

Advanced Embodied Learning

Florian Walter

Technical University of Munich

Department of Informatics

Chair of Robotics, Artificial Intelligence
and Real-Time Systems



TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik

Advanced Embodied Learning

Florian M. J. Walter

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende: Prof. Dr. rer. nat. Stefanie Rinderle-Ma
Prüfer der Dissertation: 1. Prof. Dr.-Ing. habil. Alois Christian Knoll
2. Prof. Dr.-Ing. Walter Weigel

Die Dissertation wurde am 21.07.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 13.10.2021 angenommen.

Florian Walter

Advanced Embodied Learning

Dissertation

Technical University of Munich

Department of Informatics

Chair of Robotics, Artificial Intelligence and Real-Time Systems

Boltzmannstraße 3

85748 Garching bei München

Germany

Information on future research based on the results of this thesis will be made available on the website <http://dissertation.florianwalter.de>.

Cover: The image contains the 3D model “Radar Robot Tin Toy” by *rgrimble*, which is licensed under CC BY and available on *Blend Swap*.

Abstract

The brains of living creatures are the most powerful information processing systems that exist. Their versatility, adaptivity and efficiency are unmatched by any technical system ever built. In addition, the neural circuitry they are made from is one of the most complex structures known to date. Understanding how the brain works has become one of the greatest challenges of our time. The continuous improvement in computing technology, according to Moore’s law, that we have witnessed in the past few decades has enabled a completely new perspective in meeting it. Brain models can now be defined and simulated at unprecedented orders of magnitude and detail. Whereas the focus in neuroscience is on the replication of biological ground truth at a high level of fidelity, neural networks in artificial intelligence are optimized for computational power and task performance.

However, modeling the brain’s embedding in its body and in its environment has been given much less attention, even though *embodiment* has been widely recognized as a central factor in learning and development. This is mainly because, to date, experiments in this direction have been limited to simple conceptual studies or have required complex setups with physical robots that suffer from long setup times and have limited flexibility. With the development of *virtual neurorobotics* as a new field of research which lies at the intersection of robotics, neuroscience and artificial intelligence, we are on the precipice of a fundamental change. Virtual neurorobotics draws from advances in computing and simulation technology to provide both the theory and the tools for embedding simulated brains into simulated robots with a closed loop of perception, cognition and action.

This work introduces a novel framework for embodied learning that is built on the paradigm of virtual neurorobotics. At its core is a brain-derived modular-hierarchical neural network architecture which combines the feature hierarchies learned by deep artificial neural networks with the modular organization of biological brains. Its design and the training of neural networks based on it are enabled by four main contributions along the development process of virtual neurorobotics experiments. To begin with, we present two novel extensions of the *Neurorobotics Platform*, a cloud-based simulation framework for virtual neurorobotics. They enable massively parallel neurorobotics experiments and add support for state-of-the-art neuromorphic processors. To provide a link between virtual and physical neurorobotics, we present the *TUM Robot Mouse*, a 3D printed biomimetic mouse robot with an actuated spine. Secondly, we identify organizational principles in the brain that guide the development of the modular-hierarchical neural network architecture. It is comprised of component networks that process input from individual sensory modalities and hub networks that perform task-based data fusion of the component network output. Thirdly, we propose a self-supervised training procedure with a novel topographic loss function that enables the learning of common latent space representations between component networks. Lastly, we introduce the concept of training protocols that mimic biological development processes to optimize the training of modular-hierarchical neural networks. A training protocol controls the temporal sequence of learning in component and hub networks by defining a schedule with developmental steps and stages.

The results of this work lay the foundations for new learning methods that actively exploit robotic embodiment. From a practical perspective, the modularity of the proposed architecture will enable better reuse of trained networks across applications. As different sub-functions can be directly located in concrete modules, the integrated networks offer many opportunities for increasing robustness and interpretability.

Zusammenfassung

Die Gehirne von Lebewesen sind die leistungsfähigsten Informationsverarbeitungssysteme, die es gibt. Ihre Vielseitigkeit, Anpassungsfähigkeit und Effizienz werden von keinem je gebauten technischen System erreicht. Gleichzeitig zählen die neuronalen Schaltkreise, aus denen sie aufgebaut sind, zu den komplexesten Strukturen, die bislang bekannt sind. Zu verstehen, wie das Gehirn funktioniert, hat sich zu einer der größten Herausforderungen unserer Zeit entwickelt. Mit der stetigen Weiterentwicklung der Rechnertechnik in den vergangenen Jahrzehnten gemäß dem Mooreschen Gesetz hat sich eine grundlegend neue Perspektive eröffnet, da Gehirnmodelle in nie dagewesenem Umfang und Detailreichtum modelliert und simuliert werden können. Während der Fokus in der Neurowissenschaft auf der detailgetreuen Nachbildung der Biologie liegt, werden neuronale Netze im Bereich der Künstlichen Intelligenz im Hinblick auf Rechenleistung und die Lösung konkreter Aufgabenstellungen optimiert.

Deutlich weniger Beachtung hat bislang die Modellierung der Einbettung des Gehirns in seinen Körper und seine Umgebung gefunden, obwohl das Konzept von *Embodiment* weithin als ein wesentlicher Faktor für Lernen und individuelle Entwicklung anerkannt ist. Das ist vor allem darin begründet, dass experimentelle Untersuchungen zu diesem Thema bislang auf sehr einfache konzeptionelle Studien beschränkt waren oder komplexe Versuchsaufbauten mit physischen Robotern erforderten, die sich nur unter hohem Zeitaufwand einrichten lassen und unflexibel sind. Diese Situation ändert sich mit der Entstehung der *virtuellen Neurorobotik* als neues Forschungsgebiet an der Schnittstelle von Robotik, Neurowissenschaft und Künstlicher Intelligenz gerade grundlegend. Den Ausgangspunkt hierfür bilden Fortschritte in Rechner- und Simulationstechnik, die in der virtuellen Neurorobotik die Entwicklung neuer Theorien und Werkzeuge für die Einbettung simulierter Gehirne in simulierte Roboter ermöglichen. Beide Simulationen interagieren dabei in einem geschlossenen Regelkreis aus Perzeption, Kognition und Aktion mit ihrer Umwelt.

In dieser Arbeit wird ein neuartiges Framework für Embodied Learning, also das Lernen eines in einen Körper eingebetteten Systems, vorgestellt. Es baut auf dem Konzept der virtuellen Neurorobotik auf, wobei der Kernbestandteil eine vom Gehirn abgeleitete modular-hierarchische Architektur für neuronale Netze ist, die von tiefen neuronalen Netzen gelernte Merkmals-hierarchien mit den modularen Organisationsprinzipien biologischer Gehirne vereint. Die Konzeptionierung dieser Architektur und das Training von neuronalen Netzen, die auf ihr basieren, werden durch vier wesentliche Beiträge entlang des Entwicklungsprozesses für virtuelle Neurorobotik-Experimente ermöglicht. Erstens werden zwei Erweiterungen für die *Neurorobotik-Plattform* eingeführt, einer Cloud-basierten Simulationsumgebung für die virtuelle Neurorobotik. Diese ermöglichen zum einen hochgradig parallele Neurorobotik-Experimente und ergänzen zum anderen die Unterstützung für aktuelle neuromorphe Prozessoren. Als Bindeglied zwischen virtueller und physischer Neurorobotik wird die *TUM Robotermaus* vorgestellt, ein 3D-gedruckter biomimetischer Mausroboter mit aktiver Wirbelsäule. Zweitens werden Organisationsprinzipien im Gehirn identifiziert, die die Grundlage für die Entwicklung der modular-

hierarchischen Netzarchitektur bilden. Diese besteht aus Komponentennetzen, die die Eingaben sensorischer Modalitäten verarbeiten, und Hub-Netzen, die deren Ausgaben aufgabenspezifisch fusionieren. Drittens wird ein selbstüberwachtes Trainingsverfahren mit einer neuartigen topographischen Fehlerfunktion vorgestellt, die das Lernen gemeinsamer latenter Datenrepräsentationen für die Komponentennetze ermöglicht. Zuletzt wird das Konzept von Trainingsprotokollen eingeführt, die biologische Entwicklungsprozesse nachahmen, um das Training von modular-hierarchischen neuronalen Netzen zu optimieren. Ein Trainingsprotokoll steuert die zeitliche Abfolge des Lernens in den Komponentennetzen und Hubnetzen, indem es einen Zeitplan mit Entwicklungsschritten und -stufen festlegt.

Die Ergebnisse dieser Arbeit legen die Grundlage für neue Lernverfahren, die Embodiment in einem Roboter aktiv nutzen. Aus praktischer Sicht relevant ist, dass die vorgestellte Netzarchitektur eine bessere Wiederverwendbarkeit von Teilmodulen in verschiedenen Anwendungen ermöglicht. Da einzelne Funktionen in konkreten Modulen lokalisiert sind, bieten die resultierenden Gesamtnetze eine Vielzahl von Möglichkeiten zur Umsetzung höherer Robustheit und besserer Interpretierbarkeit.

Logic will get you from A to B.

Imagination will take you everywhere.

Albert Einstein

Acknowledgements

There is probably no other statement than the quote from Albert Einstein on the previous page that could better describe the work on this dissertation from the first idea to the final version. Addressing the question of how artificial intelligence, machine learning and robotics can benefit from brain research opens up an exploration space of intimidating dimensionality. How can an endeavor that has been attempted countless times before filled with new life? Logic is a prerequisite for scientific progress. But only imagination can help to find and connect the right pieces and threads from each of the two worlds to create something new that lives in both of them.

Imagination needs a foundation to thrive. This work would never have been possible without my advisor Prof. Alois Knoll, who introduced me to the fascinating topic of neurorobotics. I am deeply grateful for his confidence in me. His ideas, encouragement and critical feedback were essential for starting, pushing forward and finalizing my research for this thesis. His openness and motivation make the past years mean much more to me than what is written on these pages.

Imagination needs reality to have a meaning. I also want to thank Prof. Walter Weigel for agreeing to be the second examiner for my dissertation. With his professional background, he provides the much needed link between basic research and industry.

Imagination needs inspiration to evolve. I am extremely grateful to Prof. Christoph von der Malsburg for being my mentor. His deep involvement in the topic and his openness to new ideas made every of our discussions a source of inspiration with lasting impact. Thanks to his feedback I deviated from my path to make new discoveries.

Imagination needs a place to stay. The unique environment at the Chair of Robotics, Artificial Intelligence and Real-Time Systems made the work on my dissertation so much more enjoyable. I want to thank Dr. Alexander Lenz for his support in all administrative matters from project proposals to teaching. Many thanks also to Amy Bücherl and Ute Lomp for doing so much more than they had to.

Imagination needs to spread to become real. I also want to thank Jun-Heui Cho, Tobias Jülg, Pierre Krack, Thomas Krieger, Peer Lucas, Kok Choong Ng and Eva Siehmann for contributing to the implementation of my ideas with their Bachelor's theses, Master's theses and other projects.

Imagination needs a home to come alive and persist. I want to thank my parents Friedrich and Dietlinde for supporting me in all my endeavors. In every possible way. From the beginning to the end. But never ending. Knowing that there is a home makes every journey possible. Anywhere. Anytime.

Contents

1	Introduction	1
1.1	Cognitive Systems – The Challenge Ahead	3
1.2	Three Waves of Artificial Intelligence	6
1.3	The Case for Neurorobotics and Brain-Derived AI	11
1.4	Scope and Contribution of this Work	16
2	Models of the Brain	21
2.1	Levels of Detail in Brain Modeling	22
2.2	Common Roots of Brain Simulation and Artificial Neural Networks	31
2.3	From Perceptrons to Deep Neural Networks	36
2.4	From Spiking Neurons to Large-Scale Brain Simulations	45
3	Foundations of Neurorobotics	55
3.1	Theoretical Foundations of Embodied Systems	56
3.2	A Modern Definition of Neurorobotics	65
3.3	The HBP Neurorobotics Platform	74
4	A New Tool Set for Neurorobotics Experiments	81
4.1	Massively Parallel Neurorobotics Experiments	82
4.2	Neuromorphic Neurorobotics Experiments with Intel Loihi	97
4.3	Biomimetic Neurorobotics with the TUM Robot Mouse	101
5	A Brain-Derived Modular-Hierarchical Neural Network Architecture	109
5.1	Functional Specialization and Modularity in the Brain	110
5.2	Design Principles of Modular-Hierarchical Neural Networks	117
5.3	Self-Supervised Learning of Deep Multisensory Neural Maps	133
6	Developmental Embodied Learning	161
6.1	Modeling Biological Development with Training Protocols	162
6.2	Neuromorphic Data Fusion	167
6.3	Staged Reinforcement Learning	175

7	Conclusion and Outlook	197
7.1	Main Contributions of this Work	197
7.2	Practical Relevance	199
7.3	Future Directions of Research	202
A	Parameters of Models and Algorithms	205
A.1	Deep Multisensory Neural Maps	205
A.2	Neuromorphic Data Fusion	209
A.3	Staged Reinforcement Learning	210
	List of Acronyms	211
	List of Figures	213
	List of Tables	217
	List of Algorithms	219
	Bibliography	221

1

Introduction

It's all mechanistic. We are mechanism. If we are machines, then in principle at least, we should be able to build machines out of other stuff, which are just as alive as we are.

Rodney Brooks, TED2003 [1]

What does it take to construct a machine that performs like a living creature? A system that is autonomous and efficient, that senses and acts in real-time while it learns and adapts? Does it require a bionic mechanical structure that looks and works just like our body? Will it depend on a specific type of circuitry that is wired up like our brain? Or is it just a matter of enough data and computational power?

Over the past centuries, humankind has come up with countless visions, theories and plans of how to copy or imitate the feats of nature in artificial systems, ranging from the harnessing of divine power in Homer's Iliad [2] to the foundation of robotics as a new scientific discipline in the 20th century. What is common to all of them is that they take some form of inspiration from nature, be it the design of the body or be it the architecture of the control system. Yet no robot has ever been built that is anywhere near as versatile, efficient and flexible as its biological model. Currently, however, this seems about to change: Compute power and storage have recently become affordable enough to train large-scale *deep neural networks (DNNs)* on massively big data sets that outperform many traditional methods from artificial intelligence by orders of magnitude. While it is true that these networks share some very basic features with biological brains, they still operate

in a fundamentally different way. As DNNs are becoming more and more powerful, the demand for training data and compute power is growing prohibitively large – just to train a network that can only solve a single task [3]. What is it that makes biological brains so much more versatile and efficient? Will the performance of artificial neural networks (ANNs) automatically improve if they are made more similar to the neural circuitry of the brain?

The success story of ANNs started with the adoption of only a few elementary organizational principles from biological neural networks. Similarly, an aircraft is not a full copy of a bird's body but the technical implementation of a small set of common design principles. The challenge therefore lies in the identification of the constituent computational and organizational principles of the brain that are missing in current ANN models. So far, systematic research in this direction has been hindered by technical limitations: While neuroscientists have collected tremendous amounts of data about the brain over the last century, the knowledge gained from these individual studies is fragmentary and there are no tools for integrating it into a common framework. At the same time, measurement data are static snapshots and therefore offer only limited explanation about the functional dynamics of the brain in response to sensory input and output. Computational models that address this issue need to make assumptions and simplifications that also yield a rather narrow and limited view of the brain. Progress in understanding the brain is consequently becoming more and more limited by the tools available in neuroscience. Deciphering the inner workings of the brain has become one of the biggest challenges in science to date.

This thesis builds on two recent methodological innovations that considerably widen the possibilities for studying the brain. First of all, *simulation neuroscience* [4] establishes a completely new experimental paradigm where measurements are used to synthesize high-fidelity brain simulations that do not attempt to simplify but to indeed reflect all details of the physical world. Secondly, *neurorobotics* [5] provides a novel tool set for embedding these simulations inside robotic embodiments in order to study brain models under realistic conditions as they interact through a body with the environment in a closed loop of *perception – cognition – action*. Like simulation neuroscience, state-of-the-art neurorobotics is based on simulations. This enables fully virtual brain research which is not limited by any physical constraints. Multiple neurorobotics experiments can be designed, instantiated and evaluated in parallel at almost no additional cost. The results obtained from them are fully reproducible and do not depend on specific hardware or certain environmental conditions, which, for the very first time, makes research on brain models that interact with an environment fully reproducible.

The goal of this thesis is to show that it takes neither an exact copy of the body nor a perfect simulation of the brain to build a machine that performs like a living creature. The only requirement is to implement the essential principles that underlie biological cognition. Based on findings from neuroscience and the tool set of neurorobotics, we identify three of them that have been largely neglected until now: *embodiment*, *modularity* and *development*. Each of them has been addressed earlier but they have never been studied in depth together. This is only now possible with the new tool set provided by neurorobotics and in particular the Neurorobotics Platform (NRP) [6] that has been

developed in the European Human Brain Project (HBP) [7]. By combining embodiment, modularity and development in a novel modular neural network architecture, we will show how the structure of the brain can be linked to the cognitive functions it implements and that this link can only be established inside a body that is able to interact with its environment. The work presented in this thesis lays the foundations for cognitive systems that are not only brain-inspired but *brain-derived* since their design is directly derived from computational principles that are known to be implemented in the brain rather than just taking inspiration from them.

1.1 Cognitive Systems – The Challenge Ahead

The brains of living creatures are the most powerful and most versatile control systems that exist. They enable survival under hostile conditions in ever-changing environments, are capable of processing incomplete and noisy sensory input, can reason about both concrete situations and abstract ideas, look ahead, plan the next steps and control the body that connects them with the outside world. At the same time, they continuously learn from experience and adapt to changes on the fly. In short: biological brains are capable of *cognition* and the living creatures they belong to are *cognitive systems*. More formally, cognition can be defined as follows [8]:

“Cognition is the process by which an autonomous system perceives its environment, learns from experience, anticipates the outcome of events, acts to pursue goals, and adapts to changing circumstances.”

The meaning of cognition is often reduced to *thinking*, which closely adheres to the direct translation *act of comprehension* or *investigation* of the original Latin word *cognitio* [9]. Following the definition above, this work adopts a broader view where cognition encompasses all mechanisms and processes that enable a living creature or an artificial system to act and interact autonomously in unconstrained dynamic environments to achieve its goals. In this perspective, artificial intelligence (AI) is a branch of research on cognitive systems.

The Robotics Paradox

The challenge of designing a cognitive system is best illustrated by the example of the current state of the art in robotics. Modern robots are widely used in manufacturing where they perform tasks that could previously only be done by skilled human workers after years of training and experience. At the same time, they outperform these human workers in both precision and speed. This would not be possible without processing the data from many different sensors and adapting the action accordingly in real-time. At first glance, this seems very similar to what humans and animals need to do in order to interact with their environment. However, many of the tasks that we carry out with ease and often even without thinking about them are still dauntingly far out of reach for robots. Strikingly,

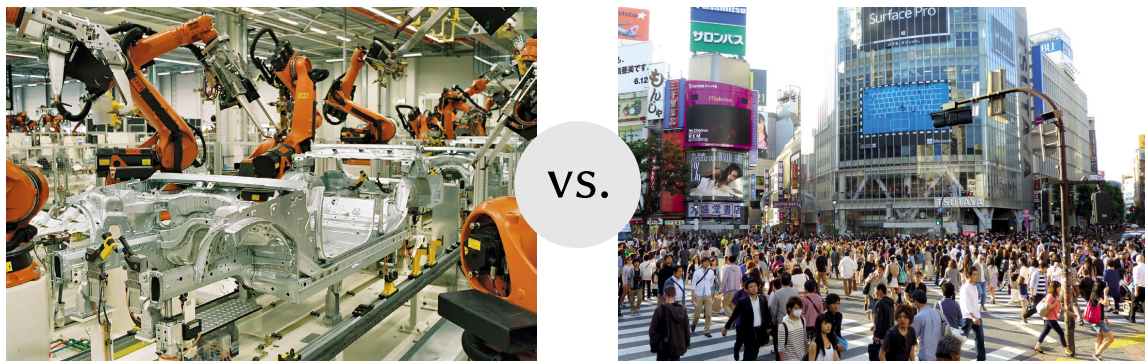


Figure 1.1: Illustration of the *robotics paradox*. While robots outperform humans in tasks that usually require years of training such as welding a car, seemingly simple routine tasks such as driving a car in a crowded urban environment are still unsolved challenges. *Left:* Photo of a body shop in a car production line by BMW Werk Leipzig from Wikimedia Commons licensed under CC BY-SA 2.0 DE. *Right:* Photo of Shibuya Crossing in Tokyo.

this concerns especially simple everyday tasks like walking, handling and manipulating objects, and navigating in cluttered environments, etc. This *robotics paradox* has been pointed out already many times [10, 11, 12] and is illustrated in Figure 1.1. The core statement of the paradox is that well-defined tasks in constrained environments are easy to solve compared to typical everyday tasks. This is because the latter are hard to define formally and usually take place in unconstrained dynamic environments. Traditional modeling and programming therefore largely fail in solving them. How could one ever cover all the variability and uncertainties for a simple task like picking up an item from a store? What can be described with as little as a single sentence in natural language might require completely different solution strategies that depend on seemingly trivial factors like the time of the day or the current weather conditions (Is the store open? Can I go by bike or should I take the bus?). On a smaller scale, even the location and placement of the item in the store can result in completely different types of requirements (Can I get the item myself or do I need to order it? Can I carry it or do I need a shopping cart?).

The robotics paradox exemplifies the challenge that is currently being faced in cognitive systems research. Building a cognitive system is not about solving a task that seems hard to us, but about designing a system that can adapt to arbitrary unconstrained environments. Importantly, the latter does not apply to the numerous landmark achievements in the more recent history of AI. Prominent examples include the victory of the IBM system *Deep Blue* over a world-class chess champion in 1997 [13], the defeat of a world-class Go champion in 2016 by Google DeepMind’s *AlphaGo* [14] and the poker AI *Pluribus* that outperformed professional human Poker players [15]. Similar progress has been made in popular strategy video games with *OpenAI Five* that plays Dota 2 [16] or DeepMind’s *AlphaStar* has reached a Grandmaster level in StarCraft II [17]. IBM’s *Watson* and *Project Debater* have successfully competed against humans in quiz shows and debates, respectively [18, 19]. This list could be continued with an endless number of smaller success stories, in every of which some form of AI was applied to solve a

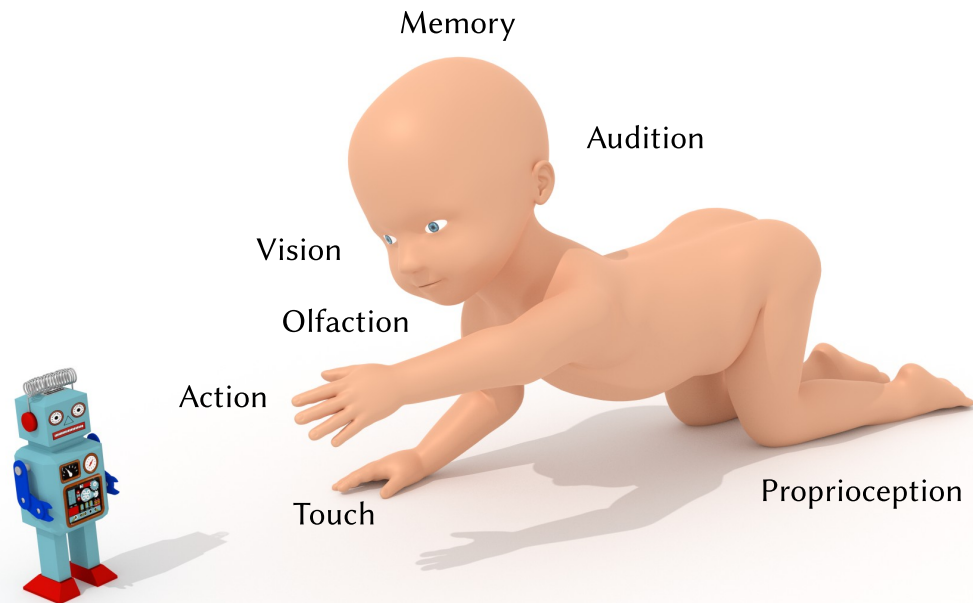


Figure 1.2: A newborn exploring its environment. The brain receives input data from multiple sensory modalities at the same time and integrates them in order to generate appropriate actions. The results of the interactions are *specific to the environment* and become aggregated over time to form memories that in turn modulate how sensory input is interpreted and processed. The body model of the newborn is based on [20], “Radar Robot Tin Toy” by rgrimble from Blend Swap is licensed under CC BY.

problem for the first time or to outperform existing methods. However, just like the major breakthroughs in mastering complex games, they all address narrow problems in well-defined constrained environments. The advantage of these problems is, of course, that they can serve as reproducible benchmarks with clearly defined success criteria. For example, the evolution of chess AIs can be tracked consistently across decades even though algorithms and hardware have dramatically changed. Substantial progress in cognitive systems research can only be made by defining similar tasks that channel research efforts towards a common goal and that make scientific progress traceable.

