some way. We use a discrete infrared sensor state with four components: $s = (f_l, f_r, b, s_{lr})$. If the left-front IR sensor is non-zero then $f_l = 1$ otherwise $f_l = 0$. The same holds for f_r (right-front IR-sensor) and for b (back IR sensors). The term s_{lr} is either 0, 1, or 2 depending on whether the left sensor value is larger than the right sensor value or vice versa or both are zero. Thus, there are 24 possible states. The policy we use is a simple

[6] We cannot prove that the system is actually Markovian, but the dependence on the past is very small.

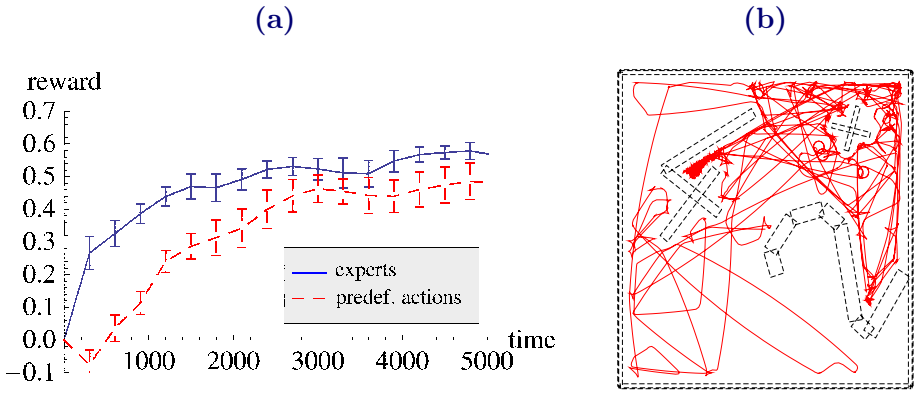


Figure 6.19: Comparison of the performance of reinforcement learning with experts opposed to reinforcement learning with predefined actions for the Four-Wheeled robot. **(a)** Mean accumulated reward over 300 sec sliding windows in a run of 5000 sec; **(b)** Trajectory of the wheeled robot (**red** line) after learning the obstacle avoidance task with behavioral primitives. **Dotted black** lines correspond to obstacles and walls. Parameters: $\epsilon_P = 0.1$, $\gamma = 0.7$, $\varrho = 0.1$, $\lambda = 0.7$, RL-update rate 5 Hz.

randomized strategy [81] that takes the action with the highest expected reward (greedy method), but with the probability of ϱ chooses an action at random. Hence,

$$a_{t+1} = \begin{cases} \arg\max_a \mathcal{Q}(s_t, a) & \text{with probability } 1 - \varrho, \\ \text{random action} & \text{with probability } \varrho. \end{cases} \tag{6.36}$$

For an evaluation we compared the performance of $\mathcal{Q}$-learning using the 7 experts with $\mathcal{Q}$-learning using predefined actions. These actions are given by the Cartesian product of the four quantized actuator states. In order to reduce the number of actions we use three discrete actions: forward rotation, backward rotation and no rotation, i. e. in total $4^3 = 81$ actions. Figure 6.19(a) compares the performance using the mean accumulated reward. Both methods lead to a positive overall reward, however with the behavioral primitives a higher reward is obtained after a short time. This result was expected given the difference in the size of the action space. For the robot controlled with behavioral primitives, the trajectory after learning the obstacle avoidance task is shown in Fig. 6.19(b). A large area of the cluttered environment is covered while a clear distance to the walls is maintained, see also [Video 13].

Application to the Spherical Robot

We also applied the learning scheme to the Spherical robot, which is in this experiment additionally equipped with six infrared (IR) sensors that extend the internal axes towards

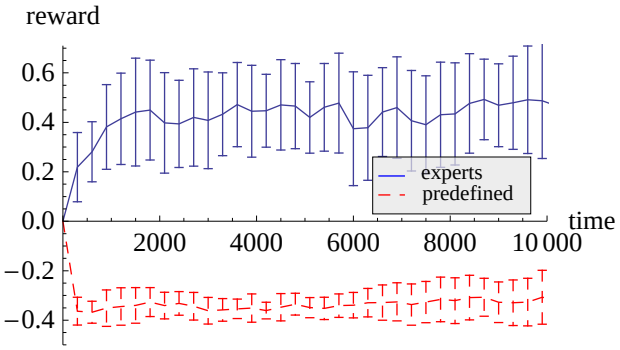


Figure 6.20: Comparison of the performance of reinforcement learning with experts opposed to reinforcement learning with predefined actions for the Spherical robot. Mean accumulated reward over 300 sec sliding windows in a run of 10000 sec. The low performance with predefined actions shows that a structuring of the behavior space is crucial. This is efficiently performed by the behavioral primitives. Parameters: $\epsilon_P = 0.1$, $\gamma = 0.9$, $\varrho = 0.1$, $\lambda = 0.7$, RL-update rate 5 Hz.

the outside of the robot as depicted in Fig. 2.12 (p. 25). For simplicity in this experiment only the walls and obstacles are supposed to reflect the IR beams, such that the floor does not influence the IR sensor values. The Spherical robot is very difficult to control because it shows huge inertial effects and the orientation and velocity of the robot influences the result of the actions significantly. The task for the robot is to roll around one particular axis while avoiding obstacles. Hence, the reward is given by (similar to Eq. (6.35))

$$r_t = \omega_{x,t} - 2 \max_i \left(x_t^{IR}\right)_i , \tag{6.37}$$

where $\omega_{x,t}$ is the normalized rotation velocity around the first internal axis at time t and x_t^{IR} is the vector of IR-sensor values.

The 20 behavioral primitives represented by the experts from Fig. 6.12 (p. 161) are used. The state $s = (w^p, d)$ comprises the best predicting expert w^p (Eq. (6.34)) and a discrete IR-state $d = (l, r, f)$, with $l, r, f \in \{0, 1\}$. Imagine the robot is rolling around the first internal axis with positive velocity, then we can define left, and right IR-sensors, which are the ones extending the first axis. If the left sensor measures an obstacle then $l = 1$ otherwise $l = 0$ and the same for the right sensor and r. If one of the remaining sensors is activated then $f = 1$, otherwise $f = 0$. Thus there are $20 \cdot 8 = 160$ states.

For comparison predefined actions are used in the same way as in the previous section, i. e. for the three motors 3 positions are given $(-1, 0, 1)$, which gives $3^3 = 27$ actions. The state space in this case contains the infrared state d as above and the current action, i. e. $s = (d, a)$ which results in 216 states. Figure 6.20 displays the performance of both algorithms in terms of the mean accumulated reward. The reinforcement learning with predefined actions is not able to achieve the task even after a long time. This is mainly due to the fact that the pure rolling mode is hard to achieve with the elementary actions.

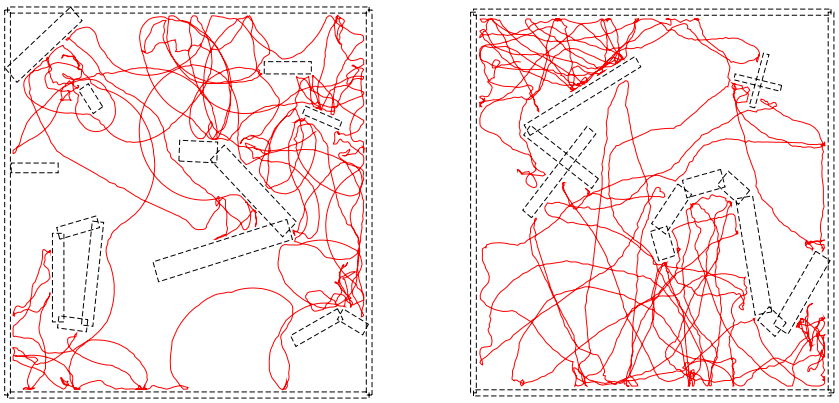


Figure 6.21: Trajectories of the Spherical robot after learning to move freely in two different environments based on the same set of behavioral primitives.

The performance using the experts is very effective. We performed the comparison to show that it is very important to structure the behavior space in complex robotic systems. Even though the dimensionality of the robot is low the dynamical complexity is high enough to make it impossible to achieve the task with an ad-hoc definition of the discrete actions.

The trajectories of the Spherical robot after learning the obstacle avoidance task in two different environments is shown in Fig. 6.21. The environments are much larger compared to the driving robot experiments, such that the robot is small in comparison. Most of the time the robot stays away from the walls, see also [Video 14].

6.3 Discussion

In the first part of this chapter we proposed a technique that enables a robot to acquire behavioral primitives from the self-organizing behavior generated by the homeokinetic controller [89]. The extraction and storage of the behaviors is done online with a set of competing experts implemented as neural networks in a closed-loop setup. We proposed a novel competition schema called 'suboptimality penalty with annealing' that is particularly suitable for the statistical properties of the self-organizing behaviors. In the considered applications, namely the FourWheeled driving robot and the Spherical robot, the expert networks are able to reproduce the learned behaviors in a self-stabilizing way, such that the term 'behavioral primitive' is justified. The acquired behavioral primitives are not merely a fixed sequence of actions but rather code for the dynamics, i. e. the sensorimotor coordination of that particular behavior. For example, in the case of the Spherical robot a single behavioral primitive accounts for the rolling mode around one particular axis including all possible transients to this behavior from other points in the behavior space, cf.

Fig. 6.16. This includes also the acceleration from a resting position or the change of the rotation axis. According to Schaal [140], these are the types of motor primitives that are required to be able to scale up to high-dimensional systems.

The acquisition of behavioral primitives solves the long standing problem that the self-organized behaviors are not persistent. The combination of homeokinetic controller and competing experts is an important step towards autonomous robot development. It allows a robotic system to explore its physical capabilities and to gradually build up a repertoire of behaviors. The competition algorithm was deliberately designed to have long-term flexibility while minimizing the interference with already learned behaviors.

The second part of this chapter was devoted to the creation of goal-oriented behavior based on the behavioral primitives. First we elaborated on the integration of the primitives into the $\mathcal{Q}$-learning algorithm. We proposed a generic way to obtain a measure of the current behavior by exploiting the expert's capability to predict the sensory dynamics. Inserting this into the state of reinforcement learning leads to a space-efficient quasi-Markovian description for robots with inertial effects. In a proof of concept the driving robot and the Spherical learned an obstacle avoidance task in cluttered environments. Especially the experiment with the Spherical robot demonstrated that the usage of behavioral primitives is distinctively more effective than a set of predefined actions. The self-organized acquisition of behavioral primitives is an promising way to structure the vast space of possible actions into a small set of potentially useful primitives. This enables the usage of simple reinforcement algorithms on high-dimensional continuous systems. The combination with more sophisticated reinforcement algorithms such as the adaptive actor-critics [79] can also eliminate the structuring of the context-sensor space, which was so far performed manually.

Chapter 7

Conclusions

I never think of the future.
It comes soon enough.

Albert Einstein (1879 – 1955)

In this thesis we investigated control algorithms for autonomous robots that learn to control the physical body without specific external pressures and continue to explore the dynamical properties of the body and the environment. We pursued three complementary objectives. First, we aimed at a better understanding of the homeokinetic controller, which led us to the improvement of its basic constitutes, especially with respect to the representation of the world. The second objective was to utilize this versatile controller to achieve goal-oriented behaviors, which we solved with novel mechanisms of guided self-organization. The third aim was to autonomously acquire a pool of reusable behavioral primitives that can be used to perform tasks.

The control algorithms used in this thesis are based on the principle of homeokinesis [39, 44] that we described in Chapter 3. Essentially, the idea of homeokinesis is to keep the dynamical regime of the sensorimotor loop in a range, where high sensitivity to sensory inputs is assured, but chaotic behavior is avoided. The representation of the sensorimotor loop in the homeokinetic control paradigm is a self-referential stochastic dynamical system. The parameters are self-regulated such that the system is slightly above the bifurcation point[1].

First, we created an appropriate workbench for our studies, namely a full-fledged robot simulator called LpzRobots [91] including many virtual robots with different complexities (Chapter 2). This has proven to be very useful to test hypotheses and control algorithms on a variety of robots. LpzRobots allows for physically realistic simulations of complex rigid body objects and features a novel support for material, different motors and sensors,

[1] We mean the bifurcation point of the deterministic system (without noise). With noise there is an effective bifurcation region [96].

graphical rendering, an interface to interactively change parameters, and so forth. The simulator is an open source project and is used by other researchers as well. Apart from the virtual robots, we also constructed a real robot, namely the Rocking Stamper [43] (Section 2.2.3).

With the help of the computer simulations we applied the homeokinetic control to more complex robots than those real robots that were available to our group. In these systems we analyzed and illustrated the properties of the control algorithm. We have demonstrated with a set of simulated robots the emergence of highly body and environment-related behaviors, even in systems where the control of these behaviors by hand is complicated [43, 46] (Chapter 4). For a similar study see [62]. In most cases those are not merely random movements, but develop into coordinated whole body motions where all degrees of freedom are actively participating. The homeokinetic control can be interpreted as an implementation of the ideas of cognitive embodied systems [122, 124]. Nevertheless, the application is limited to robots with purely proprioceptive sensors. In systems with many degrees of freedom we identified a mismatch between theoretical predictions and experiments, which was mainly due to the insufficiency of the world model. The tendency of the parameter dynamics to enter regimes of high frequency oscillations in the state dynamics is not fully understood.

Extensive simulations with different robots revealed that there is still necessity for improvement and extension of the original homeokinetic controller. On the side of the overall controller algorithm we improved the robustness by introducing different types of regularizations. The formalization of the dynamical system in terms of motor space coordinates enables the efficient integration of many sensors. Before, the system was limited to maximally the same number of sensors as motors. Apart from that, an additional adaptive model of the prediction error was proposed, allowing for its direct minimization in setups with multiplicative noise. For setups with additive noise this remains an open problem. An important constitute of the homeokinetic controller is the adaptive internal world model that is learned online. We investigated two problems arising from the simultaneous learning of the controller and internal model. The first is the effect of cognitive deprivation, which occurs if the actions of the controller are constantly limited to a subspace of the available action space. We showed analytically and experimentally that this problem is solved by the homeokinetic controller in a natural way by producing explorative actions pointing into unexplored subspaces [45]. The second problem arises when the internal world model is extended to receive sensory inputs in addition to the motor values. Such an extension leads to an ambiguity between self-induced and environmentally-induced changes in the sensor values. This ambiguity must be resolved because knowledge about the actual effects of the actions is required to regulate the controller parameters to the appropriate regime. We proposed a general solution by specifying a bias towards self-induced observations and provided experimental evidence for its functional efficiency. A computationally optimized variation uses a discount in the learning rule.

Our extensions enable the integration of exteroceptive[2] sensors that allow the robot to perceive changes in the environment. Furthermore, the new controller can cope with action-independent dynamics of the body or within the environment, for instance those brought about by inertial effects or moving objects. The extension of the controller network to multiple layers or to include recurrent connections remains open.

The second part of this thesis is concerned with the generation of goal-oriented behaviors based on the homeokinetic control. We established several ways to shape the self-organization process using specific goals (Chapter 5). On a general level, we proposed a method to integrate additional constraints, in the form of energy functions, to the learning dynamics of the homeokinetic controller. This makes it possible to combine supervised learning with self-organization of behavior. For example, we demonstrated teaching in terms of desired motor and sensor patterns. Using such mechanisms we derived a method that we called cross-motor teaching, which allows for a specification of certain symmetries in the desired behavioral patterns or in the morphology of the physical system and thus reduce the effective dimensionality of the system. This is especially important in cases where useful behaviors lie on a manifold of small codimension, which are very rarely found by the self-organizing controller. We demonstrated the effectiveness of cross-motor teaching with the example of the high-dimensional ARMBAND robot. The robot developed a complex forward and backward locomotive behavior depending on the specified symmetries[3]. Only a certain class of behaviors are possible to stimulate with this form of guidance, namely those which are sufficiently constrainable by cross-motor symmetries. A formal description was not elaborated upon here. The cross-motor teaching needs to be specified externally, which requires a certain insight into the dynamics of the behaviors. A learning of these connections presents an interesting topic for further investigations. Another way to guide the self-organization is to modulate the learning rate, such that wanted behaviors persist longer and unwanted ones are left quicker in comparison to the unguided case [90] (Chapter 5). This is a more generic way of shaping the behavior, but conceptually only those behaviors can be obtained that are also exhibited in the unguided case.