Learning like a Baby

This work begins with the most basic task faced by newborns of virtually all higher animal species. Babies are born with only little knowledge about their body, the environment it is embedded in, and the laws that govern the interplay between them. They need to *learn* how to survive in the world around them and how to *make sense* of what they *perceive* from it. As illustrated in Figure 1.2, this process of learning and perception crucially depends on the body and its senses. Information about the surroundings is gathered by different sensory modalities such *vision*, *touch* or *audition*. Each of them captures only a small and limited fraction of the ground truth and depending on the current conditions

(day vs. night), one might be more reliable than another one (vision vs. audition). As a result, even though all sensory modalities are stimulated by the same environment, they still produce very different sensory input streams. Perceiving and making sense of the world means combining them into a single coherent representation. Technically, the main task faced by a newborn, therefore, is a problem of *sensor data fusion* or, as it is commonly referred to in neuroscience, *multisensory integration*.

Figure 1.2 already indicates that data fusion is not a passive process solely driven by the outside world. This is because the sensory input is highly dependent on the actions that control the body. Just a slight movement of the head can already entail a considerable change in visual and proprioceptive stimuli that resonates with the information processing in the brain and in turn determines the next action. This intricate reciprocal relationship between perception and action is at the core of what a newborn does when it explores the world and gains experience through playing and interacting. Over time, the sum of all experiences is aggregated in the memory to form the basis for goal-directed purposeful behavior. The exclusive focus of most current research on the resulting behavior without addressing the processes from which it develops can be seen as one of the main reasons why existing artificial cognitive systems only can solve very narrow tasks. They lack appropriate means for processing, fusing and interpreting sensory information from the world around them. Brooks summarizes this insight as follows [21]:

“This suggests that problem solving behavior, language, expert knowledge and application, and reason, are all rather simple once the essence of being and reacting are available. That essence is the ability to move around in a dynamic environment, sensing the surroundings to a degree sufficient to achieve the necessary maintenance of life and reproduction.”

This sets a clear frame for the challenge ahead. While the algorithms that were originally developed for solving games have been successfully applied in many practical applications that are focused on narrow tasks, substantial progress in understanding and replicating the cognitive skills of living creatures will require a different approach. The starting point for this thesis is therefore the simple problem outlined above: *How can a newborn, without any knowledge of the world around it, learn to perceive, act and understand?*

1.2 Three Waves of Artificial Intelligence

The first principled efforts to model and implement the cognitive skills of living creatures date back to 1956, when AI was established as a scientific discipline during the Dartmouth Summer Research Project on Artificial Intelligence [22]. Its primary objective is the development of computer systems with human-like capabilities that can act autonomously in complex environments [23]. This sets the field apart from related disciplines such as *neuroscience* or *cognitive science*, which are concerned with the discovery of the principles underlying cognition, but not with their practical implementation in technical systems. AI is therefore strongly embedded into computer science and its applications are commonly

referred to as *agents*. Since its early beginnings, AI has gone through several waves during which new models and methods were first taken up euphorically after promising initial results before they were abandoned due to unsurmountable difficulties and limitations. Each wave was governed by a prevalent modeling paradigm that set the main direction of research with deep implications for the types of systems that were studied and the way they were implemented.

The First Wave: Symbolic Reasoning

The focus of early AI research after the influential meeting at Dartmouth College was on models of *symbolic reasoning*. A prototypical example is outlined in Figure 1.3. At the core of every symbolic reasoning system is a *knowledge base* that stores facts and rules about the world which are typically specified in a formal language such as logic. Content needs to be provided manually by a programmer through *knowledge engineering* at the appropriate level of abstraction. The actual “intelligence” of the system lies in an *inference engine* that automatically computes answers to input queries based on the rules and facts from the knowledge base. One of the most prominent and most influential symbolic AI systems was the mobile robot *Shakey* that was developed between 1966 and 1972 [24]. It was equipped with a cognitive model that allowed it to navigate, plan and interact in a reduced environment of simple geometric objects. While these restrictions might seem quite narrow, the results of the project were groundbreaking. Some of the methods developed for Shakey such as the *A** algorithm for heuristic graph search [25] or the *STRIPS* planning system [26] set standards in their respective fields that are still valid today. These

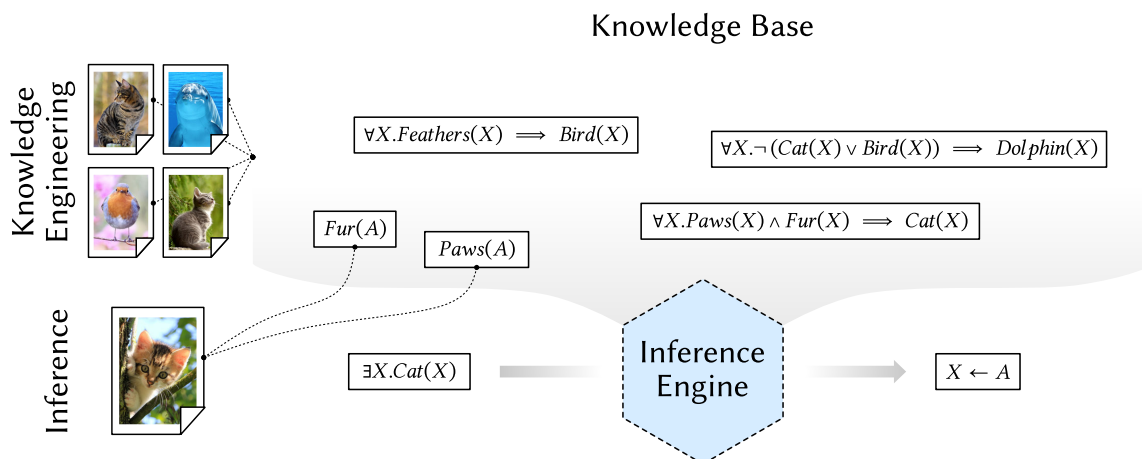


Figure 1.3: Example of a basic symbolic reasoning system. Knowledge is represented in *first-order logic* formulas that are stored in a *knowledge base*. General rules and facts need to be explicitly provided by a human programmer. The process of extracting these abstract rules from the underlying domain is also known as *knowledge engineering*. During runtime, new facts are generated by an *inference engine*. Knowledge representation is not limited to simple first-order logic. Real-world systems typically feature many additions and refinements for more expressive power and better performance.

early successes fueled expectations that technological progress, as predicted by Moore's law, would sooner or later automatically translate into more cognitive power. However, as it turned out, purely symbolic AI suffers from inherent complexity issues that make it impossible to apply it to real-world problems of practically relevant sizes, no matter how much compute power and storage are available. While it has become extremely successful in solving *representation hungry problems* [21] such as planning and reasoning, applications have never moved beyond this very limited domain for mainly two reasons. First of all, symbolic AI systems are based on extremely simplified assumptions about the world, such as determinism, full observability and static time-independent behavior [27]. Secondly, all of the system's knowledge must be provided by the programmer, which turned out to be impossible for real-world problems that have no compact mathematical representation. Especially the second issue becomes immediately evident when trying to provide a formal description of features for recognizing a face, a car or a cat in a picture. This inherent shortcoming of rooting symbolic representations in the real world is also referred to as the *symbol grounding problem*. But even if a system can be described formally, the manual specification of knowledge bases remains a substantial bottleneck.

The Second Wave: Machine Learning

The inherent shortcomings of physical symbol systems (PSSs), as symbolic AI was formally defined by Newell and Simon [28], shifted the focus of research towards new models and methods. The main bottleneck of traditional reasoning systems such as the one in the example from Figure 1.3 is the knowledge base. Defining an adequate knowledge representation does not only require non-trivial problem-dependent abstractions engineered by domain experts. It also involves considerable manual effort and is practically impossible for domains that cannot be described formally. This is why machine learning methods that automatically acquire knowledge from data became increasingly popular and triggered the second wave of AI. Common machine learning systems have an architecture similar to the example in Figure 1.4 that depicts a basic image classifier. Its key component is a *model* M that computes *predictions* for input samples. In the case of image classification, the input samples are images and the output is an assignment to possible classes with confidence levels. While the inference engine in symbolic reasoning systems is a problem-agnostic generic component, M needs to be parameterized specifically for the problem to be solved. This *training process* is typically accomplished by minimizing a loss function that measures the prediction error for samples in the training data set. As soon as an appropriate set of model parameters has been learned, M can be applied to new samples outside the training set. Analogously to symbolic reasoning, the generation of predictions is commonly referred to as *inference*. Successful training not only requires the right choice of M but also a suitable representation of the input data. The first is mainly guided by the concrete task to be solved. Common models for image classification include simple linear basis function models such as polynomials or support vector machines (SVMs) [29]. Although the underlying model fully determines the computational power of the system, the actual challenge lies in choosing an adequate representation of the input

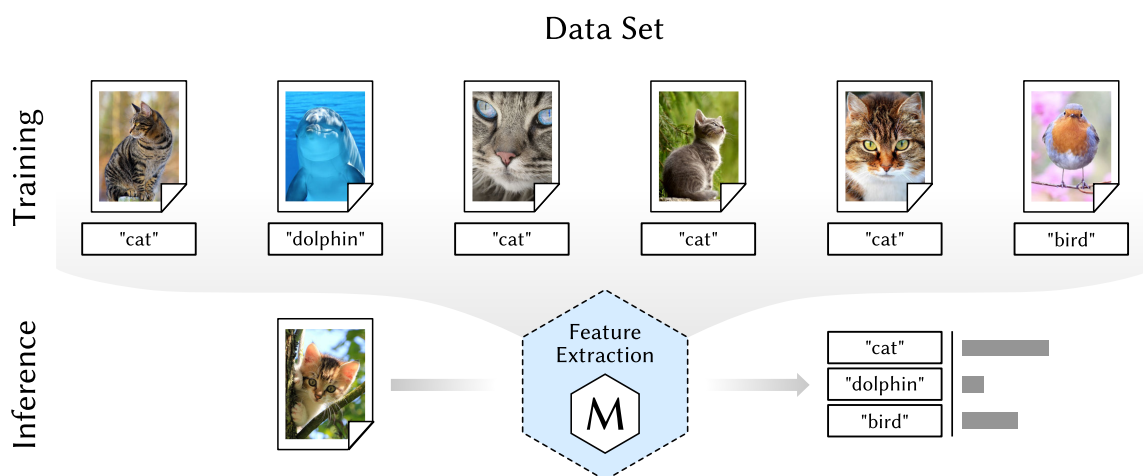


Figure 1.4: Example of a basic machine learning system for image classification. A *model* M is trained with sample images in a *data set*. During training, the parameters of M are adjusted to minimize classification errors based on a loss function that measures how well M fits the data set. Traditionally, the model does not operate on raw data. Instead, every input sample is converted to a set of *features* (e.g. colors, lines, shapes, etc.), extracting only the information that is relevant for the actual learning process. This feature extraction is also applied after the training phase during *inference* when M is used to compute predictions for new samples.

data. The high dimensionality of raw data such as images, videos or speech makes the extraction of features that are relevant for the learning task essential. In particular, these features need to be robust against data transformations that do not change the semantics. Different lighting conditions and viewpoints, for example, have considerable influence on the appearance of an image but should not affect the classification result as long as the actual content remains unchanged. Figure 1.5 illustrates the effects of the *feature space representation* of the input data on the complexity of the machine learning problem. Both graphs in the figure visualize the same 2D data set. On the left, samples are plotted in cartesian coordinates. Every dot corresponds to a sample while the color indicates the class it is assigned to (e.g. cat or bird). In the cartesian representation, the two classes can only be separated by a circle. After transforming the data into polar coordinates, however, the circles break up and the samples can be separated by a simple line. Clearly, identifying relevant features for real-world data is far more complex. Finding and describing appropriate feature space representations therefore has gradually evolved to a new bottleneck since it again depends on domain experts that need to analyze data manually to identify and formalize invariants that capture the underlying semantics without being affected by transformations and noise. Knowledge engineering has thereby been replaced by *feature engineering*. In a sense, machine learning has shifted the abstraction bottleneck of symbolic AI one level lower to defining appropriate abstractions for raw data. This lets the training process and often also the size of M scale with compute power and storage. However, the engineering of suitable features still needs to be done manually for every problem domain and can be very complex, resulting in a new bottleneck.

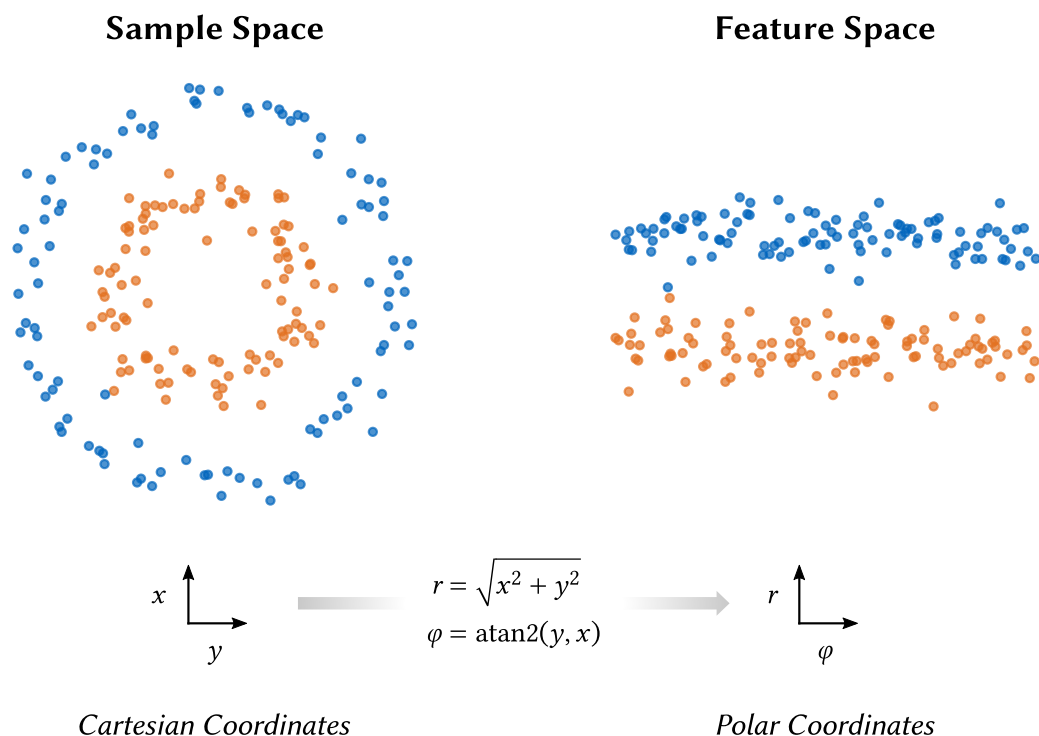


Figure 1.5: Example of a basic feature space transformation. Every colored dot in the two plots corresponds to a data sample in 2D space. The color of a dot indicates the class the corresponding sample is assigned to. In the left plot, all samples are represented in cartesian space and the two classes correspond to circles. After transforming all samples to polar coordinates, the circles become lines. This *feature space representation* is very advantageous as it enables separating the two classes with a simple line.

The Third Wave: Deep Learning

During the last decade, machine learning has rapidly evolved to the most powerful tool in AI to date. This has been only possible after a seminal breakthrough resolved the feature engineering bottleneck and set off an avalanche of developments that finally resulted in the current third wave of AI. The starting point of this development were attempts to solve the issue of feature engineering by learning not only the actual task but also the features themselves. Clearly, *feature learning* or, as it is also called, *representation learning* [30] seems like the next logical step after the transition from symbolic reasoning to learning from examples. The actual breakthrough, however, was not due to a new model or algorithm. Instead, it turned out that *deep* ANNs with many layers of neurons stacked on top of each other are capable of automatically learning powerful hierarchical feature space representations even from complex raw data such as images [31]. Remarkably, the theoretical foundations of ANNs were laid already in the first half of the 19th century, more than a decade before the Dartmouth conference [32]. What had suddenly changed was that for the first time compute power and storage had reached a scale at which networks with a sufficiently large number of layers and neurons could be simulated within reasonable time at affordable cost. This is not to say, of course, that there has been no progress in theory

since the publication of the first ANNs, but rather that the theoretical developments in this field could only be fully capitalized when the necessary computational resources became available. *Deep learning* finally gained unprecedented momentum when a deep convolutional neural network (CNN) outperformed all competitors in the ImageNet Large Scale Visual Recognition Challenge 2012 [33, 34]. The key to training a network with millions of parameters on millions of raw images was to accelerate the training process by offloading computation to GPUs. Today, this approach has become standard in both academia and industry.

The focus of AI research has shifted almost completely to deep learning since it scales exceptionally well with computational resources. Moore's law thereby seems to translate directly to more powerful models and more accurate predictions. However, with growing task complexity the demand for compute power is increasing faster than the performance gains predicted by Moore's law [3]. At the same time, a new bottleneck is already narrowing down possible applications of DNNs. Deep learning has replaced manually engineered features by massively large data sets from which feature space representations are learned automatically. This approach works well for problems for which a sufficiently large body of data is available. Examples include images and videos that can be downloaded together with contextual information from the internet. Data for which this information is not available need to be labeled manually. Unlike feature engineering, this labeling usually does not involve special domain knowledge but instead requires the manual processing of millions of samples, which means that complexity is traded in for quantity. The bottleneck is thereby shifted another level lower from data abstraction to data description and feature engineering is replaced by what could be called *data engineering*. Unlike before, this bottleneck can be resolved by simply spending enough resources and leveraging new collaboration models such as crowdsourcing [35]. Nevertheless, manual data processing considerably narrows down the number of possible applications and again imposes physical boundaries on what can be achieved with AI. These limitations in particular apply to the seemingly simple task of a newborn learning to understand, perceive and act in its environment defined at the end of the last section. The newborn's experience cannot be modeled as a static data set but is a stream of experiences without labels that needs to be processed in real-time at a fraction of the power consumption of a modern GPU.

1.3 The Case for Neurorobotics and Brain-Derived AI

At first glance, the ever-growing demand for more data and compute power of DNNs appears to be mainly a problem of resources and costs. Most of the recent successes mentioned in Section 1.1 would indeed not have been possible without millions of hours of training in data centers. Unlike the engineering of knowledge representations or features, the data bottleneck in deep learning could be considered rather a practical issue than a theoretical limitation. Nevertheless, reducing the open challenges in the field to generating data and accelerating computation would fall too short.

Limitations of Deep Learning

DNNs have proven especially successful in processing images and natural language where they outperform virtually all competing approaches. This makes them in particular also attractive for applications in robotics and autonomous driving. But in contrast to consumer applications such as image classification and speech recognition that have become available on any recent smartphone, deep learning is only very slowly gaining momentum in these fields. This is because of the high safety requirements for components that are deployed on robots and autonomous cars, where any malfunction of a system component may pose a life-threatening danger. Severe accidents with autonomous cars provide appalling evidence for the necessity of failure-free operation [36, 37]. The complexity of DNNs with millions of trainable parameters, however, makes any type of analysis or verification challenging and produces undesired artifacts. Already small perturbations of the input data that are invisible to humans can result in completely different network output [38, 39]. Such *adversarial examples* can be used to attack systems that are based on deep neural networks. These attacks are not limited to direct modifications of the network's input data but can also succeed when prepared images or objects are captured from the physical world [40, 41, 42]. DNNs that implement control policies for a robot simulation have been shown to be vulnerable to adversarial attacks, too [43].

Adversarial attacks on machine learning systems are not specific to deep learning and were discovered earlier for traditional models [44]. What makes the problem so challenging for DNNs is the huge number of trainable parameters and their *end-to-end* architecture. The latter means that the mapping between raw input data and predicted output is fully determined during learning without any external control except for parameterizing algorithms and defining the network architecture. For this reason, DNNs and neural networks in general are black boxes that do not offer any means for direct control over the mapping they compute. It is therefore extremely difficult to explain why some input triggers a specific output. Efforts towards making the decisions of neural networks *explainable* have so far yielded methods for visualizing learned features and tracing back predictions to those parts of the input data that triggered them [45]. Both approaches can be combined to a powerful tool set for inspecting the inner workings of a network [46] and getting an intuition of how information is processed along its layers. It is needless to say that visualization is by no means a replacement for formal verification methods. Work in this direction seeks, for example, to calculate reachable sets in the output for a given variation in the input data [47] or bounds for perturbations in the input region that do not affect the network's prediction for a given sample [48, 49]. Another class of approaches applies techniques from software testing to produce a set of inputs that maximizes the coverage of different network activation states [50].

Visualizing and verifying DNNs will make them gradually available to an increasing number of applications but it does not improve on their weaknesses and limitations. The monolithic design of current DNN architectures makes the reuse of trained models across different use cases hard. Adding knowledge after training is typically impossible since neural networks suffer from *catastrophic forgetting*, which means that the performance

of a network on the task that it was trained on degrades drastically as soon as training is continued on a new task [51]. Any modular reuse of complete networks or network components is thereby largely prohibited. This becomes a major issue as soon as the network processes multimodal input from different data sources. Their integration is completely intransparent and changes in one data source will most likely require re-training the full network. And it especially this use case that is of high relevance to many real-world applications in robotics.

Advances in Neuroscience: Towards Brain-Derived AI

The success of DNNs is often attributed to the fact that they take inspiration from the brain and perform brain-like computation. Indeed, recent research has revealed that they are the currently best models for neural activity in different regions of the visual system [52, 53] and many algorithmic innovations in the field can be argued to have some link to neuroscience [54]. In view of the substantial contributions DNNs have made to so many fields of AI, it does not surprise that more and more efforts are emerging that try to apply these findings to neuroscience [55, 56]. These successes should, however, not distract from the limitations identified above. Knowledge transfer from neuroscience mostly happens only for very small aspects and mainly for specific isolated functions such as attention or memory [54]. Goodfellow et al. [57] summarize the current situation as follows:

“The main reason for the diminished role of neuroscience in deep learning research today is that we simply do not have enough information about the brain to use it as a guide. To obtain a deep understanding of the actual algorithms used by the brain, we would need to be able to monitor the activity of (at the very least) thousands of interconnected neurons simultaneously.”