It is now possible to guide the emergence of sensorimotor coordination towards desired behaviors. However, in the current setup the behaviors are still impermanent and cannot be retrieved on demand. To address this shortcoming we designed an additional learning system that acquires and stores behavioral primitives (Chapter 6). We used a competing expert schema [117] with a novel competition principle named suboptimality penalty with annealing. We achieved an online partitioning of the behavioral space, such that each expert learns a particular behavior [89]. When the robot was controlled by a particular expert, a stable behavior with a global basin of attraction was observed in most of the considered cases. These are necessary properties for the further combination of the primitives.

[2]Exteroceptive sensors perceive the environment directly in contrast to the proprioceptive sensors which measure the position of body parts. Note that before the extended world model, only proprioceptive sensors have been possible.

[3]The locomotion behaviors are almost never found by the original controller and are now established within short time.

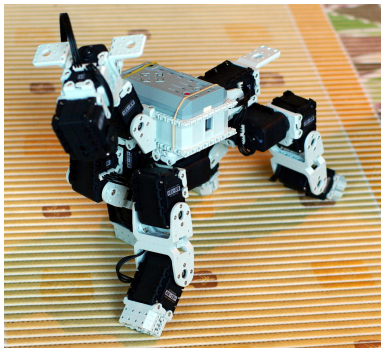


Figure 7.1: Bioloid Puppy robot by Robotis [135] with 15 DoF.

Thus, the set of experts represents a repertoire of combinable behavioral primitives. Note that this set is obtained in an unsupervised way, without providing specific information. Thus, we see this work as a promising approach for developmental robotics [20, 85]. The huge space of possible action sequences is reduced to a small set of potentially useful primitives. These can be used to solve tasks, which we have demonstrated using reinforcement learning techniques. The behavioral primitives served in this way as discrete actions. We illustrated the method on the example of an obstacle avoidance task, where we have shown that it is much more effective to combine the reinforcement learning with the acquired behavioral primitives than with a set of externally predefined actions, especially for robots with dynamically complex behaviors.

Perspectives

Many learning methods struggle with the curse of dimensions occurring in realistic robotic setups. The self-organization of control offers a way to structure the vast space of possible sensorimotor mappings to a smaller set of body and environment-related behaviors. The mechanism of guided self-organization can be used to further constrain the behaviors space, such that suitable modes are developed quickly. This is especially useful in combination with the acquisition of behavioral primitives to scale up to high-dimensional systems, for example to our new Dog-like robot, see Fig. 7.1.

The guidance of self-organizing behavior is also applicable in the context of evolutionary robotics. It was shown that, in general, lifetime plasticity can speed up the evolutionary process and yield more powerful behaviors [103, 175]. The combination of homeokinetic control with evolution algorithms was already suggested a decade ago [48]. Nevertheless, the suitable mechanisms to influence the self-organization process have been missing. The combination is now possible with the mechanisms proposed here. For instance, only the configuration of the cross-motor teaching could be evolved, which has a much lower dimension than the full set of controller parameters. Additionally, the controller remains adaptive and is likely to cope with different environmental conditions. In contrast to other

learning mechanisms, the homeokinetic controller generates coherent behaviors by itself, such that a much smaller amount of information has to be genetically encoded. Apart from the combination with evolutionary robotics, we hope that our work facilitates more studies of the combination of goals and self-organization that has just started to attract scientific attention [130].

A promising approach to complex control networks is based on recent findings in neuroscience. It was revealed that the bursts of neural activity in cortical slices show a self-organized critical behavior [16], which was also implemented in artificial neural networks [82]. Such a network regulates the synaptic connections in a way that responses on all scales are observed, which will bring about activity in the sensorimotor loop. However, the integration of additional learning mechanisms, e. g. to maintain predictability or to pursue goals, is not solved yet.

Another fascinating topic is the interaction of multiple homeokinetic agents, where first insights have been obtained by the author [92]. According to Luc Steels, one of the research challenges in the evolution of communication is "how populations of agents can self-organize a shared repertoire of discrete building blocks grounded in a continuous physical medium" [151]. Thus, the emergence of social interaction, synchronization phenomena and the negotiation of a communication schemata are interesting directions for further investigations.

The application of homeokinetic control is especially promising for those robots that are oriented towards entertainment, e. g. the AIBO robot by Sony [172] or the BIOLOID robots by Robotis [135]. In this field it is important that the robot is self-developing and capable of learning new things, because it makes them much more exciting. We reckon that the application to traditional robotics and to prosthetics has become more realistic with the here-proposed mechanisms. We see major strengths of our approach in the learning of new skills and in unforeseen situations. The reliable performance of tasks can be achieved in combination with traditional learning algorithms. However, there are many open questions, for example, how lifelong adaptation of the behavioral primitives can be obtained and how to achieve a balance between self-organized skill learning and goal-oriented learning and so forth.

The consideration of the homeokinetic principle in the framework of information theory is also very interesting, as it enables an analysis of the system from a different perspective. For instance, whether the control parameters are optimally regulated in terms of the flow of information through the system [8, 41]. Furthermore, the information theoretical interpretation might give rise to a different formulation of the learning dynamics.

There remains much to be done, but the path ahead of us is full of excitement.

Appendix A

A.1 Derivation of Matrix Calculation Rules

This section contains the derivation of the matrix calculation rules that were given in Section 4.1.3 (p. 53).

Let us first consider Eq. (R6), i. e.

$$\frac{\partial}{\partial X} a^\top X b = a b^\top .$$

Let X be an $m \times n$ matrix. We can write $a^\top X b$ as

$$a^\top X b = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i X_{ij} b_j . \tag{A.1}$$

Since a and b are defined to be independent of X we find for the derivative with respect to an element X_{kl}:

$$\frac{\partial}{\partial X_{kl}} \sum_{i=1}^{m} \sum_{j=1}^{n} a_i X_{ij} b_j = a_k b_l . \tag{A.2}$$

This gives in matrix notation

$$\frac{\partial}{\partial X} a^\top X b = (a_k b_l)_{\substack{k=1,\dots,m \\ l=1,\dots,n}} = a b^\top \tag{A.3}$$

□

The derivation of equation Eq. (R7), i. e.

$$\frac{\partial}{\partial X} a^\top g(Xb + c) = (a \circ g') b^\top$$

goes along the same line as above. Again a, b, c are defined to be independent of X and we find component-wise

$$\frac{\partial}{\partial X_{kl}} \sum_{i=1}^{m} a_i g \left(\sum_{j=1}^{n} (X_{ij} b_j) + c_i \right) = a_k g' \left(\sum_{j=1}^{n} X_{kj} b_j + c_k \right) b_l \,. \tag{A.4}$$

Since $g(\cdot)$ is a component-wise function the term $g' \left(\sum_j X_{kj} b_j + c_k \right)$ is the k-th element of the result vector of $g'(Xb + c)$ and hence we write only g'_k. In matrix notation we can use the row-wise multiplication and we obtain

$$\frac{\partial}{\partial X} a^\top g(Xb + c) = (a_k g'_k b_l)_{\substack{k=1,\dots,m \\ l=1,\dots,n}} = (a \circ g') b^\top \tag{A.5}$$

□

The derivation of Eq. (R9), i. e.

$$\frac{\partial A^{-1}}{\partial X} = -A^{-1} \frac{\partial A}{\partial X} A^{-1}$$

where A is a square matrix can be obtained by considering the derivative of $A^{-1}A$ first, i. e.

$$\frac{\partial A^{-1} A}{\partial X} = \frac{\partial A^{-1}}{\partial X} A + A^{-1} \frac{\partial A}{\partial X} = 0 \,. \tag{A.6}$$

Multiplying Eq. (A.6) with A^{-1} from the right and rearranging the terms gives directly Eq. (R9). □

A.2 Convergence of Enhanced World Model in a Simplified System

This section provides the calculations of the limit behavior of the learning dynamics of the enhanced world model, Section 4.8.5, in a simplified linear world with a fixed linear controller.