The narrow limitations of traditional experimental approaches have also become evident in neuroscience. Collecting measurements from experiments is slow and costly while at the same time the huge body of data available already now can no longer be captured efficiently by the simplified abstract models put forward in computational neuroscience. Brain simulations that integrate this data into a unified model offer a new perspective to the field. Like deep learning, constructing these simulations at a relevant scale has become possible only in recent years with the availability of sufficient amounts of affordable compute power and storage. Most notably, this development has triggered two major research efforts that have set out to revolutionize neuroscience: the Blue Brain Project (BBP) [58, 59] and the Human Brain Project (HBP) [7, 60]. While the goal of the BBP is to create and simulate a digital reconstruction of a juvenile rat’s neocortical column on a high performance computer, the HBP expands on this vision by developing a simulation-based digital infrastructure for virtual brain research. This sets it apart from the many other brain initiatives launched worldwide that promote primarily traditional

neuroscience. Both projects mark the beginning of a new direction of research called *simulation neuroscience* [4]. High-fidelity brain simulations open up a virtually unlimited space of exploration where new hypotheses can be tested within minutes and brain activity can be probed at any level of detail. They therefore not only have the potential of disrupting neuroscience but also to make substantial contributions to AI.

The coordinated efforts for setting up the technical research infrastructure that is required for large-scale brain simulations have already triggered a cross-fertilization between brain research and computer science. Besides the construction of optimized high performance computers [58], novel neuromorphic microprocessors are being developed that have architectures which are based on selected principles of neural information processing in the brain [61, 62]. This in particular means that they are designed to simulate spiking neural networks (SNNs) rather than standard ANNs to support the simulation of brain models from computational neuroscience. SNNs not only enable the implementation of a new set of learning rules [63], their execution on neuromorphic chips is also extremely energy-efficient [64]. One of the main challenges in the field is, however, the development of models and algorithms that actually leverage this potential. While the hardware is available, applications are still very limited since most of the current models and algorithms for SNNs do not even come close to the state of the art in deep learning. The big data approach put forward for DNNs is not directly applicable to neuromorphic computing. Knowledge transfer from neuroscience is therefore essential. Brain simulations seem to be the much needed link between neuroscience and AI with the potential to yield new *brain-derived* models that not simply take superficial inspiration from the brain but that reflect new computational principles discovered by neuroscience. It is important to note that this is a substantial difference to the prevalent *brain-inspired* systems that use findings from brain research mainly as a guide. This is, of course, not to say that brain-derived systems need to implement direct copies of what is observed in the brain. Instead, the focus is on the identification and adoption of the most relevant principles.

In practice, knowledge transfer between brain research and AI is hindered by incompatible goals: while the first aims to *describe structure* with measurements taken from the brain, the latter *implements a function* with data from the task. The main challenge towards brain-derived AI is therefore to bridge this gap and to *link structure to function*. In this work, we argue that the key to closing this gap lies in *embodiment*, the grounding of a brain model in an environment through a body by means of which it can act and interact. Through the actions of the body, the output of a brain model that has been defined in terms of its structure can be directly interpreted in terms of the resulting function. Seen in perspective, it seems only natural that a simulation of the brain can only be realistic if it receives the same input and output from the environment as a real brain does inside its body. So far, systematic research on embodiment was not possible because of a lack of the required tools and theories about how to connect a brain model to a body model. With the emergence of *neurorobotics* as a new scientific discipline, this is now about to change. The NRP developed in the HBP for the first time delivers a comprehensive tool set for the principled study of embodiment.

The Neurorobotics Platform of the Human Brain Project

The NRP is an integrated simulation environment for connecting simulated brains to simulated robotic bodies that has been developed by the Neurorobotics Subproject of the HBP [5, 6]. It is a programmatic effort aimed not only at providing a tool for neurorobotics research but at laying the foundations for a new methodology in neuroscience, robotics and AI that is based on the study of embodied systems. The research behind the NRP consequently goes considerably beyond the mere design, development and deployment of a software system. Simulations of both brain models and robot systems have been developed independently over decades. Connecting them to form a coherent embodied system where a body model and a brain model exchange sensory data and motor commands in a closed loop is only possible with a new framework that enables the definition of mappings between the two. Clearly, this is not only a technical issue but a fundamental theoretical challenge that includes questions on internal representations, neural coding and neuroanatomy.

The NRP's tool set is specifically tailored to addressing these questions through virtual *neurorobotics experiments*. Since all parts of these experiments are simulated, it is not only possible to replicate physical experimental setups but also to define completely new types of studies. Researchers are no longer constrained to any physical limitations but can

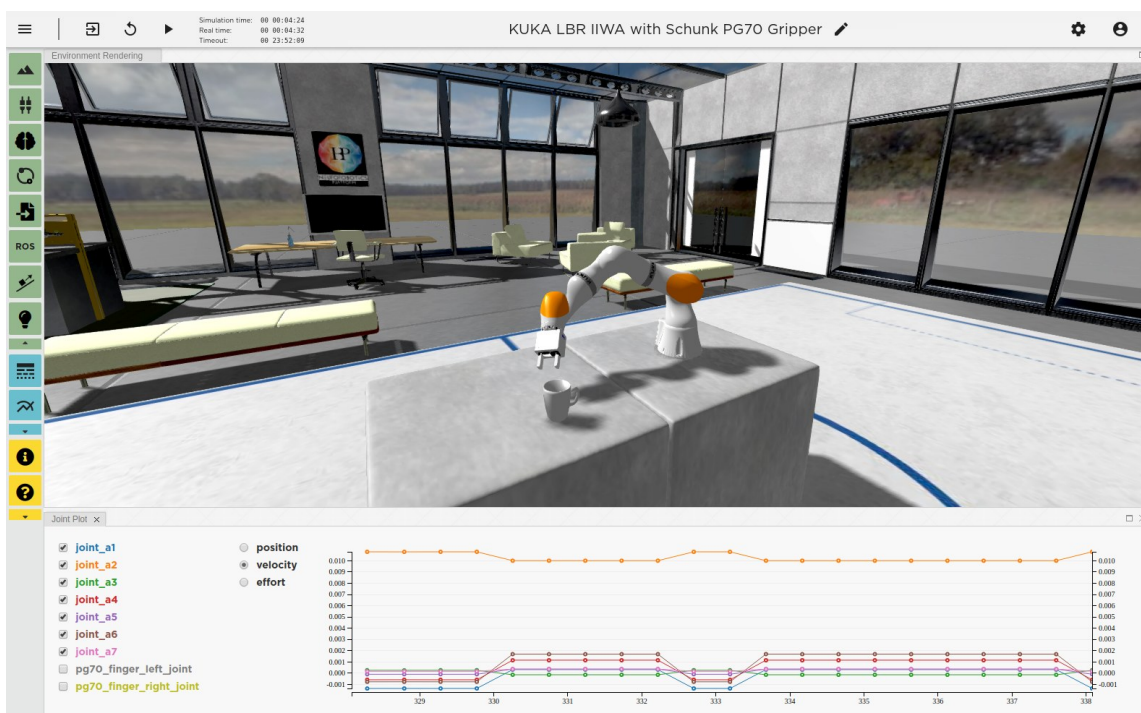


Figure 1.6: Web front end of the HBP Neurorobotics Platform. The screenshot depicts the platform's experiment designer, an interactive development environment for designing, running and visualizing neurorobotics simulations directly in the web browser.

observe the whole brain and the whole body at any level of detail. For example, whereas even today's most advanced technologies for single-neuron recording do not go beyond the scale of a few thousands of neurons [65], neurorobotics simulations enable the full observation of every neuron in the brain at any point in time.

Every neurorobotics experiment is comprised of four components: *a brain simulation*, *a robot simulation*, *transfer functions* and an *experiment protocol*. While the transfer functions define the flow of information between the brain model and the robot model, the experiment protocol determines how the experiment is executed. This is very similar to the physical world where protocols specify, for example, the steps required to start an experiment and the conditions under which measurements need to be recorded. The NRP makes all four components available through an interactive web front end that runs without installation directly in the browser. Figure 1.6 depicts a screenshot of the platform with an active experiment. The graphical user interface gives direct access to the simulated environment and contains dedicated editors for the brain model and the transfer functions. There is support for both DNNs and SNNs, which makes the NRP equally suited for AI and neuroscience.

That experiments can be designed simply in the web browser without installing any software packages is owed to the fact that the NRP is implemented as a cloud service. All computationally intensive tasks such as the brain simulation run in a high performance computing center. The web front end does not perform any computation but only serves as a graphical user interface for setting parameters, starting the simulation and visualizing the results. This architecture scales seamlessly with the amount of resources available and natively supports massively parallel simulations for testing multiple hypothesis at a time, parallelizing machine learning tasks or enlarging the space of exploration by testing a brain model in a multitude of different environments. Applications of this framework do not only lie in neuroscience but in particular also in the development of brain-derived systems, which renders the NRP a unique tool for the knowledge transfer from brain research to robotics, machine learning and AI in general.

1.4 Scope and Contribution of this Work

Up to now, none of the approaches put forward during the three waves of AI has managed to address the challenge formulated at the end of Section 1.1: *How can a newborn without any knowledge of the world around it learn to perceive, act and understand?* The goal of this work is to show that neurorobotics has the potential to spark a fourth wave of AI because it is the first research paradigm that is built on realistic embodiment and brain modeling as first principles. While no biological brain exists without a body, work in both AI and neuroscience is still primarily focusing on the study of disembodied systems. This is to some extent due to a lack of appropriate tools. Neurorobotics research was so far highly dependent on the availability of physical robots, which made progress costly and slow. The NRP now offers a completely new perspective by virtualizing neurorobotics experiments and thereby resolving all constraints of the physical world. In this work, we

take advantage of this new research opportunity to develop a novel neurorobotics-based learning framework that addresses the shortcomings of the current state of the art in machine learning by leveraging the unique properties specific to embodied systems.

Problem Statement

How can the challenge of designing a system capable of learning to perceive, act and understand be cast into a well-defined problem statement with a concise definition of preliminaries and expected outcome? In the scope of this thesis, we argue that the computational task solved by the brain can be formulated as a problem of *task-based sensor fusion*. Learning to perceive, act and understand then means learning how to integrate data from multiple modalities into a coherent representation that is optimized for the task at hand. Based on this hypothesis, the problem statement can be formulated as follows:

Let S be an embodied system comprised of a neural network-based brain model N and a body model B that tries to solve a set of tasks $T = \{t_1, \dots, t_m\}$ in an environment E . S interfaces with E through a set of sensory modalities $M = \{m_1, \dots, m_n\}$ and a set of actuators $A = \{a_1, \dots, a_o\}$ that are both part of $B = \{M, A\}$. How can N be designed so that the following three properties hold:

1. N solves a task $t \in T$ by interacting with E through A (*embodiment*)
2. N computes the actions for t by integrating data from M (*sensor fusion*)
3. N optimizes data integration for every $t \in T$ individually (*task adaptation*)

In terms of Section 1.1, S corresponds to the newborn while N and B are its brain and body, respectively. Analogously, M and A denote its senses and muscles. T could contain tasks such as grasping an object or crawling towards a goal in its environment E .

Contribution

This thesis addresses the problem stated above by three main contributions which are outlined in Figure 1.7. As illustrated in the diagram, the model input shifted closer towards direct data input from the physical world throughout the three waves of AI: Whereas symbolic reasoning required high-level abstractions from the ground truth, machine learning introduced feature space conversions of the raw input and deep learning finally enabled direct processing of raw data. In the course of this development, knowledge engineering evolved to data engineering, which rendered the availability of appropriate data sets the only remaining bottleneck. Neurorobotics completely alleviates the need for any manual engineering since the embodied system learns through interaction with the

environment and autonomously controls the collection of samples or, more appropriately, *experiences*. Based on this paradigm, we make three main contributions:

1. As explained earlier, the NRP can drastically speed up research in neurorobotics through the virtualization of experiments. To fulfill this potential, we develop a *framework for massively parallel virtual neurorobotics experiments* with the NRP. Our proposed approach supports modern cloud computing infrastructure and seamlessly scales with the amount of resources available. Experiments can be configured and controlled from a single point of entry and can exchange data through a shared document data base.
2. We develop a new *modular-hierarchical neural network architecture* that is optimized for task-based sensor data fusion along with a new learning algorithm that is based on a novel *topographic loss function*. Its design is directly derived from the architecture of the human cerebral cortex and has a structure that enables the flexible re-use of component networks across tasks. This makes computations more transparent compared to traditional monolithic deep neural networks.
3. We propose a new method for training modular-hierarchical neural networks based on insights from human development. In our approach, training is not performed from scratch on the full task but follows *training protocols* that increase task complexity over time and therefore shape the exploration space to optimize the learning process.

The framework for massively parallel virtual neurorobotics experiments is a technical prerequisite for implementing and training the proposed network architecture. The modular design of this architecture draws on findings from neuroscience about the structure from the brain. During training, the individual structural elements are directly linked to a function that is expressed through the behavior of the neurobotic system. Our approach thereby integrates principles from both neuroscience and AI to deliver a theoretical basis for the design of brain-derived systems. It is important to note the main motivation behind the proposed method is not to make a quantitative improvement over a system that is based on DNNs. The goal is rather to propose a qualitatively new methodology that directly benefits from advances in deep learning and at the same time accommodates relevant findings from neuroscience.

Structure and Outline

The structure of this work follows the scheme from Figure 1.7. Not shown are the Chapters 2 and 3, which introduce theoretical concepts in brain modeling and neurorobotics that will be relevant throughout this work. In Chapter 4, we develop a novel tool set for virtual neurorobotics based on the NRP that enables massively parallel experiments and adds support for the neuromorphic processor *Intel Loihi* [62]. The results are an important prerequisite for the following chapters. We further present the *TUM Robot Mouse*, a

3D printed biomimetic mouse robot with an actuated spine. It is designed with the goal of providing a link between virtual and physical neurorobotics. In Chapter 5, we develop the modular-hierarchical neural network architecture along with a training method that is based on novel topographic loss function. We then introduce the concept of training protocols in Chapter 6. All proposed models will be implemented and evaluated in the NRP. Chapter 7 concludes this work and provides an outlook to practical applications and future directions of research.

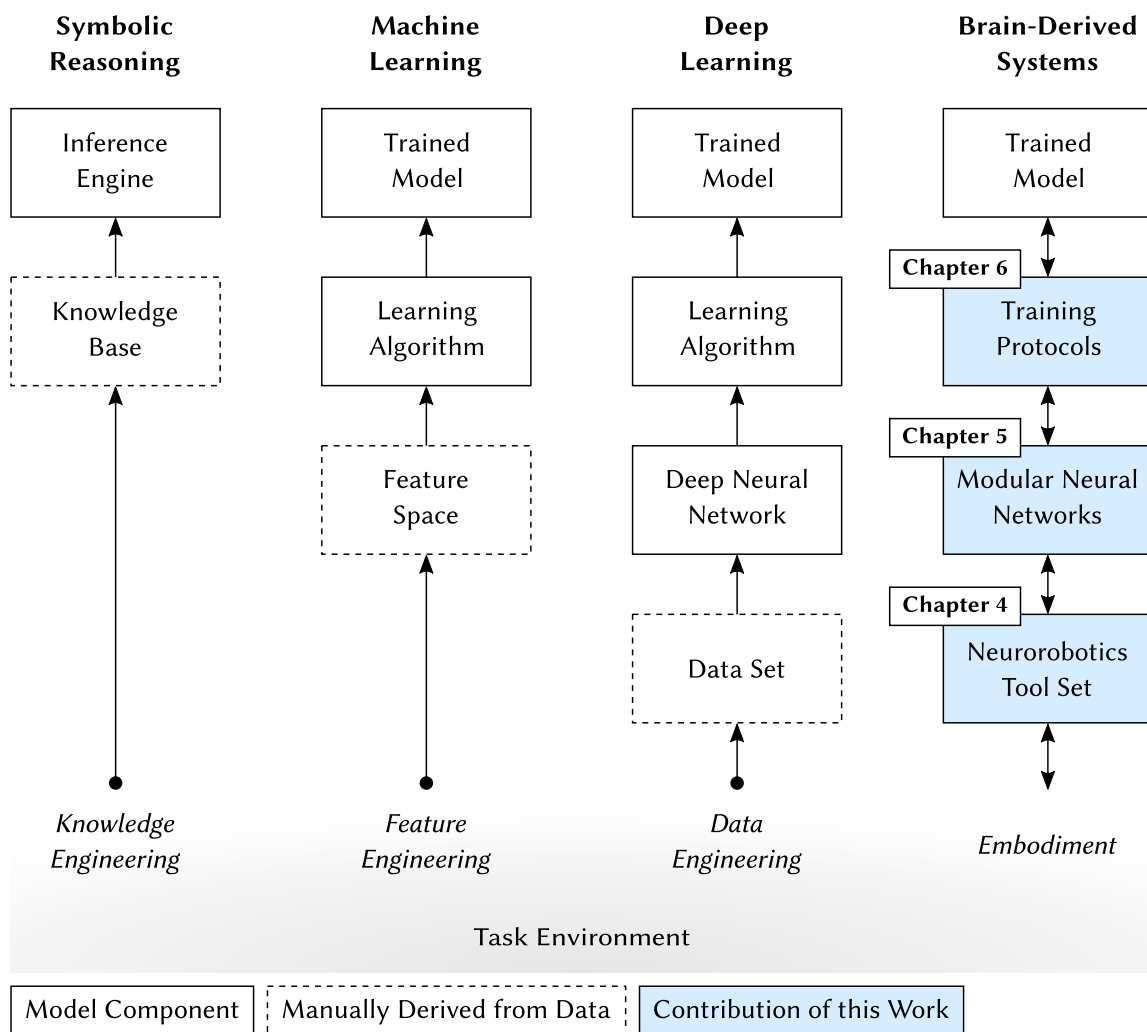


Figure 1.7: Contribution of this work in the context of the three waves of AI as defined in Section 1.2. Neurorobotics resolves the bottlenecks of symbolic reasoning, machine learning and deep learning through embodied closed-loop interaction with the task environment. Information from this environment no longer needs to be processed manually but is automatically collected by the embodied system based on the task it tries to achieve. The concept of the figure is based on [57].

2

Models of the Brain

Biological brains enable living creatures to survive under hostile conditions in ever-changing environments. They are capable of processing incomplete and noisy sensory input, reasoning about both concrete situations and abstract ideas, looking ahead, planning the next steps and controlling the body that connects them with the outside world. They continuously learn from experience and adapt to changes on the fly. The neural control system that gives rise to this capability is arguably one of the most intricate structures known to date. The numbers in Figure 2.1 give proof of its impressive size and complexity. How is it possible to understand such a complex structure that is so delicate and inaccessible that even today's most advanced technology can only capture a tiny fraction of its inner workings?

Throughout history, there have been many attempts to explain how the brain works. First mentions of the brain as an organ even date back to ancient Egypt more than 5000 years ago [73]. However, it was not before the beginning of the 20th century that modern neuroscience was established as an acknowledged and scientifically grounded field of research. This development would not have been possible without the invention of a new tool. In 1873, the Italian biologist Camillo Golgi discovered a new staining method that for the first time enabled the identification of individual cells in neural tissue, which laid the foundation for the description of the brain as a network of independent interconnected neurons put forward by the Spanish histologist Santiago Ramón y Cajal [74]. Since then, progress in neuroscience has continued to be driven by innovations in tools ranging from electron microscopy to electroencephalography (EEG) and functional magnetic resonance imaging (fMRI). But even today's most advanced measurement devices produce highly fractional data that reflect only a tiny portion of the whole brain. Devising models that interpret and predict experimental findings has therefore become essential. The first mathematical descriptions of the nervous system were conceived long before the invention

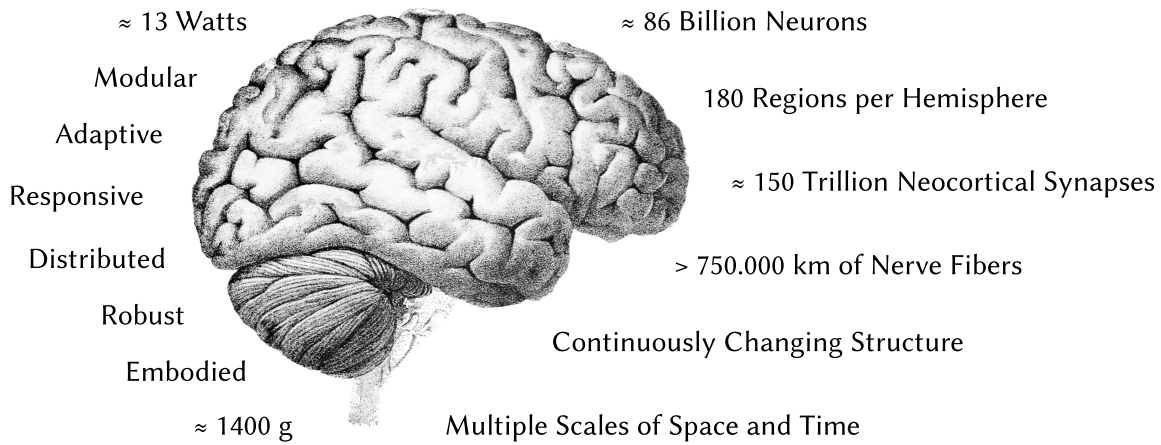


Figure 2.1: The complexity of the human brain in facts and figures. The data are based on [66, 67, 68, 69, 70, 71], the drawing of the brain is a modified version of the original from [72] (public domain, PD-US-expired).

of the computer. Today, they have become a cornerstone of modern neuroscience and also played an important role in the development of DNNs.

Brain modeling is the starting point for knowledge transfer between neuroscience and AI, which makes it an indispensable prerequisite for the design of brain-derived devices. This chapter introduces basic theoretical concepts of neural brain models. After identifying a level of abstraction that is appropriate for the scope of this work, we introduce the common foundations of neural networks in neuroscience and AI before discussing the specific developments in each field in detail. The last section of this chapter highlights how insights from brain modeling are driving the creation of novel neuromorphic chip designs that aim to mimic computational principles which are hypothesized to be implemented in the brain.

2.1 Levels of Detail in Brain Modeling

The complexity of the brain renders meaningful abstractions indispensable. This is particularly exemplified by the astonishing numbers from Figure 2.1. Modeling the brain therefore requires focusing on features that are most relevant in a particular context, which means that every model is necessarily defined within a certain *scope* and serves a specific *purpose* [75]. In *theoretical neuroscience* and *computational neuroscience*, one of the main purposes of modeling is to *quantitatively* explain a set of experimental findings or to provide quantitative predictions that can be verified or falsified experimentally. This work, by contrast, uses neuroscientific findings to guide the design of a model that solves a concrete task in a given environment on a specific computational substrate. Compared to neuroscience, where the type of model being used is partly constrained by the experimental data source, this results in a vast space of modeling approaches. It is therefore essential to determine the right type of model at the most appropriate level of abstraction that fits the intended purpose best.

Types of Brain Models

In the context of this work, we will only consider brain models that can be executed as *brain simulations* on a computer. Of special relevance in this context is the classification by Dayan and Abbott [76], who have identified three different types of models that are typically considered in theoretical neuroscience. As further elaborated by Levenstein et al. [77], each of them answers a specific question:

- *Mechanistic Models*: “How does the phenomenon arise?”
Mechanistic models capture the actual implementation of a phenomenon in terms of its low-level building blocks and their interactions with each other.
Example: Microarchitecture of a microprocessor that describes its implementation in terms of logical gates.
- *Descriptive Models*: “What is the phenomenon?”
Descriptive models capture an observed phenomenon without specifying the underlying building blocks and processes that give rise to it. They are often also referred to as *phenomenological models*.
Example: Description of a microprocessor in terms of assembly commands with corresponding inputs and outputs.
- *Interpretive Models*: “Why does the phenomenon exist?”
Interpretive models “explain a phenomenon in terms of a function and goal” [77]. They thereby play an essential role in relating a specific phenomenon to its role for cognition and behavior.
Example: Description of different functional units of a microprocessor and the features that they implement.