Let the controller be linear and fixed controller, Eq. (4.92), as

$$y_t = C x_t \tag{A.7}$$

and the toy world be given by (Eq. (4.93))

$$x_t = \mathcal{A} y_{t-1} + \mathcal{S} x_{t-1} \,. \tag{A.8}$$

Let us repeat the prediction errors used for the world model matrices: The prediction error used for adapting A is (Eq. (4.108))

$$\xi'_t = x_t - (Ay_{t-1} + (1-\delta)Sx_{t-1} + b) \tag{A.9}$$

and the one used for S and b is

$$\xi_t = x_t - (Ay_{t-1} + Sx_{t-1} + b)\,. \tag{A.10}$$

The learning dynamics Eqs. (4.109–4.111) converges if the prediction errors ξ_t and ξ'_t are zero. Thus, setting Eq. (A.8) into Eq. (A.9) and $\xi'_t = 0$ we get (omitting the time index)

$$\mathcal{A}y + \mathcal{S}x = Ay + (1-\delta)Sx + b\,. \tag{A.11}$$

The bias b vanishes, such that we can omit it. Substituting y using Eq. (A.7) and resolving for A yields

$$A = \mathcal{A} + (\mathcal{S} - (1-\delta)S)C^{-1}\,. \tag{A.12}$$

Doing the same with Eq. (A.10) gives

$$S = (\mathcal{A} - A)C + \mathcal{S}\,. \tag{A.13}$$

Putting Eq. (A.12) in Eq. (A.13) reveals that

$$S = (1-\delta)S\,, \tag{A.14}$$

which implies that $S = 0$. Setting this into Eq. (A.12) gives

$$A = \mathcal{A} + \mathcal{S}C^{-1}\,. \tag{A.15}$$

A.3 Experiment using the Enhanced World Model

This section present the results of the experiment with the Spherical robot and the enhanced world model with discounted learning as proposed in Section 4.8.5. The experiment is identical to the one conducted with the extended world model in Section 4.8.4. The Spherical robot (Section 2.2.5) is particularly suitable for the application of the new world model because it shows a sensor dynamics that is partly independent of the actions. This occurs especially when the internal masses are located close to the center and the robot is rolling. The evolution of the behavior and the parameters during the experiment are depicted in Fig. A.1. The behavior is very similar to the one observed in Section 4.8.4 (Fig. 4.28), but we find a slower change of behavior. For a discussion of the parameter dynamics we refer to Section 4.8.4.

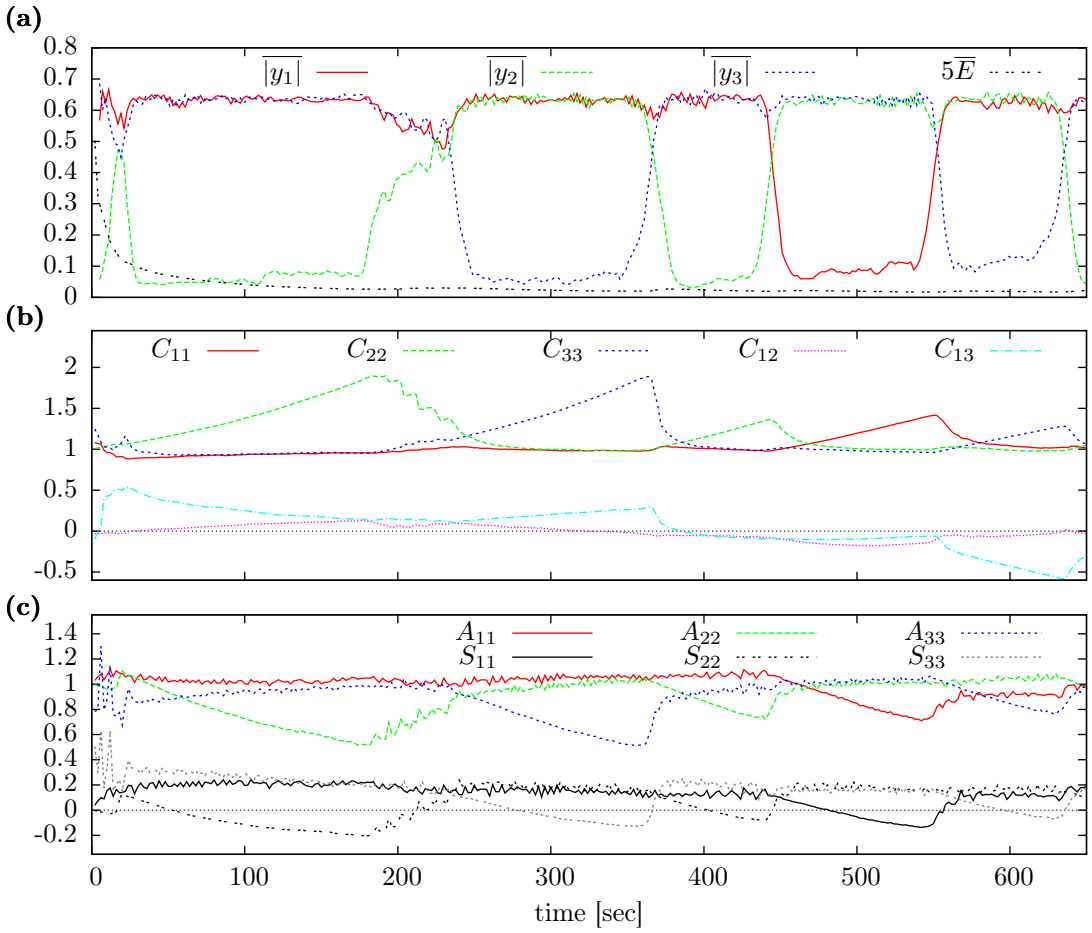


Figure A.1: Smoothly behaving Spherical robot with enhanced world model using Eqs. (4.109–4.111). The error function has a low value and the behavior is smooth but still diverse. **(a)** Envelop of motor commands and the error averaged over 10 sec (scaled for visibility). **(b)** Diagonals and two non-diagonal elements of the controller matrix C; **(c)** Diagonals of the world model matrices A and S. Parameters: update rate $100\,Hz$, $\epsilon_C = \epsilon_A = 0.1$, $\delta = 0.005$.

Video References

The following videos can be found on the web-page
http://robot.informatik.uni-leipzig.de/martius/thesis.

[Video 1] ROCKING STAMPER: starts to move and shows sensitive reaction on sensors, Section 4.3 (p. 67). start_sensitivity.mpg

[Video 2] ROCKING STAMPER: Adaptation to changed sensor setup and to a disabled sensor, Section 4.3 (p. 67). sensor_flapped_disabled.mpg

[Video 3] ROCKING STAMPER: Close view of hardware and walk-like behavior, Section 4.3 (p. 67). close_walklike.mpg

[Video 4] BARREL robot: Sweep through behavior space, Section 4.4.1 (p. 68). barrel.avi

[Video 5] SPHERICAL robot: Waxing and waning of rolling modes around different axes, Section 4.4.2 (p. 71). Sphere_IR_roll_different_axis.avi

[Video 6] TWOWHEELED robot: At the start the controller is restricted. After the restriction is released informative actions are performed, Section 4.5.4 (p. 79). cogdepr_2wheeled.avi

[Video 7] SNAKE robot: Low-dimensional modes, Section 4.6 (p. 86). Snake1D_lowdimmode.avi

[Video 8] TWOWHEELED robot: reaction of the robot to impulse shaped actions, Section 4.8.4 (p. 104). nimm2_impuls.avi

[Video 9] SNAKE robot: Behavior with continuity preference and extended world model and advanced sensor setup, Section 4.8.7 (p. 112). Snake1D_withExt.avi

[Video 10] ARMBAND robot with cross-motor teaching: Slow locomotive behavior with different postures (small guidance factor)), Section 5.2.2 (p. 133). wheelie_golden_rolling_jumping.avi

[Video 11] ARMBAND robot with cross-motor teaching: Fast locomotive behavior with small exploratory actions (normal guidance factor), Section 5.2.2 (p. 133). wheelie_golden_wobble_rolling.avi

[Video 12] ARMBAND robot with cross-motor teaching: Fast locomotive behavior with change in direction of motion after couplings were swapped, Section 5.2.2 (p. 133). wheelie_golden_teaching_pointsym.avi

[Video 13] FOURWHEELED robot: obstacle avoidance using reinforcement learning and behavioral primitives, Section 6.2.3 (p. 173).
FourWheeled_obstacle_avoidance.avi

[Video 14] SPHERICAL robot: obstacle avoidance using reinforcement learning and behavioral primitives, Section 6.2.3 (p. 175).
spherical_obstactle_avoid_square.avi

Bibliography

[1] D. Aberdeen. POMDPs and policy gradients. In *Proc. of the Machine Learning Summer School (MLSS)*, Canberra, Australia, 2006.