These modeling types are closely related to the three levels of understanding proposed by Marr [78], who distinguishes between *computational theory* (interpretive), *representation and algorithm* (descriptive) and *hardware implementation* (mechanistic). It is obvious that mechanistic

However, it is important to note that mechanistic, descriptive and interpretive models cannot be treated in isolation from each other. They are interdependent and a model of one type can be comprised of models of another type. In the microprocessor example, the gates of the mechanistic model are themselves descriptive models of the underlying electrical circuits. In this sense, mechanistic models can be seen as links between different levels of abstraction, which are in turn described by descriptive and interpretive models [77]. For this reason, this work will start from a mechanistic model in order to link behavior to the underlying neural circuitry. Identifying the most appropriate level of modeling for this task crucially depends on identifying the levels available.

The Multiscale Organization of the Brain

As outlined in the definition above, mechanistic models require the specification of a system's building blocks and the dynamics of their interactions. With the brain, neither of these two are fully known to date, which is why it has become common to draw analogies to technical systems in an attempt to explain its inner workings. Over time, these analogies have evolved along with technological development and range from steam engines and radio frequencies to, most recently, circuits, networks and computations [79]. What is common to all these systems is that they are compound structures that are made up of many different components, which themselves can be further diversified into sub-systems. In a steam engine, for example, a connecting rod and a flywheel convert steam pressure generated in a boiler into rotational motion, a radio is comprised of an antenna, an amplifier and speakers, and a computer is built from a multitude of electrical circuits, each of which has up to billions of transistors. All three devices spearheaded the state of the art in engineering in their day and age. Breaking them down into different sub-systems at different levels of abstraction has become the key to managing their complexity. Is it also possible to identify such structure in the brain?

A key contribution in this direction, which had lasting impact on neuroscience, was made by the German neuroanatomist Brodmann at the beginning of the 20th century [80]. Based on his studies of the cytoarchitecture of the of the human brain, he proposed to subdivide the cerebral cortex into distinct areas. The term *cytoarchitecture* refers to “the arrangement of cells in organs and tissues, particularly those in the neocortex” [81]. Figure 2.2 shows his original drawings. The left sketch provides a detailed overview of the identified areas and the numbers they were assigned. In the map on the right, they are grouped into larger brain regions, which can already be seen as a form of abstraction. The Brodmann areas have become an important tool in neuroscience. In more recent

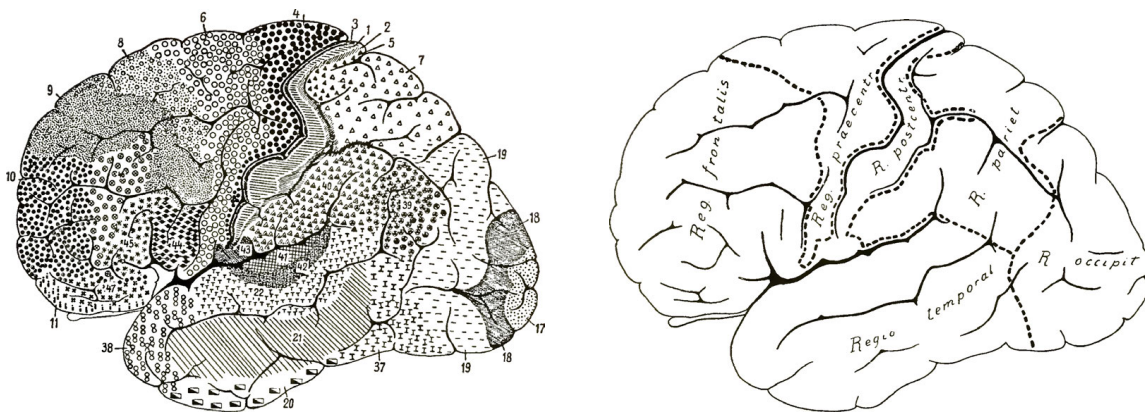


Figure 2.2: Drawings of the Brodmann areas for the lateral view of the human brain. Brodmann originally identified 52 different distinct areas. *Left:* Detailed map with areas defined by cytoarchitectonic properties. *Right:* Main brain regions extracted based on the map on the left. The drawings are from [80] (public domain).

research, some of them could be directly mapped to specific cognitive functions and state-of-the-art imaging techniques suggest the existence of 180 areas on each hemisphere of the human brain [70]. In a modern version of the map from Figure 2.2, a high-resolution brain atlas has been created with the goal of establishing a new reference map that serves as a common basis for future research [82].

The ongoing efforts to map the building blocks of the brain have yielded important insights as to how individual cells at the microscopic level form different areas at the macroscopic level that in turn can, in some cases, be assigned specific cognitive functions. Further research on each of these levels has revealed additional structures and layers of organization, the most important of which are summarized in Table 2.1. At the lowest level, molecular processes form the basis of cell physiology and determine how genetically encoded information is expressed in the properties of a multitude of different types of neural and glial cells. One level above, neurons form circuits that in turn are the building blocks of the brain regions that have been partially identified by Brodmann. At the highest level of organization, the whole brain itself is embedded in a body that provides an interface to the external world. The different levels of abstraction haven been specifically chosen for the context of this work, but can also be identified with the three levels of Marr [78] mentioned before: *system integration* corresponds to computational theory, *cognitive architecture* to representation and algorithm, and *physical implementation* to hardware implementation. *Logical implementation* contributes to both of the last two in the list.








The components and processes along the hierarchy sketched in Table 2.1 span multiple orders of magnitude in both space and time. Every macroscopic event, such as a small movement of a finger tip, entails and is entailed by millions of cellular processes at the microscopic level that are governed by molecular processes at the sub-microscopic level. While the former may take seconds, the latter can happen within milliseconds. The challenge of brain modeling lies in addressing those levels that are relevant for the phenomena of interest. In general, these levels are not uniquely defined and, independent of the investigated phenomenon, typically all levels are involved to some degree. The mapping of phenomena to levels in the table therefore only serves as a guideline, but not as a fixed assignment.

Balancing Abstraction and Detail

The task of a newborn making sense of its environment, formulated in Chapter 1, is clearly defined at the uppermost level of organization in Table 2.1. Any candidate brain model that is viable for addressing this challenge must therefore necessarily encompass all functions and features that arise from the organization levels below. At first glance, there are two obvious strategies for implementing such a model. In a purely interpretive modeling approach, the entire brain can be described in terms of its cognitive skills that can be technically realized as arbitrary computer programs. Optionally, some of these programs may be identified with specific brain regions based on the functions they implement. The result is a set of different cognitive processes that are orchestrated to give rise to the desired behavior. Such a design strategy lends itself very well to common practices

2 Models of the Brain

Table 2.1: Multiscale organization of the brain. The table covers only the most distinct levels of organization, each of which can be further subdivided. Likewise, the attribution of phenomena to single levels does not express a strict mapping but highlights common cases. Distances and times indicate the overall scale and are partly based on [83, 84]. Molecular dynamics spans itself several time scales, which is why no order of magnitude is specified. The drawing of the brain is adapted from a digitized version of the original in [85] (public domain). Other sources of the drawings are referenced in Figures 1.2 and 2.4.

	Level of Organization	Relevant Phenomena	Level of Abstraction	Spatial Scale	Temporal Scale
Environment		Behavior and Interaction	System Integration	10^0 m	Minutes to Years
Organism		Sensing and Actuation			
Brain		Cognitive Processes	Cognitive Architecture	10^{-2} m	Seconds to Minutes
Brain Regions		Cognitive Functions			
Circuits		Neural Dynamics	Logical Implementation	10^{-3} m	Milliseconds
Neurons		Information Coding and Processing			
Molecules		Genetics and Molecular Processes	Physical Implementation	10^{-9} m	—

in software engineering and has been specifically put forward in the field of *cognitive architectures* with many applications in AI and robotics [86]. At the other end of the spectrum, one could try to construct a fully mechanistic bottom-up model that is defined solely in terms of molecular processes.

It should be obvious that neither of the two alternatives outlined above would provide a desirable brain model. While the first would be too coarse and too arbitrary to establish a meaningful quantitative relationship to neuroscience, the second would not be manageable with respect to complexity and computational requirements. These two extremes exemplify the trade-off between abstract models that take inspiration from the brain at the conceptual level and detailed models that are directly derived from neuroscientific findings. The most important constraint for choosing the right level of balance between them is

that the mechanistic interactions between low-level components can still be interpreted in terms of the full system's behavior. This is impossible for models defined at molecular level since the model size for tracing down an organism's actions to the dynamics of individual molecules is prohibitively large. Since at the same time the whole brain and its regions have been identified as too coarse above, the possible levels of organization are narrowed down to neurons and neural circuits. The decision for one of the two can be constrained by the intended application and the available technical infrastructure.

Neural circuits are commonly described with neural mass models and neural field models that abstract from single neurons by capturing the dynamics of larger neural populations. Whereas mass models only address the temporal dynamics of population activity, mean field models also represent its spatial distribution [87]. The main purpose of these models is to provide a compact mathematical description of brain activity as recorded by EEG or fMRI. Unlike the abstract cognitive architectures discussed before, they can directly incorporate and explain experimental data while still being simple enough to be analytically tractable [88]. Neural field models have been successfully employed for the construction of larger brain models comprised of different subsystems [89] and as components for a cognitive architecture in robotics [90]. It has been argued that they are suited as a basis for embodied cognitive systems [91]. However, neural fields by design fall short in capturing information about the detailed structure of the neural circuitry that they represent. Specific questions about network architectures and fine-grained connectivity patterns can therefore not be addressed at all. Moreover, software and hardware development in recent years has had a strong focus on simulating neural network models that are built from discrete neurons. Additionally, large-scale network simulations are becoming an increasingly important tool in neuroscience. The focus on populations in neural field modeling can leverage this progress only to a limited extent. In the remainder of this work, we will therefore only consider only brain models that are defined at the level of neurons.

Discrete Neuron Models

Biological neurons are intricate structures with a complexity that arguably even matches that of the whole brain. Neuron models are therefore highly diverse and cover numerous levels of abstraction akin to those outlined in Table 2.1 for the whole brain. Herz et al. identify five modeling levels, the following three of which are especially relevant in the scope of this thesis [92]:

- *Detailed Compartmental Models*: The functional properties of a neuron are closely related to its spatial geometric structure or, more precisely, its *morphology*. This type of model approximates morphological properties by discretizing them into compartments, each of which has an own set of state variables.

- *Reduced Compartmental Models*: Detailed compartmental neuron models can be comprised of hundreds of compartments [59]. Reduced compartmental models abstract from exact morphology and use compartments only to differentiate between the most salient structures of a neuron. This not only reduces computational complexity but also makes the system dynamics analytically tractable.
- *Point Neuron Models*: This class of models completely neglects spatial structure and describes neural dynamics with a single compartment, which is why it is also referred to as *single-compartment models*.

Even though they are computationally extremely demanding, brain models that are built from networks of detailed compartmental neurons can be simulated at large scale today [59]. The massive amounts of required compute power and the huge modeling effort make them currently especially attractive for use cases in neuroscience where an exceptionally high level of simulation fidelity is required. For research on how higher-level phenomena arise from low-level neural dynamics, neurons can be approximated sufficiently well by point neuron models. That this simplification is valid is supported by a recently published method for the conversion of detailed compartmental models to phenomenologically equivalent point neurons for a given activity regime [93]. Moreover, a single complex neuron model can be approximated by a network of multiple point neuron models [92]. Focusing on the latter consequently does not infer any computational limitations. It does, however, provide a direct link to AI, where all current methods in machine learning are based on point neurons.

Neural Information Coding: Spikes and Rates

Biological neurons transmit information through digital electrical impulses called *action potentials* or *spikes*: the only information a spike carries is the time of its occurrence. Consequently, all spikes emitted by a neuron have identical shape [94]. Information can either be encoded in the *precise spike timing* or by temporal averaging over a spike train to compute a *spike rate*. Deciphering the neural code used by the nervous system is one of the most central questions in neuroscience and there are both models and experimental evidence that support each of the two coding schemes. An overview of different codes is provided in Figure 2.3.

In rate-based coding schemes, the identity of a neuron represents the stimulus or feature that it encodes. Rate changes signal updates of the stimulus value that is in turn encoded by the magnitude of the firing rate [95]. The early prevalence of rate-based modeling is rooted in a huge body experimental studies where recorded stimuli could be explained very well through rate codes. In this context, sensory cells are of particular interest since they allow for a direct observation of relationship between an input signal and the encoded neural output. Examples include mechanoreceptors in the skin where, depending on the receptor type, the strength of a stimulus translates proportionally into a firing rate [96]. Rate codes are also involved in higher levels of cognitive processing.

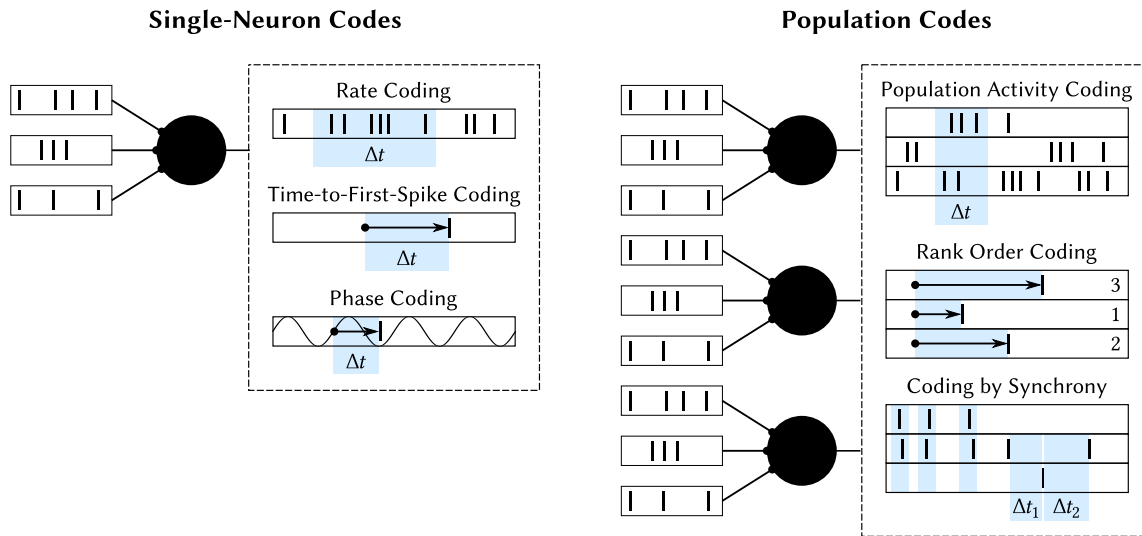


Figure 2.3: Overview of different types of neural codes. *Left:* Single-neuron codes are realized by an individual spiking neuron. Besides simple averaging-based rate coding, temporal codes can be expressed as the timing of a spike with respect to a certain reference point (time-to-first-spike coding) or the phase of some oscillatory background signal (phase coding). *Right:* Population codes are defined based on the output of a set of neurons. Examples include population activity coding as a natural extension to single-neuron rate codes and temporal coding through relative spike times (rank order coding) or synchronized spike patterns (coding by synchrony). The latter also includes correlated sequences of spikes with fixed inter-spike intervals (coding by correlation). The overview is based on [94]. The visualizations are adapted from [63] by permission from Springer Nature Customer Service Centre GmbH, ©2016.

For example, neurons in the visual cortex respond only to specific visual stimuli [97, 98] and grid cells in the entorhinal cortex fire as soon as the body is located on a vertex of the spatial grid encoded by the cell [99]. A major drawback that is typically attributed to rate coding schemes is the time required for decoding by temporal averaging. Maximum spike rates imposed by physiological constraints moreover place a hard limit on the resolution and value range that can be represented by rate coding. A potential remedy for this issue are population codes, where individual values are encoded by the overall spike rate of a set of neurons, which enables faster readout without limitations imposed by maximum firing rates. In fact, the representation of visual stimuli or spatial locations by cells with specific *receptive fields* in the visual cortex and the entorhinal cortex, respectively, is also a form of population coding where each cell in the population represents a range of stimulus values. In motor neuron populations that drive muscles, both types of rate coding have found to be combined since muscle force is controlled by both firing rate and the number of motor neurons activated [100]. The latter control scheme is also known as the *recruitment principle*.

A potential advantage of temporal codes as opposed to rate codes is that the amount of spikes required to transmit information can be considerably reduced. This has been argued to be especially relevant in the visual system that is capable of recognizing a photograph in less than 150 ms [101]. Rate codes are most likely not sufficient to transmit

information fast enough due to the delays caused by spike count averaging, which is why rank order coding was proposed where only the relative sequence in which a set of neuron fires is relevant [102]. Concrete experimental evidence for temporal spike coding was observed from spike train recordings in the cortex during a fixed behavioral task that contained repetitions of spike patterns emitted by the recorded populations [103]. More recently, affirmative results have been reported with more advanced probing and analysis methods [104]. Importantly, the identified patterns could be mapped to specific behaviors. In general, the synchronous firing of neural populations is hypothesized to be an important mechanism for the coordination of distributed neural information processing [105]. As Uhlhaas et al. [105] argue in their review, network oscillations play a key role in temporal coding since they provide a reference for the synchronization of neural activity or, in an alternative coding scheme, the encoding of information through phase-shifted spike emission.

Choosing to transmit information through a temporal code narrows down the class of possible neuron models to those that are capable of producing spikes. Rate codes can also be implemented with simplified analog neuron models that approximate spike train frequencies with real numbers. While typical ANNs are computationally less complex, it has been shown that SNNs can implement certain functions with considerably less units [106]. This trade-off is similar to that between multi-compartment and single-compartment neuron models. One must, however, not mistakenly conclude that SNNs should therefore be preferred. In general, there is not always a clear separation between rate-based and spike-based codes. For example, the time-to-first-spike coding scheme from Figure 2.3 can also be interpreted as a rate code since Δt will also grow smaller for increased firing rates [94]. In conclusion, both SNNs and ANNs are legitimate modeling choices. The decision for one class of neuron models over another mainly depends on the specific computational features that are required for the task at hand and, in the case of neuroscience-driven studies, which types of measurement data are available. It is important to note that spikes may not or not only serve a computational purpose. Sensory neurons, for example, encode stimuli by analog voltages before they are converted to spikes for transmission across longer distances in the nervous system [96]. That the stimulus is not transmitted by analog values is therefore at least partly enforced by the physical constraints of biological nervous tissue. Spike-based signaling can also be implemented highly efficiently in CMOS-based electrical circuits as will be explained in Section 2.4.

To summarize the findings above, this thesis will focus on both spiking and non-spiking point neuron models. This level of abstraction is well balanced to bridge the scale between embodied behavior and neural activity. At the same time, it allows for the efficient hardware-accelerated simulation of large-scale network models.

2.2 Common Roots of Brain Simulation and Artificial Neural Networks

The neuron doctrine put forward by Cajal was the starting point for research on single neurons and their modeling. With the discovery of neural cells as the constituent elements of information processing in the brain, their identification and characterization became one of the main research endeavors of modern neuroscience. While Cajal's initial work solely relied on morphological descriptions of different cell types, technological progress enabled the analysis of additional features such as physiological and molecular properties [107]. What is common to all of these methods is that they reveal the overwhelming complexity of both structure and function that is present in every single neuron. The multitude of morphological phenotypes, connectivity patterns and biochemical processes has challenged neuroscientists from the beginning on and the quest for a complete census of neural cell types in the brain is still ongoing [4].

Building Blocks of Biological Neurons

Despite this diversity, virtually all neurons share similar qualitative functional characteristics, the arguably most important of which is the already mentioned *action potential* or *spike*. The constituent elements of spike-based neural information processing are illustrated in Figure 2.4. A typical neuron is comprised of a cell body, the *soma*, *dendrites* and

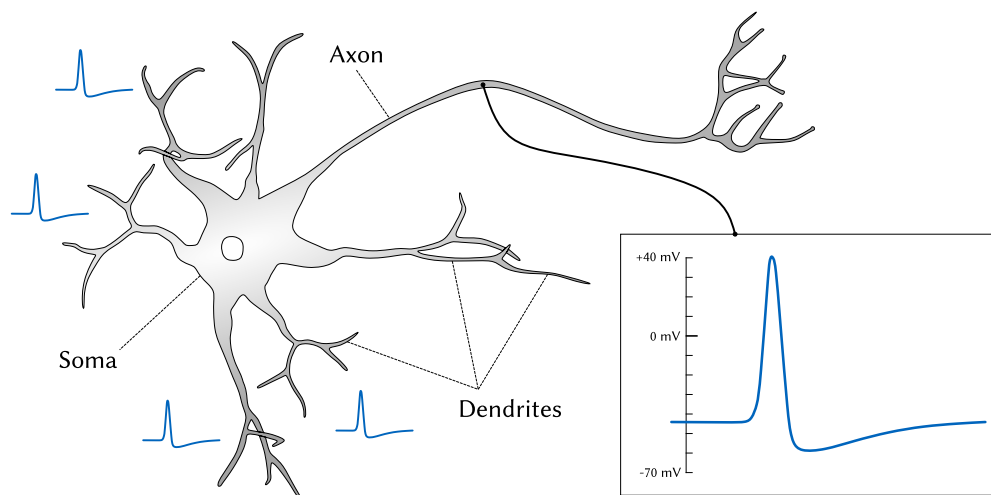


Figure 2.4: Generation and conduction of action potentials in biological neurons. Action potentials emitted by presynaptic neurons travel along the *dendrites* and are integrated in the *soma*, resulting in a rise of the membrane potential. As soon as the potential crosses a threshold, the neuron emits a spike that is propagated along the *axon* towards postsynaptic neurons. The sketch of the neuron structure is based on [108], the action potential trace is redrawn from the original recording in [109].

an *axon*.¹ It receives input stimuli from *presynaptic* neurons through synaptic connections at the dendrites that propagate incoming action potentials towards the soma where they accumulate by increasing the *postsynaptic* neuron's membrane potential. As soon as the potential rises above defined threshold, the neuron elicits a spike that travels along the axon towards the dendrites of other neurons. First recordings of the very specific voltage trace of an action potential date back to the seminal work of Hodgkin and Huxley [109] who for the first time reported exact measurements in the giant axons of squids in 1939. These data later led to the formulation of the Hodgkin-Huxley neuron model in 1952 [110], which has remained one of the most detailed and expressive descriptions of neural dynamics until the present day. However, whether a neuron model is appropriate or not depends not only on its fidelity with respect to the reproduction of electrophysiological ground truth but also on the actual scientific question at hand. This explains why there is a plethora of different neuron models today and why simpler ones have not been superseded by more complex ones. The following subsections will highlight the common roots of today's main streams of development and explain how modern ANNs and brain simulations have emerged from them.

The Leaky Integrate-and-Fire Neuron Model

Even though the Hodgkin-Huxley neuron laid the foundations for modern high-fidelity brain simulations, it was not the first mathematical description of a neuron. Already in 1907, Lapique introduced a spiking neuron model that is today widely known as the *leaky integrate-and-fire (LIF) model* [111, 112, 113]. It describes the temporal dynamics of a spiking neuron with a functionally equivalent electrical circuit, an approach that has evolved to one of the arguably most important techniques for the modeling and simulation of single neurons. The main constituents of the model are summarized in Figure 2.5. As indicated by the charges in the neuron at the top, the soma is negatively charged at resting state with respect to the outside of the cell. While the exact value of this *resting state potential* is influenced by many different factors, a common value that is often assumed in simulations is -70 mV. When the neuron receives input through its synapses or when it emits a spike, the membrane potential changes. The influx and efflux of charges is regulated by the cell membrane, which insulates the inside of the neuron from its outside. Charges can only enter or leave through state-dependent *ion channels* that are embedded in it. They control the generation of action potentials through an intricate chain of biophysical processes. The functionally equivalent electrical circuit of the LIF neuron abstracts from all these details and subsumes the complex interplay between ion channels by a capacitor C that *emulates* the behavior of a neuron's cell membrane. External input such as spikes or electrical stimulation are represented by the time-varying input current $I(t)$. In the absence of any stimulus, C discharges through resistor R until

¹It is important to note that there are also other types of neurons with a different morphological structure. In this work, we will only consider multipolar neurons as the one depicted in Figure 2.4.