[2] L. Abraham and K. Shaw. *Dynamics, The Geometry of Behaviour*. Addison-Wesley, 1992.

[3] S. Amari. Natural gradients work efficiently in learning. *Neural Computation*, 10, 1998.

[4] M. A. Arbib. Perceptual structures and distributed motor control. In V. B. Brooks, editor, *Handbook of Physiology – The nervous system*, volume II, chapter Motor Control, pages 1449–1480. American Physiological Society, Bethesda, MD, 1981.

[5] W. R. Ashby. *Design for a Brain*. Chapman and Hill, London, 1954.

[6] Atmel Corporation. 8-bit avr microcontroller. http://www.atmel.com/products/avr/, 2008.

[7] F. Aurenhammer and R. Klein. Voronoi diagrams. In J.-R. Sack, editor, *Handbook of Computational Geometry*, chapter 5, pages 201–290. North Holland, Amsterdam, Netherlands, 2000.

[8] N. Ay, N. Bertschinger, R. Der, F. Güttler, and E. Olbrich. Predictive information and explorative behavior of autonomous robots. *The European Physical Journal B*, 63(3):329–339, 2008.

[9] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: an explanation of $1/f$ noise. *Physical Review Letters*, 59:381–384, 1987.

[10] H. B. Barlow. Critical limiting factors in the design of the eye and visual cortex. *Proc. R. Soc. Lond. B*, 212:1–34, 1981.

[11] A. G. Barto. Temporal difference learning. *Scholarpedia*, 2(11):1604, 2007.

[12] J. Baxter and P. L. Bartlett. Direct gradient-based reinforcement learning. In *Proc. of the Intl. Symposium on Circuits and Systems*, pages III–271–274, 2000.

[13] J. Baxter, A. Tridgell, and L. Weaver. Learning to play chess using temporal differences. *Machine Learning*, 40(3):243–263, 2000.

[14] L. Bayindir and E. Şahin. A review of studies in swarm robotics. *Turkish Journal of Electrical Engineering*, 2007.

[15] R. D. Beer. A dynamical systems perspective on autonomous agents. Technical report, Artificial Intelligence, 1992.

[16] J. Beggs and D. Plenz. Neuronal avalanches in neocortical circuits. *Journal of Neuroscience*, 23:11167–11177, 2003.

[17] G. Beni. From swarm intelligence to swarm robotics. In E. Şahin and W. Spears, editors, *Swarm Robotics: State-of-the-art Survey*, LNCS. Springer-Verlag, 2000.

[18] C. Bereiter. Towards a solution to the learning paradox. *Review of Educational Research*, 55(2):201–226, summer 1985.

[19] N. A. Bernstein. *The Co-Ordination and Regulation of Movements*. Pergamon Press, 1967.

[20] L. Berthouze and G. Metta. Epigenetic robotics: modelling cognitive development in robotic systems. *Cognitive Systems Research*, 6(3):189–192, September 2005.

[21] W. Bialek, editor. *Princeton Lectures on Biophysics*, chapter Optimal signal processing in the nervous system, pages 321–401. World Scientific, Singapore, 1992.

[22] W. Bialek. Thinking about the brain, July 2002.

[23] R. Bianco and S. Nolfi. Toward open-ended evolutionary robotics: evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 4:227–248, 2004.

[24] R. A. Brooks. A robust layered control system for a mobile robot. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.

[25] R. A. Brooks. Intelligence without representation. *Artificial Intelligence Journal*, 47:139–159, 1991.

[26] W. B. Cannon. The wisdom of the body, 1932.

[27] L. A. Celiberto, Jr., C. H. Ribeiro, A. H. Costa, and R. A. Bianchi. *Heuristic Reinforcement Learning Applied to RoboCup Simulation Agents*, pages 220–227. Springer-Verlag, Berlin, Heidelberg, 2008.

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

[29] S. Chikazumi and S. H. Charap. *Physics of Magnetism*. Krieger Pub Co, 1978.

[30] J. Choi, R. B. Wehrspohn, and U. Gösele. Mechanism of guided self-organization producing quasi-monodomain porous alumina. *Electrochimica Acta*, 50(13):2591–

2595, 2005.

[31] A. Clark. *Being There: Putting Brain, Body, and World Together Again*. MIT Press, 1998.

[32] Cyberbotics Ltd. Webots 5.0 robot simulator. http://www.cyberbotics.com, 2008.

[33] K. Dautenhahn. A paradigm shift in artificial intelligence: Why social intelligence matters in the design and development of robots with human-like intelligence. In Lungarella et al. [84], pages 288–302.

[34] K. Dautenhahn and C. L. Nehaniv, editors. *Imitation in animals and artifacts*. MIT Press, Cambridge, MA, USA, 2002.

[35] P. Dayan and T. J. Sejnowski. TD(λ) converges with probability 1. *Machine Learning*, 14(3):295–301, 1994.

[36] E. de Margerie, J.-B. Mouret, S. Doncieux, and J.-A. Meyer. Artificial evolution of the morphology and kinematics in a flapping-wing mini UAV. *Bioinspiration and Biomimetics*, 2:65–82, 2007.

[37] E. L. Deci and R. M. Ryan. *Intrinsic Motivation and Self-Determination in Human Behavior (Perspectives in Social Psychology)*. Springer, August 1985.

[38] J.-L. Deneubourg and S. Goss. Collective patterns and decision making. *Ethology, Ecology and Evolution*, 1(4):295–311, December 1989.

[39] R. Der. Self-organized acquisition of situated behavior. *Theory in Biosciences*, 120:179–187, 2001.

[40] R. Der. Autonomous self-organization of behavior. unpublished, 2005. Working paper.

[41] R. Der, F. Güttler, and N. Ay. Predictive information and emergent cooperativity in a chain of mobile robots. In S. Bullock, J. Noble, R. Watson, and M. A. Bedau, editors, *Proc. Artificial Life XI*, pages 166–172. MIT Press, Cambridge, MA, 2008.

[42] R. Der, M. Herrmann, and R. Liebscher. Homeokinetic approach to autonomous learning in mobile robots. In R. Dillman, R. D. Schraft, and H. Wörn, editors, *Robotik 2002*, number 1679 in VDI-Berichte, pages 301–306. VDI, 2002.

[43] R. Der, F. Hesse, and G. Martius. Rocking stamper and jumping snake from a dynamical system approach to artificial life. *Adaptive Behavior*, 14(2):105–115, 2006.

[44] R. Der and R. Liebscher. True autonomy from self-organized adaptivity. In *Proc. of EPSRC/BBSRC Intl. Workshop on Biologically Inspired Robotics*, HP Labs Bristol, 2002.

[45] R. Der and G. Martius. From motor babbling to purposive actions: Emerging self-exploration in a dynamical systems approach to early robot development. In S. Nolfi,

G. Baldassarre, R. Calabretta, J. C. T. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, and D. Parisi, editors, *Proc. From Animals to Animats 9 (SAB 2006)*, volume 4095 of *LNCS*, pages 406–421. Springer, 2006.

[46] R. Der, G. Martius, and F. Hesse. Let it roll – emerging sensorimotor coordination in a spherical robot. In L. M. Rocha, L. S. Yaeger, M. A. Bedau, D. Floreano, R. L. Goldstone, and A. Vespignani, editors, *Proc, Artificial Life X*, pages 192–198. Intl. Society for Artificial Life, MIT Press, August 2006.

[47] R. Der, G. Martius, F. Hesse, and F. Güttler. Videos of self-organized behavior in autonomous robots. http://robot.informatik.uni-leipzig.de/videos, 2009.

[48] R. Der, U. Steinmetz, and F. Pasemann. Homeokinesis - a new principle to back up evolution with learning. In *Proc. Intl. Conf. on Computational Intelligence for Modelling, Control and Automation (CIMCA 99)*, volume 55 of *Concurrent Systems Engineering Series*, pages 43–47, Amsterdam, 1999. IOS Press.

[49] J. Deutscher, A. Blake, and I. Reid. Articulated body motion capture by annealed particle filtering. In *IEEE Conf. on Computer Vision and Pattern Recognition.*, pages 126–133, 2000.

[50] Y. Dongyong, J. Jingping, and Y. Yuzo. Distal supervised learning control and its application to CSTRsystems. In *SICE 2000. Proc. of the 39th SICE Annual Conference.*, pages 209–214, 2000.