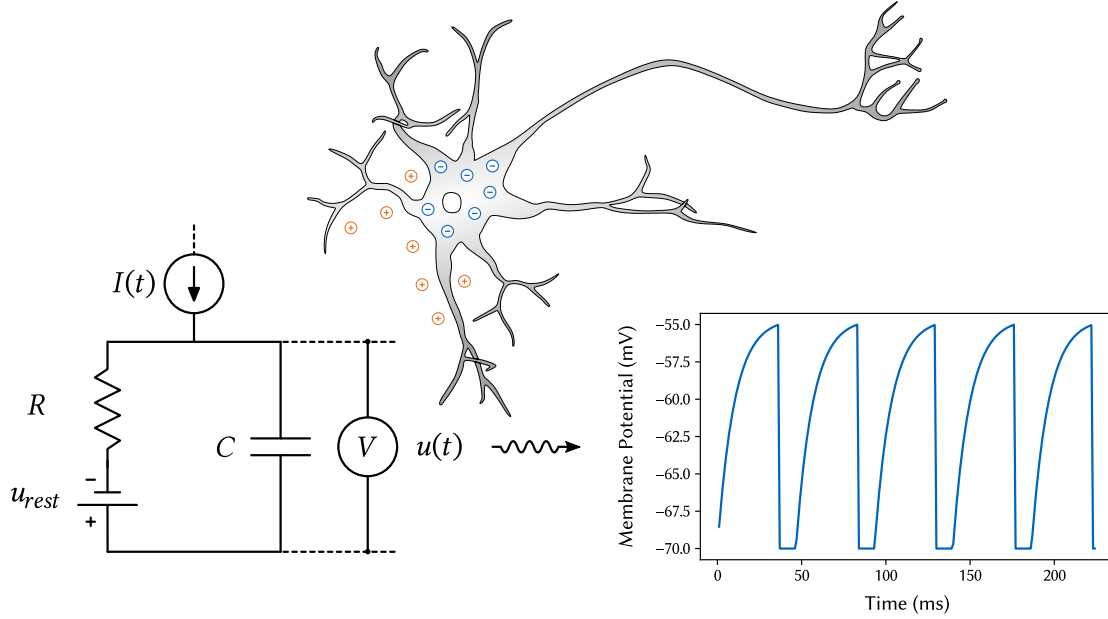


Figure 2.5: Functionally equivalent electrical circuit of a LIF neuron and the resulting membrane potential trace for a constant input current $I(t) = I_0$. As depicted in the sketch of the neuron at the top, the membrane potential is negative at resting state. Charges are separated by the neuron’s cell membrane, which is *emulated* by capacitor C . The figure is partly based on [94], the membrane voltage trace was generated with NEST [114].

the membrane potential $u(t)$ reaches its resting state at u_{rest} . The overall current in the circuit therefore amounts to:²

$$I(t) = I_R(t) + I_C(t) \quad (2.1)$$

Both currents I_R and I_C that flow through R and C , respectively, can be expressed in terms of the current membrane voltage $u(t)$ by applying Ohm’s law and using $I(t) = \dot{q}(t) = C\dot{u}(t)$:

$$I(t) = \frac{u(t) - u_{rest}}{R} + C\dot{u}(t) \quad (2.2)$$

When substituting the *membrane time constant* $\tau_m = RC$ in the equation above, one arrives at the canonical representation of the *subthreshold dynamics* of the leaky integrate-and-fire neuron:

$$\tau_m \dot{u}(t) = -(u(t) - u_{rest}) + RI(t) \quad (2.3)$$

Equation 2.3 is an ordinary linear differential equation that can be solved analytically [94]. It describes the dynamics of the neuron’s membrane potential as long as the neuron does not emit a spike, which is commonly referred to as the subthreshold regime. For a

²The following derivation of the canonical mathematical representation of the standard LIF neuron is based on [94].

constant input current $I(t) = I_0$, the solution $u(t)$ is an exponential function. This explains the specific shape of the rising edges in the plot on the right side of Figure 2.5. What is, however, not modeled by Equation 2.3, is the actual generation of spikes that are triggered every time the membrane voltage crosses a fixed threshold ϑ from below. As plotted in the voltage trace, this leads to an instantaneous decay of the membrane to voltage to the reset potential. The spike generation is implemented by a separate reset mechanism that monitors $u(t)$ and becomes active as soon as ϑ is reached from below:

$$\forall t_f : u(t_f) = \vartheta \quad \rightarrow \quad \lim_{t \rightarrow t_f^+} u(t) = u_{reset} \quad (2.4)$$

In the equation above, t_f denotes the neuron's firing time. Note that the reset potential u_{reset} can be different from u_{rest} . Typically, it is chosen to be lower than u_{rest} in accordance with measurements from biological neurons. A single spike at time t_f is commonly represented by the Dirac δ function $\delta(t - t_f)$. The set of all spikes emitted by a neuron is referred to as a *spike train* $S(t)$ that is accordingly defined as follows:

$$S(t) = \sum_f \delta(t - t_f) \quad (2.5)$$

Equations 2.4 and 2.5 complete the definition of the basic leaky integrate-and-fire neuron model. Practical implementations may feature additional parameters, e.g. to control the time of the neuron's refractory period after an action potential during which no other spike can be generated. In the plot of Figure 2.5, this dead time is visible as small plateaus of constant membrane potential between voltage resets and subsequent rising edges of $u(t)$.

The probably most salient feature of the LIF neuron is that it does not explicitly model the process of spike generation in spite of being a spiking neuron. Instead, it only outputs the resulting spike train $S(t)$. By contrast, the already mentioned Hodgkin-Huxley model features a complex system of differential equations that capture the full cycle of action potential generation without the need for any external reset mechanism. It contains explicit mathematical descriptions of the dynamics of the ion channels in the neuron's membrane and represents them by means of electrical conductances. This is why the Hodgkin-Huxley model belongs to the class of *conductance-based* neuron models and is arguably one of its most important representatives. The LIF neuron is a *phenomenological* model that abstracts from biological detail and only qualitatively captures the experimentally observed behavior without accounting for the underlying electrophysiological processes. At first glance, this makes it seem inferior to conductance-based neuron models. However, in many applications, the specific nature of action potentials renders the aforementioned inaccuracies practically irrelevant. Unlike other processes in biology, spikes are digital all-or-none events. All information is encoded in the fact whether a spike has occurred or not. The specific shape of a spike, i.e. the course of the membrane potential during its emission, is irrelevant and typically identical across different occurrences and neurons. Phenomenological neuron models exploit this property to simplify the mathematical description. This makes them not only analytically

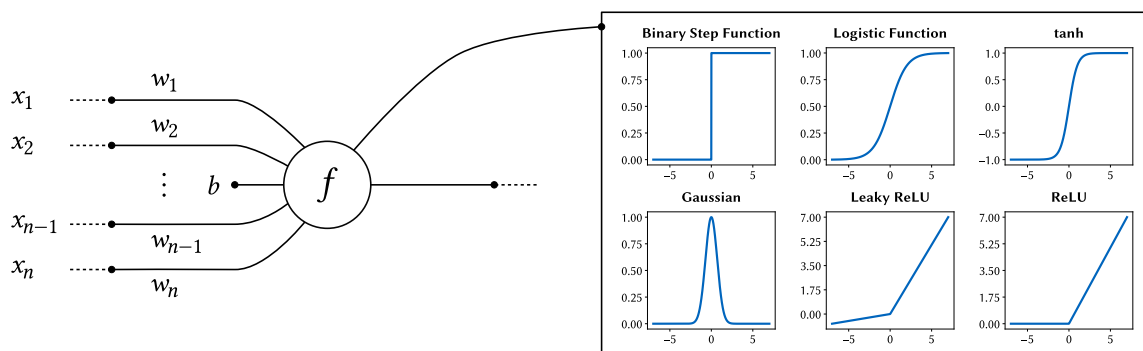


Figure 2.6: Schema of a general analog neuron model. Inputs x_i are weighted by their corresponding synaptic weights w_i before they are added. The neuron's output is computed by applying an activation function f to the weighted sum. Typically, the bias term b is replaced by a fixed input $x_0 = 1$ with weight b .

tractable but also easier to simulate, especially in large-scale models where not the state of only tens to hundreds but rather of millions to billions of neurons needs to be tracked. For research in spike-based neural information processing where the focus is on computational properties and not on biological modeling, the LIF neuron and its extensions are therefore still among the most commonly used neuron models to date.

McCulloch-Pitts Cells

Even though LIF neurons are a considerable simplification of biological ground truth, their temporal dynamics together with the discontinuous voltage reset mechanism span a complex state space. This state space might be already too complex under the assumption that only the occurrences of spikes are relevant while all other state variables such as the membrane potential serve no specific computational purpose and are consequently negligible in terms of information processing. This proposition formed the basis of the seminal work by McCulloch and Pitts [32], who formulated a drastically reduced neuron model in 1943. The McCulloch-Pitts cell has the same all-or-none output behavior like spiking neuron models. However, external input is not represented through currents that sum up in a capacitor. Instead, the neuron spikes as soon as it receives a minimum number of concurrent excitatory inputs from presynaptic neurons. If only a single inhibitory synapse becomes active, the spike is suppressed. Additionally, the model also accounts for synaptic delays that determine when a spike emitted by a presynaptic afferent arrives at the postsynaptic cell. This simple model no longer involves the computation of dynamical states but is simply a function that describes a neuron's output given a set of input signals.

The McCulloch-Pitts neuron laid the foundations for the development of the general analog neuron model that has evolved to the standard building block of ANNs. Notably, the modern definition is still very close to the original formulation and mainly generalizes some of the specific assumptions made by McCulloch and Pitts. Figure 2.6 provides an overview of the model's building blocks. The main simplification of common analog neuron models compared to spiking neuron models is that they do not have an internal

state which “memorizes” past inputs over a period of time.³ The dynamics equations of the LIF neuron are replaced by an *activation function* $f : \mathbb{R} \mapsto \mathbb{R}$ that is applied to the weighted sum of input signals x_i to compute the neuron’s output. There is also a fixed input bias b that can be interpreted as an additional synapse with weight w_0 and constant input 1. The activation function f fully determines the neuron’s computational properties and can in principle be chosen arbitrarily. A small set of examples is depicted in Figure 2.6. In the original McCulloch-Pitts cell, output signals were generated by a binary step function in order to mimic the spikes of biological neurons. Over time, however, the simple binary threshold logic was replaced by activation functions with continuous output that can be interpreted as a spike rate. Moreover, the modern model no longer contains inhibitory synapses that suppress all outputs. Instead, the weights w_i can assume both positive and negative real numbers with the former representing excitatory and the latter inhibitory synapses.

Both the LIF neuron model and analog neuron models laid the foundations for new fields of research that emerged in the following decades. While the first still remains one of the most important tools in computational neuroscience and is today complemented by models specialized for certain phenomena or describing dynamics at the molecular level, the latter has been refined and extended for many different applications in machine learning. The following two sections will highlight the main developments in neural modeling in the respective fields and shed light on both the similarities and the differences that emerged over time.

2.3 From Perceptrons to Deep Neural Networks

The neuron model by McCulloch and Pitts has become one of the arguably most important tools of modern AI and machine learning. Even more groundbreaking was the new paradigm that was put forward along with it. Differently from most neuroscientific research at that time, which was mainly focused on the identification, description and modeling of neural tissue, the McCulloch-Pitts cell was at the core of a new theory that aimed to link the specific structure of a neural network to a well-defined computational function [32]. Although limited to logical calculus, this theory for the first time enabled the use of neural networks as a tool for goal-directed information processing. Nevertheless, it took several decades until theory and computer technology had progressed far enough to make neural networks practicable and competitive models of computation.

³In fact, there are also non-spiking neuron models that have an internal state. In the scope of this work, however, we will always consider the most common case of stateless analog neuron models that operate in the space of real numbers \mathbb{R} .

Perceptrons

An important cornerstone for the modern definition of ANNs was laid with the development of the *Perceptron*, a neural network model that is capable of learning pattern recognition tasks, by Rosenblatt [115] in 1957. Unlike the networks considered in the work by McCulloch and Pitts, its functionality is not defined or determined through an analytical mapping between logical expressions and network topology. Like modern neural networks, it is a statistical model that is capable of learning from data. Even though Perceptrons are today commonly defined as neural networks with only a single neuron, the original model is a brain-inspired neural network architecture scheme with many different design parameters. A prototypical instantiation of that scheme with its main building blocks is outlined in Figure 2.7. Input patterns are mapped to cells of a *retina* layer. Like McCulloch-Pitts cells, all neurons in the model produce binary output signals. The cells of the retina randomly project onto a layer of *association cells* with both excitatory and inhibitory synaptic connections. If the total input of a neuron in the association layer exceeds a threshold θ , it in turn generates a binary output signal that is propagated towards the *response units*. While connections between association cells and response units are always excitatory, the efficacy of the signal transmission can change as part of the learning process.

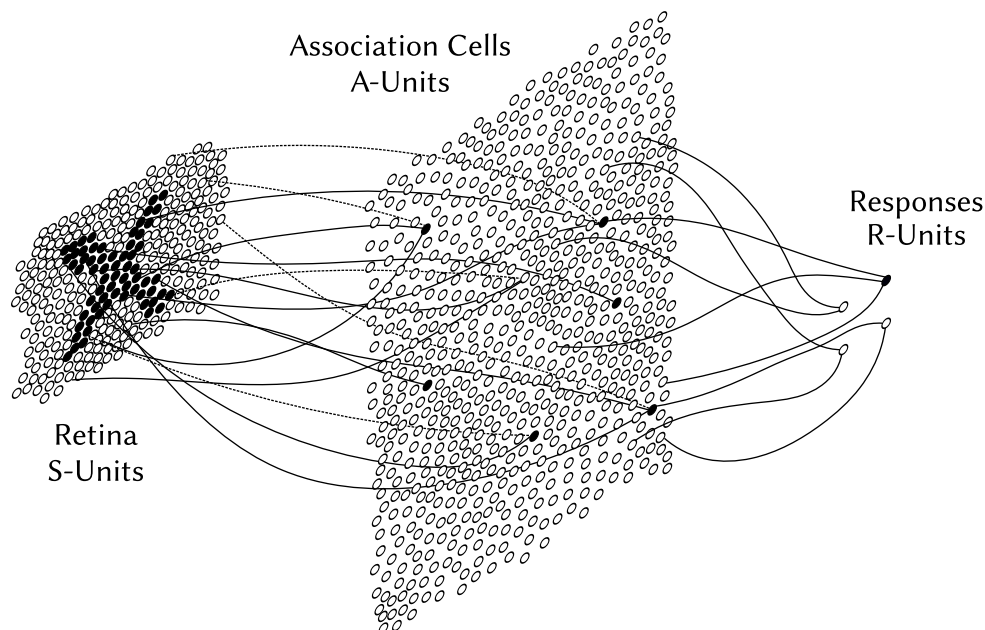


Figure 2.7: Main components of a Perceptron. Sensory input enters the system through the cells of the *retina* that are randomly connected to *association cells* by excitatory (solid lines) and inhibitory (dashed lines) connections. Association cells activate whenever the total input is above a defined threshold and stimulate the subsequent *response units* through random excitatory connections. Only a subset of the connections is shown. This specific instantiation of the Perceptron model is also called a *Photoperceptron*. The figure is based on an illustration from the report by Rosenblatt [116].

It is important to note that the Perceptron model outlined above is just one example from a whole spectrum of different variants envisioned by Rosenblatt [116]. Since it was designed to process visual input, it is called a *Photoperceptron*. Remarkably, it was already mentioned in the original report that this type is suited for processing static input patterns. A Perceptron capable of processing temporal input sequences such as audio signals would then analogously be called a *Phonoperceptron* [115]. The distinction between static and sequential input data is still one of the most important criteria for the design of a neural network architecture today. This is also true for *dataset augmentation* methods that generate additional training samples by applying different types of transformations to the input data [117]. Initial experiments with the Perceptron were concerned with rather simple tasks such as the discrimination of two letters with the actual dataset being stored as dot images on punch cards. To ensure that the trained system recognized the actual pattern rather than artifacts related to its specific input representation, the samples were modified by a set of simple transformations [118].

A third pioneering contribution that emerged from the development of the Perceptron was the construction of the *Mark I Perceptron* depicted in Figure 2.8, a device that was specifically designed according to the principles of the theory put forward by Rosenblatt [119]. Unlike a computer program, the machine did not simulate the Perceptron model but instead *emulated* it with dedicated electrical circuitry, which makes it one of the first implementations of a *neuromorphic processor*. Already at that time, the main motivation for building such a device was the insufficient scalability of standard computers to large network models [118]. *Neuromorphic engineering* is now an established area of research and has gained considerable momentum in recent years.

The Perceptron Learning Rule and the XOR Problem

While many of the concepts realized in the Perceptron were considerably ahead of their time, the underlying theory was still in its infancy. This led Minsky and Papert to conduct a fundamental study on the Perceptron's computational capabilities [11]. The contribution of their work was twofold. First, they provided a formal and compact formulation of the Perceptron model that abstracts even more from biological detail than Rosenblatt's original version. In particular, their revised definition introduces synaptic weights as model parameters, making it the basis of the modern standard analog neuron model from Figure 2.6. Given an input vector x_i the model output \hat{y}_i for a weight vector w_k is computed as follows:

$$\hat{y}_i = \theta(w_k x_i) = \theta\left(\sum_{j=0}^n w_k^j x_i^j\right) \quad (2.6)$$

All inputs of the neuron are scaled by their corresponding synaptic weights and summed. θ is similar to the binary step function from Figure 2.6 except that the output value for negative arguments is not 0 but -1. The model learns by updating its weights according to the following learning rule (adapted from [120]):

$$w_{k+1} = w_k - \eta(\hat{y}_i - y_i)x_i \quad (2.7)$$

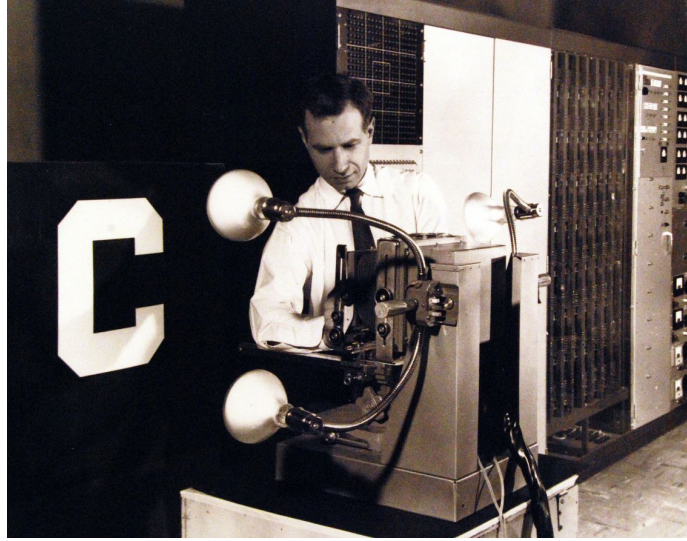


Figure 2.8: Photo of the Mark I Perceptron. The arrangement of processing elements in the device follows the schema from Figure 2.7: Input patterns of the S-units are recorded by the camera shown on the left and are then forwarded to the A-units that are located at the vertical black panel in the back. The output display of the response units is located on the rightmost part of the machine [119]. Photo courtesy of the National Museum of the U.S. Navy (public domain).

The equation above is the modern formulation of the Perceptron learning rule. w_k denotes the neuron's current set of weights, y_i the expected output for input x_i and \hat{y}_i the actual prediction of the model. η is a learning rate that controls the magnitude of the weight update. The output of the Perceptron is binary with $y_i, \hat{y}_i \in \{-1, +1\}$. Correct predictions with $\hat{y}_i = y_i$ therefore leave the weight vector w_k unchanged. Only classification errors yield a weight update that is proportional to x_i and η . These updates are applied directly and individually for every input sample, which makes the Perceptron algorithm an *online* learning rule. Closely related to this is ADALINE by Bernard Widrow, another early neural network model that was specifically designed for being implemented as a physical device [121, 122]. Both its neuron model and the learning rule are almost identical to the Perceptron with the exception that neural output is not binarized through a threshold function but directly returned as an analog value.

The second contribution by Minsky and Papert was a principled analysis of the classes of problems that can be solved by the Perceptron. Even though they introduced an extensive mathematical framework for their study, the result with the most lasting impact is surprisingly evident: The Perceptron can only classify datasets that are *linearly separable*. In particular, it cannot correctly classify the simple XOR dataset from Figure 2.9. The data set contains only four sample points in 2D space with labels that are determined by the values of the XOR function at their respective coordinates. The decision boundary x_b spanned by the Perceptron is a line that separates the two subspaces with positively and negatively classified data points, respectively:

$$0 = \theta(w_k x_b) = w_k x_b \quad (2.8)$$

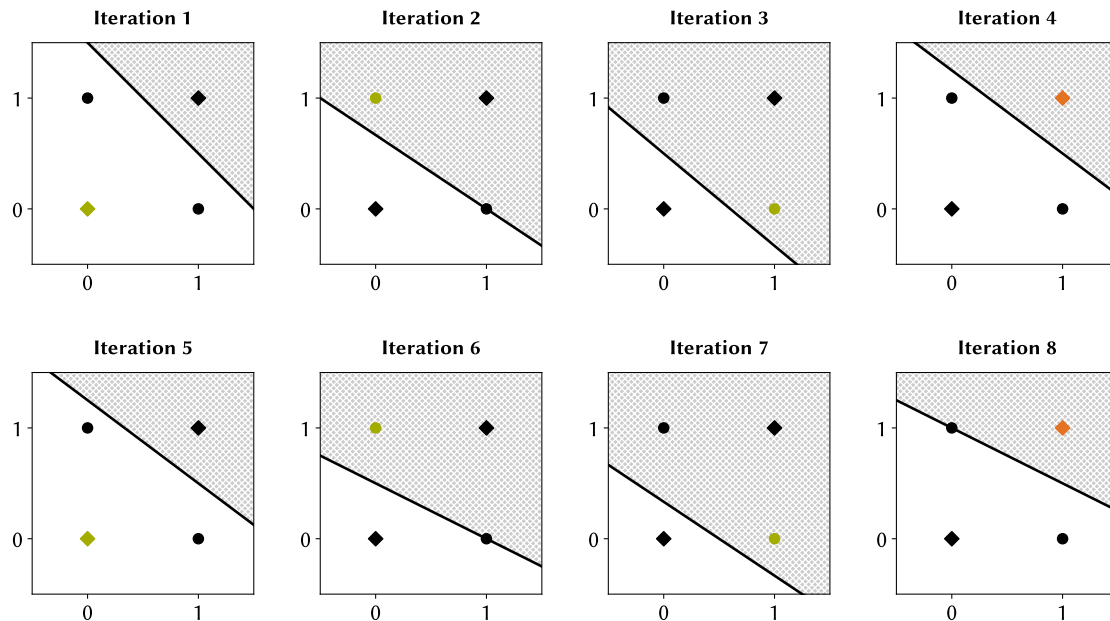


Figure 2.9: Classification of the XOR dataset with the Perceptron. Every plot above corresponds to an iteration of the algorithm. Positive and negative samples of the dataset are printed as circles and rectangles, respectively. Colors highlight the sample for which the last weight update was computed. Green indicates a correct classification and red an error. The black line indicates the decision boundary. Samples in the shaded area classified as +1. As one can easily see, the dataset is not linearly separable, which is why the decision boundary continuously changes without converging.

Already by mere visual inspection, one can directly see that the two classes of the dataset are not linearly separable, which means that they cannot be separated by linear decision boundary. As a result, the classifier learned by the Perceptron does not converge but continues to jump between the two classes. One can prove that the learning rule from Equation 2.7 always converges for linearly separable datasets [120]. The finding that the model is not even capable of learning to classify such a simple problem was one of the reasons for a drastic decline in neural network research at the onset of the first *AI winter* in 1969.

Early Neural Networks beyond the Perceptron: Hopfield Networks and Self-Organizing Maps

When the Perceptron was first presented in the popular press, it was called the “embryo of an electronic computer” [123] with an “electronic ‘brain’ [that] teaches itself” and that would one day be able to “walk, talk, see, write, reproduce itself and be conscious of its existence” [123, 124]. The reaction to the discouraging results of Minsky and Papert turned out to be just as inflated as these overly optimistic predictions. In the following years, *connectionism*, the study of neural networks, became highly unpopular and research

in AI shifted largely to symbolic representations, planning and logical reasoning methods. It was not before the beginning of the 1980s that the interest in neural modeling started to rise again. Within a surprisingly short time span, three important contributions were made by Hopfield, Hinton and Sejnowski, and Kohonen. Hopfield [125] introduced a completely new neural network architecture that implemented an associative memory system. Unlike the Perceptron, Hopfield networks have recurrent connections and weights can be computed analytically to store a set of memory patterns. Each pattern corresponds to an activation state of all neurons in the network. While this state can in principle be any real number, the original version of the network was built from Perceptron-like neurons with binary output. Input is fed into the network by clamping a subset of the neurons to their desired values. The activation dynamics then cause the other cells to converge to the complete pattern that is cued by the input cells and thereby implement the behavior of a content-addressable memory. Boltzmann machines are based on the same principles but have different dynamics equations that make the activation frequency of every neuron in the network correspond to the probability of a hypothesis [126]. This allows for statistical inference by setting neurons that corresponds to known facts to their observed values and for learning of latent space representations by *hidden* cells that are not connected to any input. Differently from the Perceptron, the activation of neurons is stochastic. The third influential network architecture from that time is the self-organizing map (SOM) developed by Kohonen [127]. It introduced local competition between neurons aligned on a sheet to learn topologically correct representation of input data by means of a self-organizing process.

On the Computational Power of Artificial Neural Networks

The commonality of the three examples outlined above is that, although they implement fundamentally different principles of computation, the underlying neuron models are directly derived from the original Perceptron model. This clearly highlights that the computational properties of a neural network depend more on its overall *architecture* rather than on the specific behavior of its individual neurons. The minimum requirements for a neural network architecture that is capable of performing meaningful computation in a general sense were defined with the proof of the *universal approximation theorem (UAT)* [128]: A neural network with only a single *hidden layer* of neurons between the input and output units can approximate any Borel measurable function⁴ that is defined on finite input and output spaces to any level of accuracy as long there are sufficiently many units in the hidden layer. Notably, the activation functions can be chosen arbitrarily as long as they are bounded and non-constant. For a given function, the UAT then guarantees the existence of a neural network that approximates it. However, it does not make any statement about the number of hidden units required or how to set the network's synaptic weights. What at first glance seems like a major impediment for

⁴In particular, all continuous functions are Borel measurable.

practical applications was actually already solved at the time the proof appeared. The development of the *backpropagation algorithm*, an efficient method for computing weight updates in feedforward neural networks with any number of hidden layers, had already started almost two decades before [129]. Today, it is still at the core of virtually all research in ANNs.

Brain-Inspired Pattern Recognition with the Neocognitron

With the universal approximation theorem and the backpropagation algorithm, two important ingredients for modern ANNs were readily available by the end of the 1980s. Nevertheless, machine learning research at that time was largely dominated by graphical models, kernel methods and feature engineering. For the breakthrough of neural networks, two more components were still missing: appropriate architectures and sufficient computational resources. An early contribution to the first part was the *Neocognitron* by Fukushima [130], a neural network architecture for visual pattern recognition. The synaptic connectivity of the network takes inspiration from the findings by Hubel and Wiesel [98] on the organization of the visual cortex of the cat, where populations of *simple cells* (S-cells) project onto populations of *complex cells* (C-cells). While the former respond to simple visual features such as lines at specific positions in the input image, the latter aggregate the combined information received from the S-cells to detect higher-level position invariant features in the input. In the Neocognitron, all neurons are arranged in a layered network with alternating S-layers and C-layers. Cells within a layer are organized in planes that are topographically consistent with the input image plane. The subsets of neurons from the previous layer that project to S-cells in the next layer have identical shape but are shifted across the image plane. This specific arrangement of neurons and synaptic connections makes representations in later layers increasingly abstract and enables the learning of representations that are invariant to the position of a specific pattern. The Neocognitron can therefore recognize known patterns even if they are shifted. Learning a specific pattern recognition task is, however, problematic since the model's original learning rule is unsupervised and therefore does not allow to enforce a target mapping between input and output.

Deep Neural Networks

Lecun et al. [131] refined the bioinspired architecture scheme underlying the Neocognitron in two important aspects. First, all weights of the network are trained through backpropagation. Second, synaptic connectivity is constrained in such a way that the resulting operations carried out between subsequent network layers are *discrete convolutions*, each of which extracts a specific feature from the image. Figure 2.10 shows a basic example. The convolutions act as *filters* that extract features from the input image. Every filter is defined by a matrix, the *convolution kernel*, and converts the input image to a *feature map*. As depicted in the figure, the detection of different types of features requires different kernels and therefore results in multiple feature maps. At first glance,

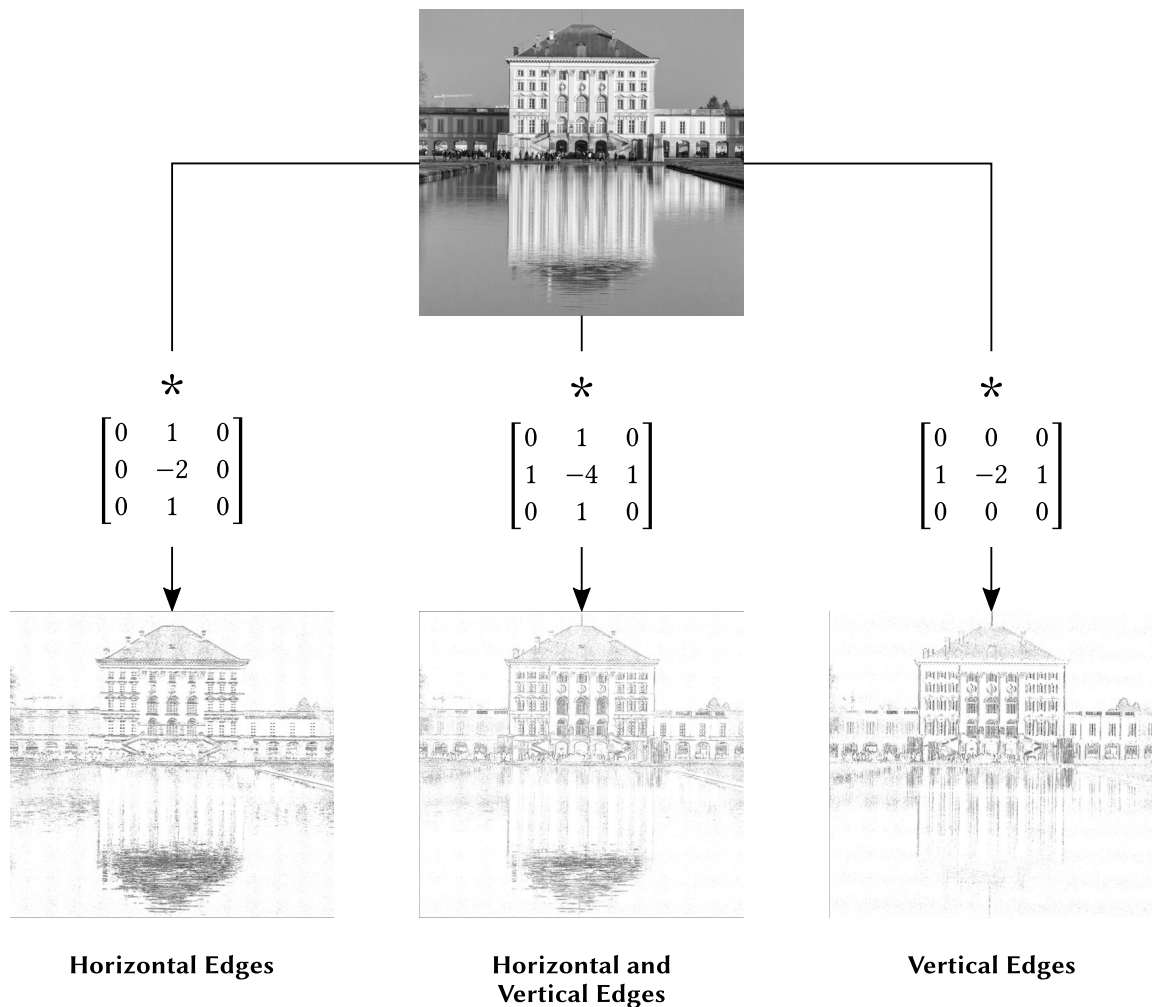


Figure 2.10: Convolution-based image filters. Basic image features such as edges can be extracted by applying a *convolution kernel* to the input image. The example above illustrates how the shape of the kernel controls the extraction of horizontal edges (*left*), vertical edges (*right*) or both horizontal and vertical edges (*middle*). Better kernels can enhance the extraction quality. The filtered images have been post-processed for better visibility, the kernels are based on [132].

this seems very similar to traditional approaches in machine learning where suitable features are engineered manually. The main innovation of CNNs is that the kernels are encoded by synaptic weights that are optimized during the training of the networks. This completely alleviates any need for feature engineering since the network learns the best matching set of features automatically. It is remarkable that already the first modern CNN successfully addressed a commercial use case. The data set that was created to train it for the recognition of handwritten digits in zip codes is called MNIST and still widely used as a baseline today.

The network developed by Lecun et al. [131] was still comparatively small in size. Scaling its architecture to a modern *deep* CNN with multiple layers of neurons was only

possible after further theoretical insights and, most importantly, sufficiently large data sets and enough compute power had become available. While many different streams of development contributed towards this goal [129, 133], there were two particularly significant milestones that set the ground for the breakthrough of deep learning. Hinton et al. [31] presented a new training procedure that enabled the learning of hierarchical feature representations in *deep belief networks (DBNs)*, a neural network architecture built from stacked restricted Boltzmann machines. Unlike modern DNNs, this method is based on *unsupervised learning* and outperformed all competing not task-tailored backpropagation-based approaches at that time on the MNIST data set. As it turned out, *pre-training* neural networks with this method in the first step made it possible to apply the backpropagation algorithm in a second step to train the network on the actual task [133]. The final breakthrough of modern DNNs was the introduction of the first deep CNN that outperformed all other competing approaches in an image recognition challenge. It was trained solely with backpropagation on a massively large data set and leveraged GPUs for accelerated computing [134]. Importantly, it was not based on a new theory or a single innovation but leveraged and integrated a tool set that had been in development for decades.

Today modern AI has become unthinkable without DNNs, which is also reflected in publication counts that have grown from about a hundred to tens of thousands papers per year. The widespread adoption of deep learning-based models and methods in virtually all fields of research and engineering has also been facilitated by efforts from industry on the development of tools and standards that make the design of DNNs both comparatively simple and affordable. In fact, the whole field is currently driven to a considerable extent by large companies such as Google, Microsoft, Facebook, Huawei, Baidu and Sony, all of which are providing their software frameworks as open source [135, 136, 137, 138, 139, 140]. The huge economic impact has even triggered the design of new hardware accelerators that are specifically optimized for the training and execution of DNNs [141, 142, 143, 144].

Remarkably, modern DNNs are still based on the same principles that were discovered decades before. It is therefore true that deep learning is to some extent inspired by the brain. Also more recent developments in the field are argued to be influenced by neuroscience research [54]. Nevertheless, the field was largely driven by technical considerations already before the breakthrough of DNNs. For example, *recurrent neural networks* with reciprocal connections could not be trained efficiently before the introduction of *long short-term memory (LSTM)*, a new analog neural network model that mitigated the *vanishing gradient problem* [145]. This model was designed with goal of optimizing the learning process without any specific consideration of biological plausibility. As a result, modern DNNs are largely based on neuroscientific research from the first half of the 20th century. Today, brain simulations offer a new perspective for the transfer of findings from neuroscience to AI.

2.4 From Spiking Neurons to Large-Scale Brain Simulations

The early work by Lapicque [111] and later Hodgkin and Huxley [110] on the mathematical modeling of biological neurons laid the foundations for modern computational neuroscience. Unlike indicated by the word *computational*, the focus of this branch of neuroscience is not necessarily on the design computer-executable models but rather on the modeling and analysis of neural dynamics [146]. However, only comparatively simple models are analytically tractable, which is why the study of more complex or detailed ones is not possible without computer simulations. As with AI, massive increases in computing power in recent decades have enabled the study of ever-larger brain models. This makes brain simulations no longer only a tool that supports theoretical studies but a promising new field of research in neuroscience.

Simulation Neuroscience

Like virtually any other field of research, modern neuroscience would be unthinkable without the computer. No other tool can manage the exploding amounts of data and no other tool enables the efficient analysis of theoretical models. The advent of big data in neuroscience has even lead to the foundation of *neuroinformatics* as a new branch of informatics that is specifically concerned with the storage and organization of neuroscientific data [147]. Neural network simulators for computational neuroscience such as *GENESIS*, *NEURON*, *NEST* and *BRIAN* have been in use since decades to enable modeling and simulation based on task-tailored standardized tools and programming interfaces [114, 148, 149, 150]. But in all these use cases, the computer only assumes a supportive role that complements and augments traditional methods, be it the acquisition of imaging data during brain scans or the analysis of theoretical models. The underlying scientific practice has largely remained unchanged for several decades. In many other fields of research and engineering, the development has been quite different. Simulations have become an indispensable cornerstone of scientific discovery in weather prediction, automotive engineering and astronomy, just to name a few [151, 152, 153]. They no longer assume a merely supportive role but have evolved to both the subject and methodology of research. This has mainly become possible as a result of the continuous progression of Moore's law, which has enabled the construction of more powerful computers that can execute more realistic simulations. Technically, simulating biological nervous systems at a high level of detail is therefore in reach.

Simulation neuroscience has been proposed as a new branch of neuroscience that bundles all research efforts on data-driven *large-scale brain simulations* [4]. At the core of these simulations are highly realistic brain models that are designed to capture as much biological detail as possible. Rather than condensing new hypotheses or experimental findings into an abstract representation, simulation neuroscience strives to synthesize a virtual simulated brain by integrating and unifying the vast amounts of largely isolated neuroscientific theories and datasets. The resulting brain simulations are then no longer

just computational models but comprehensive computer-executable implementations of the knowledge and data that were used to build them. Not only can they be used to replicate the data that they were created from but, more importantly, also to predict data that cannot be measured. However, one cannot expect that there will ever be sufficient experimental data to fully digitize a biological brain. Simulation neuroscience addresses this challenge by not modeling individual brains but instead extracting principles that describe how different types of neurons are structured and form synapses. From these principles that are implemented as statistical rules, the actual brain model can be synthesized. Their identification is one of the main scientific insights gained through simulation neuroscience [4], which is best described by Richard Feynman's famous postulate "*what I cannot create, I do not understand.*"

Besides the knowledge gained from their construction, brain simulations also bear huge methodological advantages with far-reaching impacts on neuroscience. Most importantly, they decouple the subject of study from any physical constraints. The space of exploration is no longer limited by experimental procedures or mathematical abstractions but only by computing power and storage. Brain research can thereby benefit from the same unprecedented efficiency gains that information and communications technology (ICT) already made available to many other disciplines. In this sense, brain simulations lay the foundations for the *virtualization of neuroscience*. They enable the recording of brain activity at any desired temporal and spatial resolution. Physical measurement devices become virtual probes that can be combined and distributed without any limitations and that can be added in arbitrary number at no additional cost. This makes the complete state of the simulated neural circuitry both observable and modifiable, which lays the foundations for reproducible research and fosters exchange with AI.

Large-Scale Brain Simulations

There is no standardized set of criteria a brain simulation needs to fulfill in order to qualify as a large-scale brain simulation. Simulation neuroscience only addresses a relatively small subset of the field due to its focus on data-driven methods that aim to reconstruct biological neural circuitry at an extremely high level of detail. The resulting simulations necessarily become large in scale because the number of neurons needs to approximate that of the biological modeling target. There are, however, also many other simplified large-scale models that are neuroscientifically grounded. Even though they capture less detail at the cellular level, they can still provide mechanistic explanations of emergent phenomena on the network level and on the level of observable behavior. For this reason, this work will adopt a wider scope for the term *large-scale brain simulation* that also includes these simplified models. Table 2.2 provides an overview of selected research on this topic starting with early work from the beginning of the 1980s. At that time, brain simulations were considerably constrained by limited computing power and storage. But while it is true that advances in computer science have over time enabled the creation of increasingly large models, it is important to note that the number of neurons alone is not necessarily directly related to the complexity of the model. Unlike in DNNs,

2.4 From Spiking Neurons to Large-Scale Brain Simulations

Table 2.2: Overview of selected large-scale brain simulations.

Year	Model	Brain Region	Neurons	Model Type
1982	Traub and Wong [154]	Hippocampus	1.00×10^2	Data-Driven Compartmental Neurons Static Synapses
1987	Pearson et al. [155]	Somatosensory Cortex	1.50×10^3	Simplified Analog Point Neurons Plastic Synapses
1988	Traub et al. [156]	Hippocampus	9.90×10^3	Data-Driven Compartmental Neurons Static Synapses
1989	Finkel and Edelman [157]	Visual Cortex	2.22×10^5	Simplified Analog Point Neurons Static Synapses
1997	Lumer et al. [158]	Visual Cortex and Thalamus	6.50×10^4	Simplified Spiking Point Neurons Static Synapses
2000	Medina et al. [159]	Cerebellum	1.16×10^4	Data-Driven Spiking Point Neurons Plastic Synapses
2008	Izhikevich and Edelman [160]	Cortex and Thalamus	1.00×10^6	Data-Driven Compartmental Neurons Plastic Synapses
2009	Ananthanarayanan et al. [161]	Visual Cortex and Thalamus	1.62×10^9	Simplified Spiking Point Neurons Plastic Synapses
2012	Eliasmith et al. [162]	Whole Brain	2.50×10^6	Abstract Functional Spiking Point Neurons Static Synapses
2012	Potjans and Diesmann [163]	Neocortical Microcircuit	8.00×10^4	Simplified Spiking Point Neurons Static Synapses
2012	Preissl et al. [164]	77 Cortical Regions	6.50×10^{10}	Simplified TrueNorth Neurons Static Synapses
2015	Markram et al. [59]	Neocortical Microcircuit	3.10×10^4	Data-Driven Compartmental Neurons Plastic Synapses
2019	Casali et al. [165]	Cerebellar Microcircuit	9.67×10^4	Data-Driven Spiking Point Neurons Static Synapses
2019	Antolík et al. [166]	Visual Cortex	6.50×10^4	Data-Driven Spiking Point Neurons Plastic Synapses
2020	Billeh et al. [167]	Visual Cortex	2.31×10^5	Data-Driven Compartmental Neurons Static Synapses

neuron models are very diverse, ranging from simple point neurons to complex spatial representations with hundreds of compartments. The computing power for simulating a single detailed neuron can therefore be orders of magnitude higher than for a simple one. The rightmost column of Table 2.2 summarizes basic model features that allow for a better comparison between different studies. If synaptic transmission can change over longer or shorter periods, the synapses are indicated as *plastic*. The distinction between data-driven and simplified models is less obvious. Except for simulations that explicitly focus on a data-driven workflow [59, 165, 166, 167], the classification in the table can therefore only provide a rather coarse indication of how well a simulation is aligned to biological ground-truth.

Notably, the earliest model in the table by Traub and Wong [154] was already data-driven. The overall model only contained 100 neurons but employed a detailed compartmental representation of individual cells. It was specifically designed to explain findings on neural synchronization that could not be derived from measurements alone. In comparison, the work by Finkel and Edelman [157] takes a top-down approach to investigate how a recurrent connectivity scheme called *reentry* can enable the integration of different streams of information in the visual cortex. Rather than augmenting experimental findings, the goal is to explore a new hypothesis. While the brain model is based on the structure of the primate visual system, it makes many simplifications. In particular, every neuron in the network represents a whole group of biological neurons and therefore outputs an activity average instead of individual spikes. Nevertheless, the simulation responds to certain visual illusions in the same way as humans.

Massively large simulations that cover not only smaller fractions of brain regions but larger areas up to the whole brain require simplifications and abstractions to remain computationally feasible. For example, the cortical simulation by Izhikevich and Edelman [160] is largely data-driven, based on compartmental neurons and even supports synaptic plasticity. To enable efficient simulation, it employs a highly optimized phenomenological neuron model that reproduces typical spike patterns of biological neurons without simulating the underlying electrophysiological processes [168]. Ananthanarayanan et al. [161] focus primarily on the efficient scaling of their newly developed simulation engine to simulate a model of the visual cortex and thalamus which exceeds the size of the cat's cortex. Their model captures essential aspects of cortical activity but it is based on simple LIF neurons and does not incorporate any detailed ground-truth data. This is different in the whole-brain model *Spaun* by Eliasmith et al. [162]. It is also based on LIF neurons but uses parameters from literature. Nevertheless, *Spaun* does not qualify as a data-driven model because, even though biologically constrained, it is synthesized from functional descriptions by means of a tool set called *Neural Engineering Framework* [169]. It is the only brain simulation in this list that has been designed to expose actual observable behavior and can solve basic tasks like image recognition or counting. The largest model in terms of the number of neurons in Table 2.2 was published by Preissl et al. [164] and also was created to demonstrate the scalability of the simulation on a high performance computer. The underlying neuron model is even simpler than the LIF model and was also implemented in hardware on IBM's TrueNorth chip [170].

The starting point for modern large-scale data-driven brain simulations was marked by the synthesis of a highly detailed model of the neocortical microcircuit by Markram et al. [59] in the BBP. While the total number of neurons in the model at first glance seems comparatively slow, the simulation required the full capacity of a high performance computer due to the high level of detail at which each cell was modeled. Even though it was not engineered to reproduce any specific neural activity, the final model generated the same firing statistics observed in experiments. More recent efforts in this direction by Billeh et al. [167] have yielded a data-driven brain simulation of the mouse primary visual cortex. Notably, the authors created two models with detailed compartmental neurons and point neurons, respectively, and could prove that both types of models reproduced network activity similarly well, which confirms that point neurons are an adequate level of modeling in the scope of this work. Antolík et al. [166] even focused exclusively on building a realistic data-driven model of the cat visual cortex solely from simple spiking point neurons.

The different large-scale modeling efforts summarized in Table 2.2 are highly diverse. Nevertheless, they also share two important common goals. First, there is a clear focus on modeling cortical networks, especially the visual cortex. Only a few studies implement simulations of the hippocampus and the cerebellum. Furthermore, there is also work on modeling the spinal cord that is, however, only loosely related to brain simulations and therefore not included in the table [171]. The same applies to extremely detailed simulations of single neurons [172, 173] and the OpenWorm project, which aims to provide a complete simulation of the nematode *Caenorhabditis elegans* including its nervous system with 302 neurons [174, 175]. Second, many of the presented studies use the developed models to explore the influence of parameters on the resulting dynamic properties of the neural circuitry. This exploration is completely impossible in experiments and less informative in traditional reduced models. Even though the data produced by brain simulations is just as complex and diverse as the measurements recorded with traditional methods, the possibility of changing parameters and observing the resulting changes opens up a completely new perspective for neuroscientific research.