[51] A. Donzée. On temporal difference algorithms for continuous systems. In J. Filipe, J. Andrade-Cetto, and J.-L. Ferrier, editors, *ICINCO*, pages 55–62. INSTICC Press, 2005.

[52] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

[53] J. A. Fodor. Fixation of belief and concept acquisition. In M. Piatelli-Palmerini, editor, *Language and learning: The debate between Jean Piaget and Noam Chomsky*, pages 142–149, Cambridge, MA, 1980. Harvard University Press.

[54] L. Frommberger. A generalizing spatial representation for robot navigation with reinforcement learning. In *Proc. 20th Intl. Florida AI Research Society Conf. (FLAIRS-2007)*, pages 586–591. AAAI Press, 2007.

[55] S. Goschin, E. Franti, M. Dascalu, and S. Osiceanu. Combine and compare evolutionary robotics and reinforcement learning as methods of designing autonomous robots. *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1511–1516, Sept. 2007.

[56] G. Gottlieb. Probabilistic epigenesis. *Developmental Science*, 10(1):1–11, 2007.

[57] F. Güttler. Realitätsnahe simulationsumgebung einer selbstorganisierenden roboterwelt. Master's thesis, University Leipzig, 2007.

[58] H. Haken. *Synergetics: An Introduction. Nonequilibrium Phase Transition and Self-Organization in Physics, Chemistry, and Biology*. Springer Verlag, 3rd revised and enlarged edition edition, 1983.

[59] H. Haken. *Information and Self-Organization. A Macroscopic Approach to Complex Systems*. Springer Series in Synergetics; Springer eBook Collection. Physics and Astronomy. Springer Berlin Heidelberg, Berlin, Heidelberg, 3. ext. edition edition, 2006.

[60] N. Hamed. *Self-Referential Dynamical Systems and Developmental Robotics*. PhD thesis, University of Leipzig, 2007.

[61] J. M. Herrmann. Dynamical systems for predictive control of autonomous robots. *Theory in Biosciences*, 120:241–252, 2001.

[62] F. Hesse. *Self-Organizing Control for Autonomous Robots*. PhD thesis, University of Göttingen, Institute for Nonlinear Dynamics, 2009.

[63] F. Hesse, G. Martius, R. Der, and J. M. Herrmann. A sensor-based learning algorithm for the self-organization of robot behavior. *Algorithms*, 2(1):398–409, 2009.

[64] M. Hild. *Neurodynamische Module zur Bewegungssteuerung autonomer mobiler Roboter*. PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2007.

[65] X. Huang and J. Weng. Novelty and reinforcement learning in the value system of developmental robots, 2002.

[66] M. Hülse, S. Wischmann, P. Manoonpong, A. von Twickel, and F. Pasemann. Dynamical systems in the sensorimotor loop: On the interrelation between internal and external mechanisms of evolved robot behavior. In Lungarella et al. [84], pages 186–195.

[67] F. Iida, R. Pfeifer, and A. Seyfarth. Ai in locomotion: Challenges and perspectives of underactuated robots. In Lungarella et al. [84], pages 134–143.

[68] H. Iizuka and T. Ikegami. Adaptability and diversity in simulated turntaking behavior. *Artificial Life*, 10:361–378, 2004.

[69] A. J. Ijspeert. A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander. *Biological Cybernetics*, 84(5):331–348, 2001.

[70] A. J. 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.

[71] A. J. Ijspeert, J. Hallam, and D. Willshaw. Evolving Swimming Controllers for a Simulated Lamprey with Inspiration from Neurobiology. *Adaptive Behavior*, 7(2):151–172, 1999.

[72] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87, 1991.

[73] H. Jaeger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots. *Artificial Life and Robotics*, 2:76–79, 1997.

[74] M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992.

[75] J. Kohlmorgen, K.-R. Müller, J. Rittweger, and K. Pawelzik. Analysis of wake/sleep eeg with competing experts. In *ICANN '97: Proc. of the 7th Intl. Conf. on Artificial Neural Networks*, pages 1077–1082, London, UK, 1997. Springer-Verlag.

[76] T. Kohonen. *Self-organization and associative memory: 3rd edition*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[77] C. Kolodziejski, B. Porr, M. Tamosiunaite, and F. Wörgötter. On the asymptotic equivalence between differential hebbian and temporal difference learning using a local third factor. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Proc. NIPS 08*, pages 857–864. MIT Press, 2008.

[78] M. Komosinski and S. Ulatowski. Framsticks: towards a simulation of a nature-like world, creatures and evolution. In D. Floreano, J.-D. Nicoud, and F. Mondada, editors, *Advances in Artificial Life.*, volume 1674 of *LNAI*, pages 261–265. Springer-Verlag, 1999.

[79] V. Konda and J. Tsitsiklis. Actor-critic algorithms, 2001.

[80] Y. Kuniyoshi, Y. Yorozu, Y. Ohmura, K. Terada, T. Otani, A. Nagakubo, and T. Yamamoto. From humanoid embodiment to theory of mind. In *Embodied Artificial Intelligence*, pages 202–218, Berlin, Heidelberg, New York, 2003. Springer.

[81] A. M. Leslie Pack Kaelbling, Michael Littman. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[82] A. Levina, J. M. Herrmann, and T. Geisel. Dynamical synapses causing self-organized criticality in neural networks. *Nature Physics*, 3:857–860, 2007.

[83] R. T. Liu, S. S. Liaw, and P. K. Maini. Two-stage turing model for generating pigment patterns on the leopard and the jaguar. *Physical Review E*, 74(1):011914, 2006.

[84] M. Lungarella, F. Iida, J. C. Bongard, and R. Pfeifer, editors. *50 Years of Artificial Intelligence*, volume 4850 of *LNCS*. Springer, 2007.

[85] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental robotics: a survey. *Connect. Sci.*, 15(4):151–190, 2003.

[86] M. Lungarella and O. Sporns. Mapping information flow in sensorimotor networks. *PLoS Comput Biol*, 2(10):e144, 10 2006.

[87] P. Manoonpong, T. Geng, and F. Wörgötter. Exploring the dynamic walking range of the biped robot "runbot" with an active upper-body component. In *IEEE Conf. on Humanoid Robots*, pages E–publication, Genova, 2006. (Humanoids 2006).

[88] J. Marshall, D. Blank, and L. Meeden. An emergent framework for self-motivation in developmental robotics. In *Proc. 3rd Intl. Conf. on Development and Learning (ICDL 2004)*, pages 104–111, La Jolla, CA, 2004. Salk Institute for Biological Studies.

[89] G. Martius, K. Fiedler, and J. M. Herrmann. Structure from Behavior in Autonomous Agents. In *Proc. IEEE Intl. Conf. Intelligent Robots and Systems (IROS 2008)*, pages 858 – 862, 2008.

[90] G. Martius, J. M. Herrmann, and R. Der. Guided self-organisation for autonomous robot development. In A. e Costa and Francesco, editors, *Proc. Advances in Artificial Life, 9th European Conf. (ECAL 2007)*, volume 4648 of *LNCS*, pages 766–775. Springer, 2007.

[91] G. Martius, F. Hesse, F. Güttler, and R. Der. LpzRobots: A free and powerful robot simulator. http://robot.informatik.uni-leipzig.de/software, 2009.

[92] G. Martius, S. Nolfi, and J. M. Herrmann. Emergence of interaction among adaptive agents. In M. Asada, J. C. T. Hallam, J.-A. Meyer, and J. Tani, editors, *Proc. From Animals to Animats 10 (SAB 2008)*, volume 5040 of *LNCS*, pages 457–466. Springer, 2008.

[93] M. G. Mazzapioda and S. Nolfi. Synchronization and gait adaptation in evolving hexapod robots. In S. Nolfi, G. Baldassarre, R. Calabretta, J. C. T. Hallam, D. Marocco, J.-A. Meyer, O. Miglino, and D. Parisi, editors, *From Animals to Animats 9, SAB 2006, Rome, Italy, September 25-29, 2006, Proc.*, volume 4095 of *LNCS*. Springer, 2006.

[94] D. McFarland and T. Bösser. *Intelligent behaviour in animals and robots*. MIT Press, Cambridge, MA, USA, 1993.

[95] T. McGeer. Passive dynamic walking. *Int. Journal of Robotics Research*, 9(2):62–82, 1990.

[96] C. Meunier and A. D. Verga. Noise and bifurcations. *Journal of Statistical Physics*, 50(1):345–375, 1988.