Neuromorphic Computing

The origins of *neuromorphic engineering* date back to Mead [176], whose work laid the foundations for the design of computing devices that “emulate the organization and function of nervous systems” [177]. It is motivated by the unique computational properties of biological brains that are unmatched by any traditional microprocessor developed so far. Even though computers based on the von Neumann architecture outperform the brain in some applications and with respect to some metrics, they are highly optimized for a narrow set of features at the cost of versatility. Neuromorphic processors take inspiration from the functional organization of the brain and are optimized for the efficient simulation of SNNs. While this idea is not new, it has gained considerable momentum in recent years with support of the HBP and due to growing interest in industry. Currently available systems are still at prototype stage but have reached a level of maturity that allows for

using them in productive research. In particular, recent designs support network models of practically relevant sizes, which makes them suitable for both AI and large-scale brain simulations.

Most neuromorphic chips are manufactured with the same CMOS fabrication processes that are also used for standard von Neumann microprocessors. This makes it obvious that it is impossible to directly copy molecular processes from the brain to create a neuromorphic circuit. Instead, it is required to determine a meaningful level of abstraction that is compatible with the features and constraints of CMOS technology. The natural starting point common to all neuromorphic processor architectures are the state update equations of spiking neuron models such as those of the LIF neuron from Section 2.2. Depending on how these equations are realized on the chip, it is possible to distinguish two main classes of neuromorphic systems:

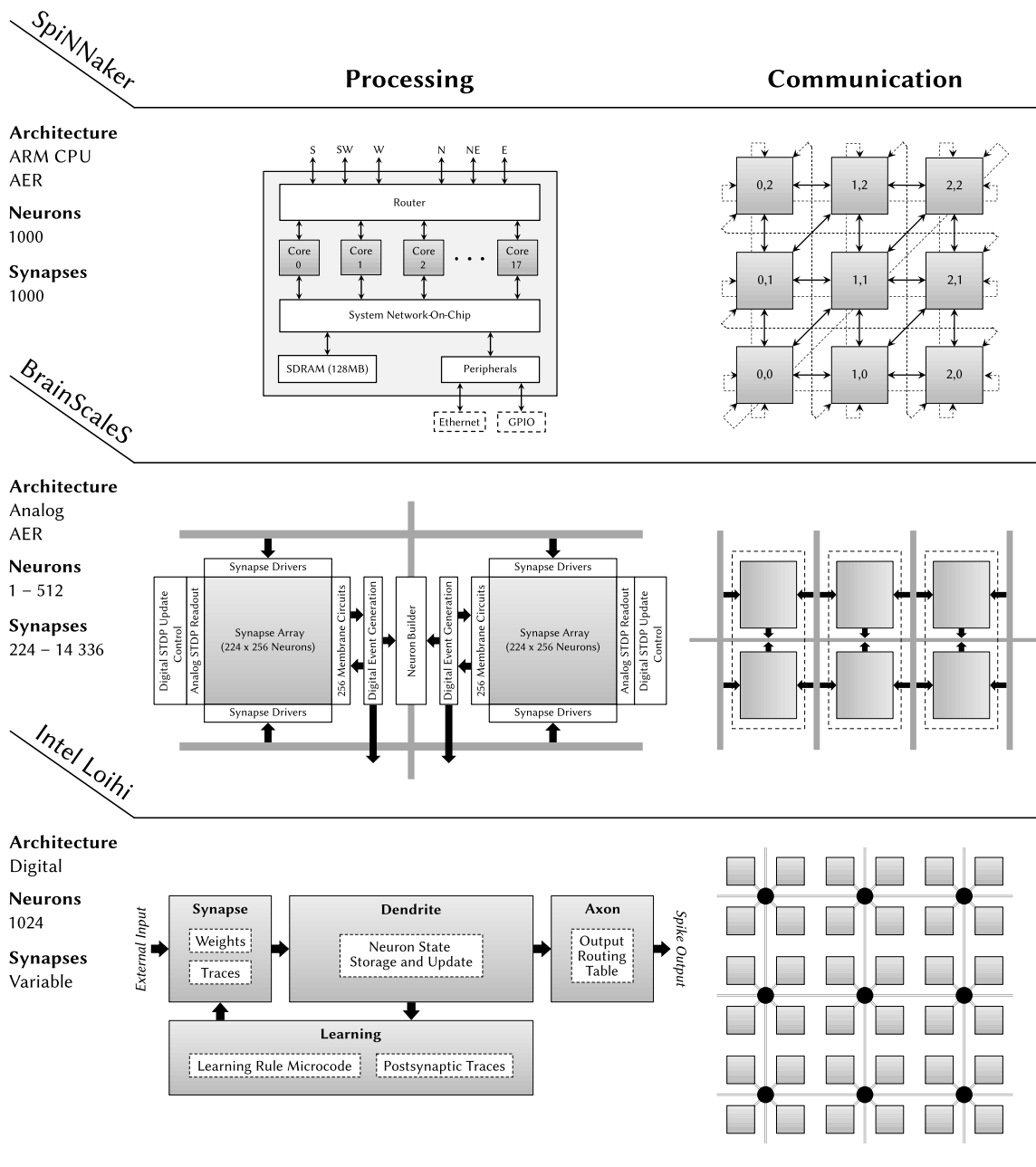
- *Analog Neuromorphic Processors*: The functionally equivalent electrical circuit of the neuron model is realized physically on the chip. Capacitances, voltages, current flows, etc. are directly represented by corresponding components on the chip, which is why the neural network is not simulated but *emulated*.
- *Digital Neuromorphic Processors*: The dynamics of the neuron model are simulated by digital circuitry with the design space ranging from optimized von Neumann processors to custom architectures.

Analog designs can operate extremely fast and efficiently since they can fully leverage the electrical properties of the circuitry on the chip. At the same time, however, these properties also impose constraints. For example, the emulation speed is fully determined by the laws of electrodynamics and, like in every analog system, there is noise. Some limitations can be mitigated by combining the analog circuitry with additional digital control logic. This approach is very common for realizing synaptic connectivity since direct connections between neurons like in the brain are not a viable design choice. Technically, dense connectivity would be extremely hard or even impossible to realize with CMOS technology. Practically, a fixed connectivity pattern would confine users to a very limited set of network architectures and applications. Most neuromorphic chips therefore share connections between neurons through *time multiplexing*, which is possible because electrical circuits can operate at much higher frequencies than biological neural tissue. In addition, the synaptic connectivity becomes freely configurable, which makes it possible to support many different types of network architectures. In summary, the use of dedicated circuits to compute or emulate neural dynamics and optimized interconnects for synaptic transmission are the two key characteristics of both analog and digital neuromorphic processors.

The multitude of implementation choices for neuromorphic systems opens up a vast design space that is reflected by the huge diversity of available architectures. Three prominent neuromorphic processors are listed in Table 2.3. *SpiNNaker* and *BrainScaleS* are developed in the HBP whereas *Loihi* is a research chip by Intel. Common to all three of them is that they have been made available to larger communities. Even though they

2.4 From Spiking Neurons to Large-Scale Brain Simulations

Table 2.3: Schematic diagrams of the neuromorphic chip designs SpiNNaker, BrainScaleS and Intel Loihi. For the former two, the first column depicts the functional layout of a single chip and the right column the layout of a system comprised of multiple chips. For Loihi, the two columns correspond to an individual core and a full chip, respectively. Neuron numbers correspond to maximum values for BrainScaleS and Loihi. In SpiNNaker, they depend on the neuron and network models. SpiNNaker and BrainScaleS transmit spikes based on the AER scheme. The drawings are partly adapted from Walter et al. [61] and are based on the corresponding original publications [62, 178, 179, 180].



are research-focused systems, they support large-scale neural network models and can be programmed flexibly through high-level programming interfaces. Each of them represents a unique approach to neuromorphic computing.

SpiNNaker is arguably the most conventional among them. Each chip is comprised of 18 von Neumann processor cores that are based on the widespread ARM architecture [180]. Neural dynamics are therefore not emulated like on fully neuromorphic designs with custom circuitry but implemented as a simulation program. The speedup is gained through a custom mesh-based routing system that is optimized for asynchronous spike-based communication and highly scalable from a single chip to up to 2^{16} chips [181]. All nodes of a SpiNNaker system are organized in mesh of toroidal topology in which each chip can only communicate with its direct neighbors as visualized in Table 2.3. Since the actual neural computation runs on the ARM cores of the individual chips, it is possible to implement many different types of neuron models and learning rules. Another special feature is SpiNNaker's redundant design. Out of the 18 processor cores, one is reserved as a spare. Similarly, additional diagonal connections in the routing mesh can compensate for defect links. In the HBP, a large-scale system with 28 800 chips has been set up [182]. In general, SpiNNaker is able to simulate neural network models in *biological real-time*, which means that the temporal scale of neural dynamics is identical to that in the brain. However, the actual execution speed is highly dependent on the model. For example, van Albada et al. [183] simulated a version of the cortical column model by Potjans and Diesmann [163] from Table 2.2 with 80 000 neurons on 217 chips at a speed 20 times lower than biological real-time.

A fundamentally different system design compared to that of SpiNNaker is realized in the BrainScaleS system [179, 184, 185]. It is a hybrid neuromorphic architecture with analog neuron circuitry and digital signal transmission. The system's core element is the HICANN (High Input Count Analog Neural Network) chip that can host up to 512 analog neurons and runs at a speed 10^4 times faster than biological real-time. Unique to BrainScaleS is its *wafer-scale* integration. Individual chips are not cut and individually packaged but instead remain on a wafer with a total of 384 units. Chip defects can be addressed by reconfiguring the system appropriately to exclude malfunctioning circuits. The BrainScaleS system built in the HBP is comprised of a total of 20 wafers [182]. This in principle not only enables support for large-scale models but also the coverage of extended timescales due to the considerable speed-up. However, the increased performance brought by physical emulation of neuron states also entails less flexibility. It is not possible to change the neuron model other than by adjusting the parameters of the built-in adaptive exponential integrate-and-fire model [186]. Similarly, while the architecture supports learning, the dynamics of synaptic updates are largely determined by the hardware. Users can therefore only configure variations of the chip's trace-based learning mechanism. Especially the latter point will be addressed in the successor system BrainScaleS-2 [187].

Loihi by Intel in many ways combines different aspects of SpiNNaker and BrainScaleS. The architecture has an asynchronous design with dedicated digital circuits for a LIF neuron model [62]. A single chip has 128 cores, each of which can host 1024 LIF neurons. Loihi is highly scalable and supports systems with up to 4096 cores per chip and up

to 16 384 chips. The fact that it is the most recent architecture in Table 2.3 becomes especially evident in its flexibility. Even though the underlying neuron model is hard-wired like in BrainScaleS, it is possible to specify multicompartmental neurons. Similarly, even though learning is based on spike traces, Loihi offers many trace variables and additional synaptic tags that can be combined by the user to specify learning rules. Three additional von Neumann cores allow for further customization. An important feature of the overall architecture is its determinism. In SpiNNaker and BrainScaleS, multiple simulation runs of the same network with identical input may yield different results due to the nondeterministic routing system and noise in the analog circuitry, respectively. Loihi enforces determinism by means of a synchronization mechanism that can also be leveraged to adjust the simulation speed.

The brief overview above only introduces the main features of three of the most relevant neuromorphic architectures. Not only are there many other neuromorphic processors, each of them also incorporates a huge set of features enabled by unique design choices [61, 188]. It is therefore challenging to make direct comparisons, especially since there are no standardized benchmarks [189]. More importantly, performance is not the only criterion when deciding for or against a particular neuromorphic computing platform. For example, BrainScaleS is not suited for most applications in robotics due to the highly accelerated simulation speed that does not fit the timescales in the physical world. Moreover, the energy consumption of the overall system is too high for embedded applications, which partly results from the fact that it has been designed with data center applications in mind, where it can deliver high throughput at high efficiency. It is interesting to note that also standard GPUs have been demonstrated to deliver competitive performance for SNN simulations [190].

Workloads with high throughput that are typical for data centers do typically not occur in the field where robots need to process sensory input to adjust their motion or autonomous cars need to capture relevant objects on the road to compute optimal trajectories. In these applications, low energy consumption and short latencies are key requirements that are not met by traditional von Neumann architectures. Especially SpiNNaker and Loihi are very well suited for robotics due to their very low total power draw, the easy scalability to the needs of the application and the ability to run in biological real-time. This can even hold in the unfavorable case where ANNs are simulated on a neuromorphic processor. As Blouw et al. [64] have demonstrated for a keyword spotting task, Loihi is extremely efficient for network inference real-time applications where only one sample at a time is processed. In comparison, the performance of modern GPUs can only be leveraged when a large number of samples is processed in parallel, which is typically not the case in robotic perception tasks. Neuromorphic processors therefore bear a huge potential for real-time sensor data processing. Major challenges for widespread use in practical applications are the definition of appropriate standardized programming interfaces and the development of SNN models that match the performance and versatility of DNNs. Particularly with respect to the latter point, efforts are underway for all three systems presented in this section [191, 192, 193].

3

Foundations of Neurorobotics

Brain simulations establish a computational link between the anatomical and physiological properties of nervous tissue and the patterns of neural activity that emerge from them. In this sense, they are a tool to analyze and explain the brain's *structure* and do not provide any direct insight into its *function*, i.e. the cognitive processes and externally observable behavior that neural activity gives rise to. Linking structure to function cannot be achieved with neural modeling and brain simulations alone, for they can only account for *what* happens in the brain, but not *why*. Explaining the latter is only possible if both the causes and effects of neural activity are realistic and fully accessible. More concretely, this implies that the brain must be studied in its natural "habitat": embedded in a *body* that is *situated* in an *environment*. Only then one can expect neural activity to be realistic and interpretable in terms of concrete perceptions and actions. This elementary insight is widely known as *embodiment* and has been investigated in neuroscience, cognitive science and robotics alike. So far, the lack of a principled scientific methodology and corresponding tools has hindered substantial progress, rendering embodiment largely a hypothesis rather than a proven principle. In recent years, however, technological innovations in computer science and robotics brought new momentum to the field by enabling the development of *neurorobotics* as a scientific discipline with exclusive focus on embodied systems.

The early beginnings of neurorobotics date back to the mid 20th century and thus even before the foundation of modern robotics [194]. At that time, computer science was also still in its infancy. The first neurorobots were simple mobile devices driven by vacuum tubes that allowed them to follow sensory cues and even learn simple stimulus response patterns [195, 196]. Later systems benefited from the rapid advances in computing technology and started to embed detailed brain simulations in physical robots that were able to interact with their environments [197]. Nevertheless, major progress in the field

has so far been held back by the considerable financial and technical effort required to set up a neurorobotics system. High costs for customized robots result in the construction of proprietary one-of-a-kind experimental setups, make modifications difficult and results irreproducible. This situation is about to change with a new state-of-the-art methodology put forward by the Neurorobotics Subproject of the HBP: Like in many other fields before, neurorobotics research can be substantially accelerated through *virtualization*, which means replacing physical robots by robot simulations.

This chapter introduces the theoretical foundations of neurorobotics and the architecture of the HBP Neurorobotics Platform. Section 3.1 discusses the basic features of embodied systems and outlines how they can serve cognitive processes such as perception and learning. The insights gained from this discussion form the basis for the formal definitions of *neurorobotics* and *neurorobotics experiments* in Section 3.2. Finally, Section 3.3 provides an overview of how virtual neurorobotics experiments can be modeled in the NRP.

3.1 Theoretical Foundations of Embodied Systems

In the context of this work, *embodiment* describes an artificial or biological organism with a control system (the brain) that has an interface (the body) through which it can act in and interact with the outside world (its environment) where it is situated. Figure 3.1 provides a graphical summary. The central tenet of embodied systems research is that the body is not simply a mere housing for the brain with sensors and actuators but actually plays a constitutive role in its functioning. This is programmatically brought to the point by the title “*How the body shapes the way we think*” of the seminal book on embodied intelligence by Pfeifer and Bongard [198]. Embodiment stands in stark contrast to traditional views in AI, robotics and also neuroscience, where sensory experience and physical interaction are typically treated as arbitrarily interchangeable input and output channels. Embodiment makes the body an active component of cognitive processing by leveraging the specific structure induced by body morphology and sensor response properties in the sensorimotor space. The prerequisite for this structure to emerge is that the brain can interact with the environment through the body in a closed loop of *perception*, *cognition* and *action*.

Closed-Loop Perception – Cognition – Action

The processing sequence of perception – cognition – action (PCA) has evolved to a standard model in AI [23] and is also a central element of *cybernetics*, that Wiener [199] defines as the study of “control and communication in the animal and the machine.” Figure 3.2 puts all involved stages into the context of a basic robot grasping task. From a conceptual point of view, the loop is comprised of an inner part, the cognitive model *M*, and an outer part, the environment. They are connected to each other through perception and action, which are realized by the body’s sensors and actuators, respectively.

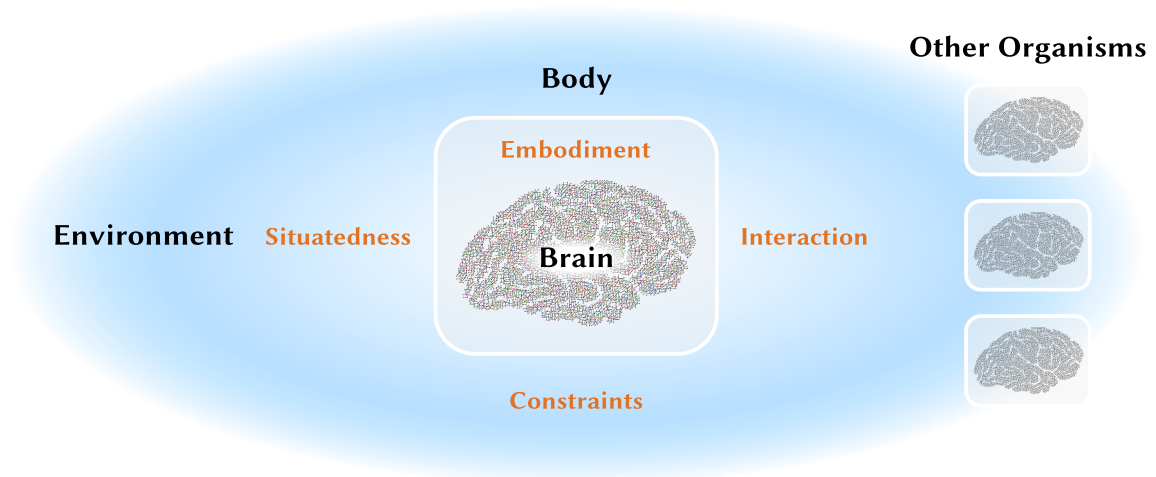


Figure 3.1: Embedding of an embodied system in its environment. An embodied artificial or biological organism is comprised of a control system, the *brain*, and its *body*, through which it can *interact* with the environment where it is *situated* by perceiving and acting. The interaction with the environment is *constrained* by the specific morphology of the body.

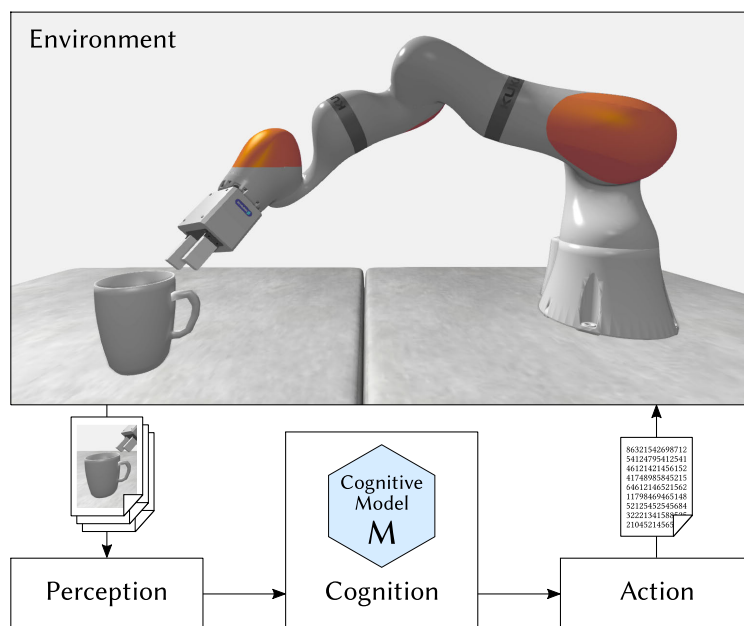


Figure 3.2: Schematic description of how an embodied system interacts with its environment in a closed PCA loop. Independently of the actual realization of the cognitive model M , which may be a biological brain or any type artificial brain model, the core of embodiment is the direct interaction of that model with the environment through its body. By means of this body, the output of M gets transformed into an action in the environment that in turn determines the following perceptual input captured by the sensors.

In general, M can take any arbitrary form ranging from a symbolic rule-based system to a biological brain. The framework can consequently be used to describe artificial and biological systems alike. In the scope of this work, M will implicitly always refer to some form of neural brain model, be it an artificial neural network or a large-scale brain simulation. Special consideration also needs to be given to the perception stage, which by definition refers to “the conscious experience of sensory stimulation” [200]. This is in line with traditional system architectures in AI with a strict separation between the conversion of external stimuli to internal representations and reasoning processes operating on them. Since such separation is not possible in the brain, we limit the perception stage to sensing, i.e. the reception and encoding of sensory stimuli. It is important to note that this decision by no means renders the perception stage a simple mechanical process. In fact, sensing has always been an independent research topic within neuroscience. The action stage will be analogously confined to the actuation of the body including low-level control mechanisms that are not directly related to the brain.

By shifting all higher cognitive functions into M , the main interaction in the PCA loop occurs between the brain model and the environment. Unlike in traditional systems following the PCA scheme, the perception and action stages of embodied systems do not decouple cognition from the body and its environment but instead establish a tightly coupled *closed loop* between them. It is exactly this closed loop that is essential to embodied systems, which is why we will also refer to them as *closed-loop systems*. In the example from Figure 3.2, the goal of the robot is to grasp the cup next to it on the table. Assuming that the gripper is equipped with sensors for contact forces, the robot can sense when it is touching the cup and try to lift it. Readings from its joint torque sensors provide additional information on whether the grasp was successful and how much torque is required for lifting. When operating in a closed loop, the actions of the robot, i.e. the movements of its joints, cause changes in the sensor readings that in turn inform the selection of the next action. The system is therefore closely coupled to its environment and changes therein are an essential factor in the information processing of the cognitive model M . Importantly, this coupling would be drastically weakened if low-level sensorimotor information was filtered out in the perception and action stages. Wiener [199] summarizes the importance of closed-loop feedback as follows:

“The central nervous system no longer appears as a self-contained organ, receiving inputs from the senses and discharging into the muscles. On the contrary, some of its most characteristic activities are explicable only as circular processes, emerging from the nervous system into the muscles, and re-entering the nervous system through the sense organs, whether they be proprioceptors or organs of the special senses.”

Morphological Computation

A fully closed PCA loop is the most important prerequisite for embodied systems but only provides the technical basis. In order for the body to be leveraged as an active part

of cognitive processing, a corresponding computational principle must be acting on top. This was addressed with the introduction of the concept of *morphological computation*, which expresses that cognitive processing, here referred to as computation, is not limited to the brain but also includes the body. Pfeifer et al. [201] have popularized the concept as follows:

“By ‘morphological computation’ we mean that certain processes are performed by the body that otherwise would have to be performed by the brain.”