[97] M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7, 1993.

[98] MobileRobots Inc. http://www.mobilerobots.com, 2008.

[99] R. Murray-Smith and T. A. Johansen, editors. *Multiple Model Approaches to Modelling and Control*. Taylor and Francis, London, 1997.

[100] M. E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46:323, 2005.

[101] N. J. Nilsson. The physical symbol system hypothesis: Status and prospects. In Lungarella et al. [84], pages 9–17.

[102] S. Nolfi. Evolutionary robotics: Looking forward. *Connection Science*, 4:223–225, 2004.

[103] S. Nolfi and D. Floreano. Learning and evolution. *Auton. Robots*, 7(1):89–113, 1999.

[104] S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press, Cambridge, MA, 2001. 2001 (2nd print), 2000 (1st print).

[105] S. Nolfi, D. Parisi, and J. L. Elman. Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28, 1994.

[106] S. J. Nowlan and G. E. Hinton. Evaluation of adaptive mixtures of competing experts. In *NIPS-3: Proc. of the 1990 conference on Advances in neural information processing systems 3*, pages 774–780, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[107] O. J. O'Loan and M. R. Evans. Alternating steady state in one-dimensional flocking, 1998.

[108] Open Source Community. OpenSceneGraph – open source high performance 3d graphics toolkit. http://www.openscenegraph.org, 2008.

[109] Open Source Community. Player/Stage/Gazebo – free software tools for robot and sensor applications. http://playerstage.sourceforge.net, 2008.

[110] P.-Y. Oudeyer, F. Kaplan, and V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(6), 2007.

[111] P.-Y. Oudeyer, F. Kaplan, V. V. Hafner, and A. Whyte. The playground experiment: Task-independent development of a curious robot. In D. Bank and L. Meeden, editors, *Proc. of the AAAI Spring Symposium on Developmental Robotics, 2005*, pages 42–47, Stanford, California, 2005.

[112] F. Pasemann. Discrete dynamics of two neuron networks. *Open Systems & Information Dynamics*, 2:49–66, 1993.

[113] F. Pasemann. Dynamics of a single model neuron. *Intl. Journal of Bifurcation and Chaos*, 2:271–278, 1993.

[114] F. Pasemann, U. Steinmetz, M. Hülse, and B. Lara. Robot control and the evolution of modular neurodynamics. *Theory in Biosciences*, 120:311–326, 2001.

[115] C. Paul. Morphology and computation. In *Proc. Int. Conf. on Simulation of Adaptive Behavior*, pages 33–38. MIT Press, 2004.

[116] I. P. Pavlov. *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex.* Oxford University Press, London, 1927.

[117] K. Pawelzik, J. Kohlmorgen, and K.-R. Müller. Annealed competition of experts for a segmentation and classification of switching dynamics. *Neural Comput.*, 8(2):340–356, 1996.

[118] K. Pearson and J. Gordon. Spinal reflexes. In E. Kandel, J. H. Schwartz, and T. M. Jessell, editors, *Principles of Neural Science*, pages 713–736. McGraw-Hill, New York, 4 edition, 2000.

[119] J. Peters and S. Schaal. Natural Actor-Critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[120] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proc. Third IEEE-RAS Intl. Conf. on Humanoid Robots (Humanoids 2003)*, 2003.

[121] J. Peters, S. Vijayakumar, and S. Schaal. Natural Actor-Critic. In *Proc. 16th European Conf. on Machine Learning (ECML 2005)*, pages 280–291. springer, 2005.

[122] R. Pfeifer and J. C. Bongard. *How the Body Shapes the Way We Think: A New View of Intelligence.* MIT Press, Cambridge, MA, November 2006.

[123] R. Pfeifer, M. Lungarella, O. Sporns, and Y. Kuniyoshi. On the information theoretic implications of embodiment – principles and methods. In Lungarella et al. [84], pages 76–86.

[124] R. Pfeifer and C. Scheier. *Understanding intelligence.* MIT Press, Boston, 1999.

[125] A. Pitti, M. Lungarella, and Y. Kuniyoshi. Exploration of natural dynamics through resonance and chaos. In T. Arai, R. Pfeifer, T. R. Balch, and H. Yokoi, editors, *IAS*, pages 558–565. IOS Press, 2006.

[126] D. Polani. Foundations and formalizations of self-organization. In M. Prokopenko, editor, *Advances in Applied Self-organizing Systems*, pages 19–37. Springer, 2008.

[127] D. Polani, O. Sporns, and M. Lungarella. How information and embodiment shape intelligent information processing. In Lungarella et al. [84], pages 99–111.

[128] B. Porr and F. Wörgötter. Isotropic sequence order learning. *Neural Comput.*, 15(4):831–864, 2003.

[129] M. Prokopenko. Design vs self-organization. In M. Prokopenko, editor, *Advances in Applied Self-organizing Systems*, pages 3–17. Springer, 2008.

[130] M. Prokopenko, N. Ay, G. Baldassarre, F. Boschetti, M. Bruenig, M. Burtsev, R. Der, J. Lizier, S. Nolfi, O. Obst, D. Polani, I. Tanev, L. Yaeger, and A. Zomaya. The First Intl. Workshop on Guided Self-Organisation (GSO 2008). http://www.prokopenko.net/gso.html, November 2008.

[131] S. Renals and R. Rohwer. A study of network dynamics. *Journal of Statistical Physics*, 58:825–848, 1990.

[132] C. W. Reynolds. flocks, herds, and schools: A distributed behavioral model. In *Proc. Computer Graphics (SIGGRAPH '87)*, number 4 in 21, pages 25–34, 1987.

[133] F. Rieke, D. Warland, R. de Ruyter van Steveninck, and W. Bialek. *Spikes: Exploring the Neural Code*. MIT Press, Cambridge, MA, 1st edition, 1997.

[134] J. Rittscher, A. Blake, A. Hoogs, and G. Stein. Mathematical modelling of animate and intentional motion. *Phil. Trans. R. Soc. Lond. B*, pages 475–490, 2003.

[135] Robotis Ltd. Robotis web page. http://www.robotis.com, 2009.

[136] A. Rodriguez. *Guided Self-Organizing Particle Systems for Basic Problem Solving*. PhD thesis, University of Maryland (College Park, Md., USA), 2007.

[137] K. Rose, E. Gurewitz, and G. C. Fox. Statistical mechanics and phase transitions in clustering. *Physical Review Letters*, 65(8):945–948, Aug 1990.

[138] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report 166, Engineering Department, Cambridge University, 1994.

[139] A. L. Samuel. Some studies in machine learning using the game of checkers. In H. Billing, editor, *Lernende Automaten*, pages 155–178. Oldenbourg, München, 1961.

[140] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 6:233–242, 1999.

[141] S. Schaal. *Learning robot control*, pages 983–987. MIT Press, 2 edition, 2002.

[142] S. Schaal, A. Ijspeert, and A. Billard. *Computational approaches to motor learning by imitation*, volume 1431, pages 199–218. oxford university press, 2004.

[143] J. Schmidhuber. Curious model-building control systems. In *In Proc. Intl. Joint Conf. on Neural Networks, Singapore*, pages 1458–1463. IEEE, 1991.

[144] W. Schultz, P. Dayan, and P. R. Montague. A Neural Substrate of Prediction and Reward. *Science*, 275(5306):1593–1599, 1997.

[145] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27, 1948.

[146] K. Sims. Evolving 3d morphology and behavior by competition. *Artificial Life*, 1(4):353–372, 1994.

[147] S. Singh, R. S. Sutton, and P. Kaelbling. Reinforcement learning with replacing eligibility traces. In *Machine Learning*, pages 123–158, 1996.

[148] B. V. Smith. Xfig 2-d cad & drawing program for the X windows system. http://www.xfig.org, 2008.

[149] R. Smith. Open Dynamics Engine – open source, high performance library for simulating rigid body dynamics. http://ode.org, 2008.

[150] O. Sporns and M. Lungarella. Evolving coordinated behavior by maximizing information structure. In L. M. Rocha, L. S. Yaeger, M. A. Bedau, D. Floreano, R. L. Goldstone, and A. Vespignani, editors, *Proc. Artificial Life X*, pages 323–329. Intl. Society for Artificial Life, MIT Press (Bradford Books), August 2006.

[151] L. Steels. Evolving grounded communication for robots. *Trends in Cognitive Science*, 7(7):308–312, July 2003.

[152] L. P. Steffe. The learning paradox: A plausible counterexample. In L. P. Steffe, editor, *Epistomological foundations of mathematical experience*, pages 26–44, New York, 1991. Springer.

[153] S. Steingrube, M. Timme, F. Woergoetter, and P. Manoonpong. Self-organized adaptation of simple neural circuits enables complex robot behavior. under review.

[154] D. Sternad and S. Schaal. Segmentation of endpoint trajectories does not imply segmented control. *Experimental Brain Research*, 1:118–136, 1999.

[155] S. Stitt and Y. F. Zheng. Distal learning applied to biped robots. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 137–142. IEEE Computer Society, 1994.

[156] P. Stone and R. S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proc. of the 18th Intl. Conf. on Machine Learning*, pages 537–544. Morgan Kaufmann, San Francisco, CA, 2001.

[157] S. Strogatz. *Nonlinear Dynamics and Chaos*. Addison Wesley, New York, 1994.

[158] R. S. Sutton. Learning to predict by the methods of temporal differences. In *Machine Learning*, pages 9–44. Springer, 1988.

[159] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, volume 8, pages 1038–1044, 1996.

[160] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, March 1998.

[161] R. S. Sutton and A. G. Barton. Reinforcement learning: Past, present and future. In *SEAL*, pages 195–197, 1998.

[162] J. Tani and M. Ito. Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 33(4):481–488, 2003.

[163] G. Tesauro. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[164] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. MIT Press, 2005.

[165] A. M. Turing. Computing machinery and intelligence. *Mind*, 49:433–460, 1950.

[166] A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc.*, 237:37, 1952.

[167] F. Varela, E. T. Thompson, and E. Rosch. *The Embodied Mind*. MIT Press, 1991.

[168] E. von Glasersfeld. Scheme theory as a key to the learning paradox. In A. Tryphon and J. Vonèche, editors, *Working with Piaget: Essays in honour of Bärbel Inhelder.*, pages 139–146. Psychology Press, London, 2001.

[169] C. J. Watkins and P. Dayan. Q-learning. In *Machine Learning*, volume 8, pages 279–292, 1992.

[170] P. J. Werbos. Backpropagation: Past and future. In *Processing of the Intl. Conf. on Neural Networks*, volume I, pages 343–353, New York, July 1988. IEEE Press.

[171] R. W. White. Motivation reconsidered: The concept of competence. *Psychological Review*, 66:297–333, 1959.

[172] Wikipedia. Aibo – wikipedia, the free encyclopedia, 2009. [Online; 3-June-2009].

[173] Wikipedia. Asimo – wikipedia, the free encyclopedia, 2009. [Online; 29-June-2009].

[174] Wikipedia. Self-organization – wikipedia, the free encyclopedia, 2009. [Online; 28-May-2009].

[175] S. Wischmann, K. Stamm, and F. Wörgötter. Embodied evolution and learning: The neglected timing of maturation. In *Advances in Artificial Life: 9th European Conf. on Artificial Life*, LNAI, pages 284–293. Springer-Verlag, 2007.

[176] F. Woergoetter and B. Porr. Reinforcement learning. *Scholarpedia*, 3(3):1448, 2008.

[177] D. M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11:1317–1329, 1998.

[178] D. M. Wolpert, R. C. Miall, and M. Kawato. Internal models in the cerebellum. *Trends in Cognitive Sciences*, 2:338 – 347, 1998.

Acknowledgments

Many people have supported me during my PhD time and have contributed directly or indirectly to this thesis, who I like to thank here.

I would like to thank Theo Geisel for giving me the possibility to work at this great institute and for the constant support of my work. Thank you for investing so much effort to create and retain such an excellent environment and working atmosphere in the group. I feel honored to be part of it.

I am grateful to Ralf Der for bringing me to the fascinating topic of robotics, for supervising my work, and for teaching me so much about what scientific work is all about. Your inexhaustible optimism made it always a pleasure to work with you and it was very helpful in times of trouble, no matter of which nature. Thanks for the guidance through my thesis work.

I would like to thank J. Michael Herrmann for supervising my work, for raising many fascinating questions (much more than one can solve in single thesis) and advising me to answer some of them. It was great to have an interactive library present next door and to have the 'Herrmannscher Textverbesserungsalgorithmus' available (though mostly non-terminating). Thank you for having time whenever I needed it and also for the countless funny conversations on scientific and everyday life issues.

I would like to thank Frank Hesse for the good collaboration, for helpful conversations and for sharing many troubles with the robot simulations.

My special thanks goes to my wife Anna for giving my life a new meaning, for unlimited support, for offering advice on scientific as well as non-scientific questions, and for giving me the opportunity to challenge my poor language skills with the Russian language. I thank my family for the love and support they gave me, for believing in me all the time and for assisting me in many aspects of my life. At this point I would also like to thank my best friends Alex, Patrick, and Jens in Leipzig for their support and fun.

Many thanks to Min, Wei, Pinar and Manamohan, with whom I shared the office, had many nice conversations, and got an idea of Chinese, Indian, and Turkish mentalities and cultures. Due to them my understanding of different accents of English improved immensely. I thank Min for the nice trips to the countryside, the dumplings, and for being a trusty friend.

I would like to thank Yorck who always had time for computer-related issues and supplied me with just about every technical equipment I needed. Thanks also to Denny for keeping the computing equipment at a rigorously high standard.

Thanks to the secretaries, especially Regina and Katharina for shielding me from most of the bureaucratic paperwork. Tobias should not be forgotten here, because he was a priceless aid in the jungle of upcoming GAUSS/GGNB/PTCN regulations.

Thank you Sascha and Dirk for happily engaging me in entertaining conversations. Thanks to Raoul, Anna, Tatjana, Marc, and all those who joined the lunch breaks to the Willi-Mensa. Many thanks also to all members of the legendary Lunch Club™: Peter, Jacob, Micha, Hecke, Wolfgang, Olaf, Tatjana, Min and Anna for deliciously prepared meals and nice discussions. Due to Micha (Mr. Monteforte), Wolfgang and Mick I got the right type of recreation by touring through the beautiful countryside of Göttingen – thanks a lot.

Apart from the above mentioned people I would like to thank Ghazaleh Afshar, Lishma Anand, Jens Arnold, Demian Battaglia, Vitaly Belik, Oliver Bendix, Dmitry Bibichkov, Armin Biess, Raffael Brune, Kai Bröking, Nikolai Chapochnikov, Vincent David, Ahmed El Hady, Stephan Eule, Katja Fiedler, Ragnar Fleischmann, Tanja Gindele, Carsten Grabow, Babara Guichemer, Harold Gutch, Joachim Haß, Holger Hennig, Moritz Hiller, Sven Jahnke, Hinrich Kielblock, Christoph Kirst, Christoph Kolodziejski, Birgit Kriener, Tomas Kulvicius, Poramate Manoonpong, Irene Markelic, Matthias Mittner, Alejandro Morales Gallardo, Jan Nagler, Neves Fabio Schittler, Lars Reichl, Martin Rohden, Holger Schanz, Benjamin Schwenker, Andreas Sorge, Kristin Stamm, Fabian Theis, Christian Thiemann, Corinna Trautsch, Frank van Bussel, Annette Witt, Steffan Wischmann, Florentin Wörgötter, and Fred Wolf for creating a distinct and wonderful atmosphere.

I am indebted to Michelle Monteforte for correcting my English and to Anna and Frank (Hesse) for proof-reading my thesis. Thanks also to Hecke, Vitaly, and Manja for valuable comments.

I would like to thank Frank Güttler, Frank Hesse, Marcel Kretschmann, and Katja Fiedler for the collaboration on LpzRobots, Harold and Kolo for the shared worries about the website, Hecke for the fun with the poster style, Patrick for tips and tricks in Mathematica, and René Liebscher for the original LaTeX-Makefile.

Thanks to Theo for making the Dynamics Symposium possible every year, which was not only very valuable in terms of scientific understanding and in terms of presentation skills, but it was also great fun, especially on the slopes.

Due to Horst Willburger my stay in Vienna was really a joy – thanks. I would also like to thank Stefano Nolfi for inviting me to Rome and for broadening my view on robotics. Last but not least I thank our neighbors Alex and Ruth for the master's coffee and the Internet at home.

I acknowledge financial support from the Bernstein Center for Computational Neuroscience, BMBF grant 01GQ0432.