The term has found widespread adoption in many different areas of robotics. But as Müller and Hoffmann [202] argue, it still misses a rigorous definition, especially since the phenomena attributed to it typically only have very little in common with computation at all. Based on a study of different robot systems that incorporate some form of morphological computation, they identify three distinct realizations of the concept:

- Actual morphological computation
- Morphology facilitating perception
- Morphology facilitating control

Actual morphological computation as defined by Müller and Hoffmann [202] is mainly relevant in terms of computational theory and refers to cases where a body acts as a form of computer in the literal sense of the word. Hauser et al. [203] propose a theoretical framework for the formal analysis of the computational capabilities that models compliant bodies as generic mass-spring systems. This abstraction enables them to show that if the dynamic response of a body to an input signal is sufficiently rich, any nonlinear time-invariant filter with fading memory can be computed from a weighted linear readout that is derived from the current system state, e.g. the lengths of all springs in a mass-spring system. It is important to note that the computation carried out does not depend on a specific form of movement pattern but rather leverages the dynamic effects of the disturbances caused by impinging input signals. Even simple input stimuli can elicit highly non-linear spatiotemporal response patterns in compliant mechanical systems. Akin to a feature space projection, the body projects input into a complex state space representation by means of its dynamics. Rigid mechanical structures such as industrial robots that are designed with the goal of minimizing the complexity of their body dynamics do not exhibit such rich response properties and therefore are not capable of performing morphological computation.

In a broader perspective, the specific form of morphological computation described above can be seen as an instance of *reservoir computing*, a concept that was originally developed as a design and training method for neural networks [204]. Figure 3.3 provides a schematic overview of the main components. A common feature of all reservoir computing systems is the tripartite structure comprised of simple input and output layers that interface with the reservoir. The readout layer typically computes a weighted linear combination of the reservoir states. While other methods are possible, the actual main distinction

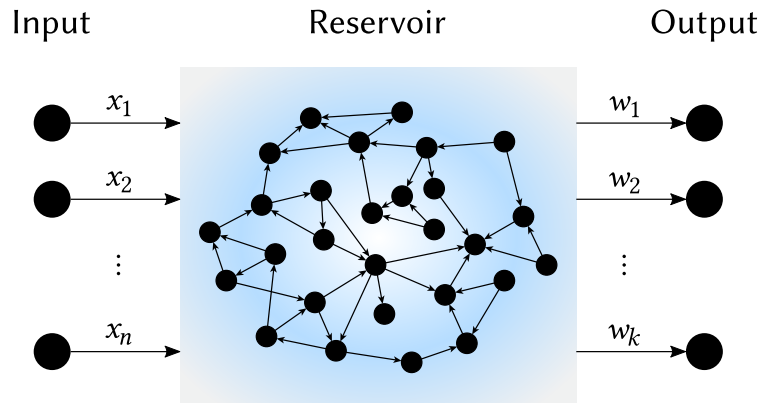


Figure 3.3: Overview of a typical reservoir computing system. Input data are fed into the reservoir, a dynamical system that can be realized as a computational model (e.g. a neural network) or a physical system (e.g. a mass-spring system). In turn, the reservoir produces a complex nonlinear response pattern that is sampled by a set of readout units as a weighted sum. The set of weights w_i that produces the desired mapping from input to output is commonly determined automatically by a learning algorithm. Adapted from [63] by permission from Springer Nature Customer Service Centre GmbH, ©2016.

between different approaches lies in the realization of the reservoir. *Echo state networks* and *liquid state machines*, the two pioneering reservoir computing methods, are based on reservoirs of random recurrent ANNs or SNNs with fixed weights [205, 206]. The neurons and their connectivity serve no specific purpose other than producing rich spatiotemporal patterns in response to input signals and can therefore be replaced by any other dynamical system such as the body of a living creature or a robot. Further research on *physical reservoir computing* also investigated reservoirs made from electrical circuits, photonic systems or even biological neuron cultures on microelectrode arrays [207].

Morphological computation in the sense defined by Müller and Hoffmann [202] takes no advantage of body morphology other than exploiting its dynamics. It does neither leverage how a body is adapted to a task or an environment nor does it bear any direct relation to the actual motion of the body, which is in direct contradiction with the core idea of embodiment, where the body actively interfaces the brain with its environment. In the context of this work, morphological computation in the sense of reservoir computing will be mainly considered as a conceptual proof that the body can support cognitive processing in the brain. The focus will be on how morphological computation can facilitate perception and control. Three examples from biology illustrate best how closely physical processes in the body and cognitive processes in the brain are intertwined with each other:

- In the spiral-shaped *cochlea* of the human inner ear, different regions of the basilar membrane along the inner spiral are sensitive to decreasingly lower sound frequencies with increasing distance from the entry point. An incoming sound signal is thereby spatially separated into its different component frequencies, a phenomenon called *tonotopy* [208]. This operation is akin to a Fourier transform that is automatically performed as a result of the physical structure of the cochlea and would otherwise need to be implemented by neurons in the brain.

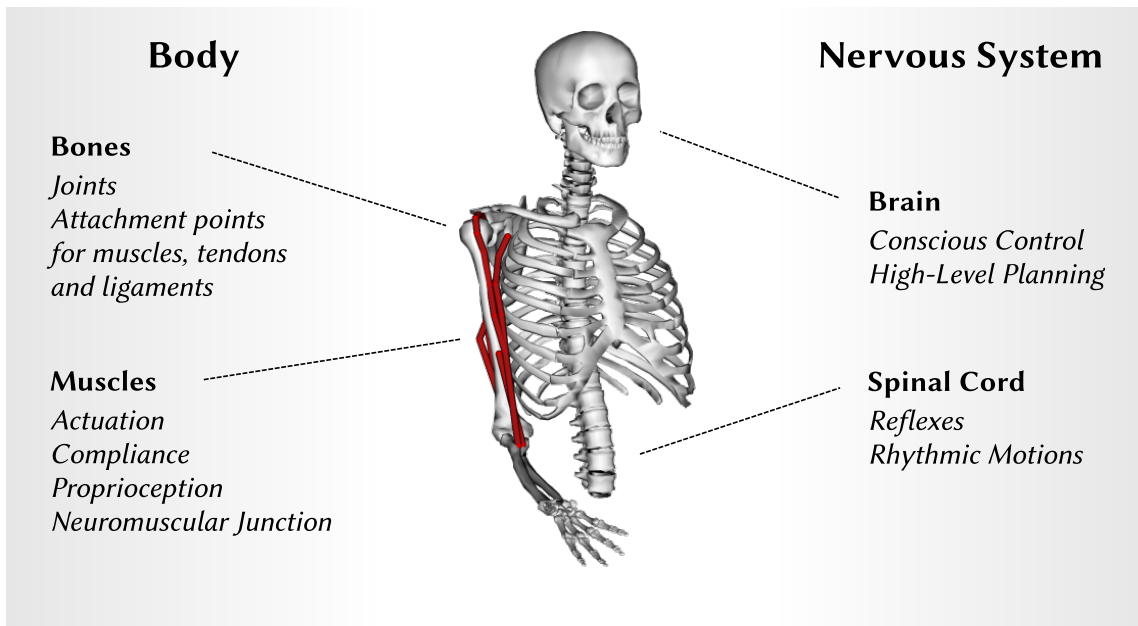


Figure 3.4: Main components of the neuromusculoskeletal system of the human body. Control is distributed across both the nervous system and the body. The latter is comprised of a rigid skeleton that is antagonistically actuated by a total of around 640 muscles [209]. Unique body properties such as the compliance of muscular tissue and the softness of the skin enable morphological computation. The rendering of the skeleton was generated with OpenSim [210].

- The specific placement of sensory receptors on the human body shapes the *structure of sensory input* (e.g. the eyes of the human head always move at the same time) and thereby reduces the complexity of the input space. At the same time, sensory streams from different modalities such as vision, hearing, touch and proprioception are correlated by the coupling through the body (e.g. grasping an object induces visual, tactile and proprioceptive sensory feedback). Sensory input is therefore not only actively modulated by movements of the body as described by the PCA loop but also synchronized, which further facilitates processing in the brain [198].
- The *neuromusculoskeletal system* of humans depicted in Figure 3.4 is inherently compliant. This is because unlike in a typical industrial robot that is fully rigid, the bodies of vertebrates are comprised of both soft and rigid materials. The rigid skeleton acts as a scaffold that ensures overall structural integrity. Its individual bones are held together by the tissue of muscles, tendons and ligaments, which account for compliance. The combination of both makes tasks such as walking considerably simpler since inaccuracies in motor control signals or uneven ground are automatically compensated by the body without the need for any active regulation.

Especially the construction of artificial musculoskeletal systems has become a major research topic in robotics. Typically, the resulting robots are based on fully custom designs that are only built a single time as a proof of concept. This is why

musculoskeletal robots are comparatively rare and expensive even though material costs are often lower than for industrial robots. A notable exception is *MYROBOTICS*, a toolkit for building musculoskeletal robots from a small set of basic mechanical and electronic components [211]. Its core feature are artificial muscles that are built from spring-mounted electric drives and coil up a thin tendon-like wire. Movements of the tendon transmit the force to the actual joint. Based on this principle, the toolkit enables the rapid construction of a huge set of morphologies optimized for different tasks. This flexibility is important when building robots that can leverage morphological computation. As pointed out by Pfeifer et al. [201], the optimization of the body for a specific task in a specific environment trades off efficiency against versatility. Passive dynamic walking machines, for example, require no control system at all but can only perform a single task in the environment they have been designed for.

Enactive Cognition

At the beginning of this section, it was emphasized that the PCA loop is an essential component of embodied systems. Morphological computation naturally operates in this loop as the body mediates between the brain and the environment. Its main contribution therefore lies in shaping sensory experience and physical interaction. And while it is in some cases even possible to provide a quantitative account of the computational contribution of the body in the sense that the controllers operating in the corresponding robot can be simplified, morphological computation still operates primarily on a low level of abstraction. At first glance, this seems to preclude the direct participation of embodiment in any cognitive processes beyond low-level perception and motor control. The theory of *enactive cognition* put forward by Varela et al. resolves this issue by laying the theoretical foundations that link seemingly low-level embodiment to high-level cognition [212]:

“In a nutshell, the enactive approach consists of two points: (1) perception consists in perceptually guided action and (2) cognitive structures emerge from the recurrent sensorimotor patterns that enable action to be perceptually guided.”

While morphological computation addresses the mechanistic effects of embodiment on perception and action, i.e. an organism’s sensorimotor experience, enactivism aims to explain phenomenologically how this experience forms the basis of cognitive processing. Even though enactive cognition does not directly rely on any properties of body morphology, it is critically shaped by an organism’s unique embodiment.

To fully understand the definition quoted above, it is necessary to know about its epistemological background. The origins of enaction lie in the concept of *autopoiesis* conceived by Varela et al. [213] as a definition of living organisms.¹ Autopoietic systems are characterized by an organization that is “continually self-reproducing” [215]. Put

¹As outlined by Wilson and Foglia [214], different notions of enaction have developed over time with distinct views on the role of autopoiesis. In the context of this work, we will only refer to the *autopoietic enactivism* of Varela et al. [212].

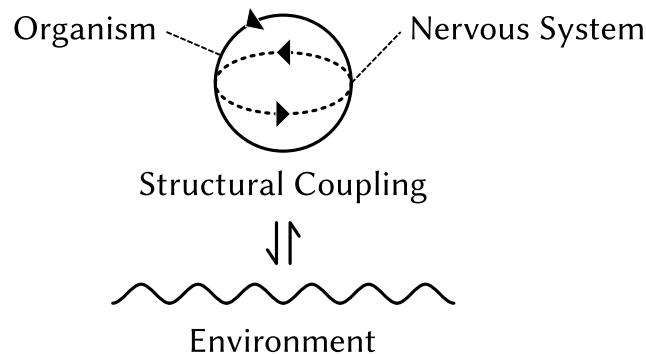


Figure 3.5: Schema of the three constituents of an enactive cognitive system and their interrelations. An *organism* is an autonomous system of components linked by processes that maintain themselves. Its *nervous system* expands the repertoire of interactions in which it can engage with the *environment* to which it is *structurally coupled*. The description and the figure are adapted and extended from [215].

differently, this means that the components and processes an *autopoietic unity* is comprised of operate to maintain themselves intact and distinct from their environment. Any system with an organization that differs from this basic structure is *allopoietic*. Whereas autopoietic systems are necessarily autonomous, allopoietic systems are not. This becomes immediately evident at the example of a standard industrial robot operating at an assembly line: Even though it automatically executes a certain task, it requires programming (not autonomous) and regular maintenance (not autopoietic). It is important to note that both practical considerations and technical constraints limit the utility of the concept of autopoiesis in the design of artificial cognitive systems. Its actual value for this domain lies in its contribution to the development of the theory of enactive cognition.

The starting point of enactive cognition are the reciprocal interactions between an autopoietic system and its environment. Figure 3.5 provides a schematic overview of all entities involved. The autopoietic system is indicated by the two loops at the top, the outer of which symbolizes the property of self-reproduction that maintains the system's structure as a unity. It is *structurally coupled* with the environment. This means that, in the terminology of Maturana and Varela [215], perturbations by the system will cause changes in the environment and vice versa. It is essential that enactivism does not view these changes as goal-directed. They are mere reactions arising from the structural coupling and their nature is, in the case of the autopoietic system, determined by the system's concrete realization in terms of sensors, actuators and possibly a nervous system. If present, the latter vastly increases the repertoire of sensorimotor response patterns. Over time, the history of interactions with the environment shapes the autopoietic system's individual development, its *ontogeny*. This process of adaptation also includes changes in the nervous system that can, for example, be interpreted as learning.

With the concept of structural coupling in place, the definition of enactive cognition from the beginning of this section can now be made more concrete: Perception is an active process that generates patterns of sensorimotor experience in a closed PCA loop established through structural coupling. These patterns are the foundation of cognitive processing and, consequently, cognition in an enactive system is solely based on its unique

individual sensorimotor experience that it collects as it interacts with the environment. It has no general and objective internal representation of the world to which perceptions are mapped. The complete abandonment of representations in favor of unique individual experience is the central hypothesis of enactivism [212]. It entails that it is only possible to understand the brain – or rather an individual brain – by taking into account its full ontogenetic history. Starting from a phylogenetically predetermined configuration at birth, it is shaped through embodied interaction with its structurally coupled environment. Changes in the body or the environment will therefore be reflected by changes in the brain, resulting in a fully individual cognitive framework. These properties make enactivism not only an explanation for inter-individual differences of brain structure and for the effects of external factors on individual development. They also make a strong case for neurorobotics.

The subjective nature of environmental perception had already been proposed before Varela et al. [212] by zoologist Jakob von Uexküll in his work *Umwelt und Innenwelt der Tiere* with the definition of the *Funktionskreis*, arguably one of the first appearances of the PCA loop [216]. In his terminology, this circle connects a living creature's *environment* (*Umwelt*) with its *inner world* (*Innenwelt*) through a *perception world* (*Merkwelt*) and an *effects world* (*Wirkungswelt*). Crucially, the environment is a completely subjective experience of the organism that only reflects a very specific subset of all its surroundings [216]:

“Was uns als außenstehenden Beobachtern der Umwelt der Tiere am meisten auffällt, ist die Tatsache, daß sie nur von Dingen erfüllt ist, die diesem speziellen Tier allein angehören.”

Uexküll [216] further argues that the constituents of both *Merkwelt* and *Wirkungswelt* are tied to objects in the environment and it is these objects to which living creatures adapt. This concept has become known as *affordances* [217], which can be loosely described as opportunities to engage in interaction. In enactivism, it is reflected by the shift away from propositional knowledge (“knowledge *that*”) to commonsense knowledge (“knowledge *how*”) that is required in unstructured real-world environments as described in Chapter 1 [212].

The principles laid down by enactivism have had significant influence on robotics. Arguably most famous is the *subsumption architecture* conceived by Brooks [10, 21] in an effort to realize autonomous robots without any internal representation of the world. The method proved very successful for a range of simple behaviors and even laid the foundations for the vacuum cleaning robot *Roomba* [218], one of first robots to be deployed at large scale outside of manufacturing plants. Robotics is also a key tool for an objective quantitative analysis of the effects of enaction. Lungarella et al. [219] derived a set of different information theoretic measures to quantify the effects of embodied interaction on sensorimotor data streams. In a series of robotics experiments, they were able to demonstrate that closed-loop embodied interaction with the environment not only induces statistical regularities but also affects the flow of information in the PCA loop [220, 221]. Importantly, not only learning but also changes in body morphology were shown to have a direct effect on the structure of the data.

3.2 A Modern Definition of Neurorobotics

The first neurorobotics systems – and arguably even the first modern robots in general – were the “*tortoises*” by neurophysiologist William Grey Walter [195]. While their design was of striking simplicity with a control system of only two vacuum tubes, they realized a closed PCA loop with two motors and two sensors for light and touch. The advancement of both robotics and neuroscience over time allowed the construction of increasingly advanced neurorobotics systems. But the diversity of brain research and the strong focus on traditional algorithmic methods in robotics led neurorobotics to remain in a rather small niche with many independent strands of exploration ranging from neural translations of existing algorithms to artificial experimental studies of ontogenetic neural development. This situation has only changed recently when neurorobotics research gained huge momentum as part of the HBP. With the development of the NRP, it is now for the first time possible to formulate a common scientific objective and a principled methodology for neurorobotics.

Directions of Research in Neurorobotics

First mentions of neurorobotics date back at least to 1987, even though the actual term used at that time was *neurobotics* [222]. Nevertheless, it was already introduced in the context of studying sensorimotor loops at the intersection of robotics and neuroscience. Over time, the scope has widened and neurorobotics today also refers to research on human-machine interfaces, prosthetics and rehabilitation robotics [223]. While these medical applications will not be considered any further, it is important to note that the contributions of this work can still be of relevance for them. To aid the formulation of a principled scientific objective for neurorobotics, the following paragraphs provide an overview of different directions of research.

Neural Implementations of Traditional Algorithms from Robotics A mainly computationally motivated approach to neurorobotics is to implement existing robotics algorithms with SNNs. Based on the *Neural Engineering Framework* [169], Menon et al. [224] modeled an analytically derived operational space force controller [225] for a robot with three degrees of freedom as a SNN running on the neuromorphic processor *Neurogrid* [226] in real-time. Bouganis and Shanahan [227] proposed a SNN controller for the arm of a humanoid robot which was inspired by the standard kinematics equations from robotics. However, unlike in the work by Menon et al. [224], the network learned the desired behavior completely autonomously via *spike timing-dependent plasticity (STDP)* [228]. The field of robot path planning was addressed by the work of Ponulak and Hopfield [229] who conceived a neural implementation of the wavefront planner [230] based on spiking neurons and STDP. An important advantage of converting traditional robotics algorithms to functionally equivalent neural networks is that existing systems and architectures can be easily reused. On the downside, the additional insight gained by this method is limited to the implementation level, which is mainly relevant for practical applications of

neuromorphic computing. As can be seen in the examples from this paragraph, effects of embodiment or a closed PCA loop are generally neglected.

Conceptual Studies Fully leveraging the potential of neurorobotics therefore requires the development new models and experiments. To reduce the complexity involved in setting up a brain model, a robot and defining the connection between them, many studies rely on simplified experimental setups that are limited to the minimum set of components which is necessary to study the targeted phenomenon. Pioneering work in this direction was done by Braitenberg [231] who constructed a series of simple *vehicles* as thought experiments. Each of them is comprised of a small set of sensors and actuators that are connected to a simple nervous systems of only a few neurons. Despite this simplicity, the emerging behaviors are considerably diverse. Mobile robot platforms similar to the vehicles by Braitenberg [231] have become a very common tool in neurorobotics. For example, Floreano and Mattiussi [232] synthesized a controller for collision-free visually guided navigation by adapting a network of spiking neurons with an evolutionary algorithm. Di Paolo [233] trained a small SNN with a combination of genetic programming and STDP to implement phototaxis in simulated robots. Florian [234] applied a similar approach to a larger network in order to train a robot to push objects around. Obstacle avoidance based on a spiking neural controller was investigated by Wang et al. [235]. Arena et al. [236] also studied obstacle avoidance but additionally conducted experiments on different goal-directed navigation tasks. Starting from a predefined set of system responses, the authors implemented a STDP-based classical conditioning rule which enabled the robot to learn the desired behavior. Brandi et al. [237] investigated classical conditioning in a mixed-reality setup with a physical robot navigating on a virtual reality track. Their goal was not to develop a robot controller but to test a hypothesis on how the cerebellum can learn sequences of actions through classical conditioning. However, the employed cerebellum model was purely computational and did not include any neural network simulation. Operant conditioning for neurorobotics was addressed by recent work of Cyr et al. [238]. The authors designed a minimal neural circuit consisting of only three neurons and evaluated it successfully both in simulations and on a physical robot.

Embodied Cognitive Models The reduced models applied in the studies from the last paragraph limit both the types of cognitive phenomena that can be modeled and the degree of neuroscientific detail. Pioneering work on more comprehensive neurorobotics systems was the brain-inspired neural network for trajectory control developed by Miyamoto et al. [239]. Their model had a cerebellar component that was capable of learning the inverse dynamics of an industrial robot manipulator online during the execution of a trajectory. The trained model generalized to different speeds and other trajectories. Reeke et al. [197] introduced the *Darwin Series of Brain-Based Devices* to study more high-level cognitive phenomena [194]. *Darwin I* and *Darwin II*, the first two of these automata, were neural network models designed for pattern recognition and categorization. The first actual neurorobotics system was *Darwin III*. It was comprised of a simulated robot arm

with four joints, an actively actuated “eye” and sensors for touch and kinesthesia. The underlying neural architecture had different subsystems for oculomotor control, reaching and touch. Sensori stimuli from the different modalities were integrated by the neural model to categorize the objects the robot interacted with. Later versions of *Darwin* were physical mobile robot platforms that, for example, were capable of stimulus tracking and object sorting [240], learned selectivity for visual patterns [241] or developed place fields in a model of the hippocampus [242]. Cox and Krichmar [243] investigated how neuromodulation can be used as a means of associating behaviors to perceived stimuli on a mobile robot platform. An even simpler robot with two degrees of freedom for balancing a tray was the basis of the research by Probst et al. [244]. The goal was to move the tray in order to keep a ball placed on top of it in a desired region. Most notably, control was based on reservoir computing in a spiking neural network with parameters and connectivity similar to layer IV of the cortex. Tani [245] proposed *Multiple Timescales Recurrent Neural Networks* to represent both high-level plans and low-level action primitives in hierarchically organized artificial neural networks. Interestingly, the hierarchy emerged through different timescales assigned to the individual components of the networks. In a series of neurorobotics experiments, the author investigated the formation of functional hierarchies during multimodal interaction with the environment. Finally, Conradt et al. [246, 247] implemented neuromorphic stimulus tracking and learning by demonstration on a mobile robot platform that was equipped with a SpiNNaker system.

Biomimetic Robot Control The connection of brain models to robots is especially promising in the field of biomimetic robotics, where morphology and actuation are designed to closely match the bodies of living creatures. Neurorobotics not only enables research on brain-derived control systems for non-rigid and highly redundant mechanical structures, but it can also benefit from the realistic embodiment realized by these types of robots. Influential work in this direction was published by Ijspeert et al. [248] who developed a robot with a salamander-like morphology. Locomotion was implemented by a *central pattern generator (CPG)* model [249] of the spinal cord that was based on coupled nonlinear oscillators. The rhythmic control output generated by this model drove the limbs of the salamander robot and reproduced biological locomotion patterns. In earlier work, the spinal cord model was realized as an ANN that was parameterized with a genetic algorithm [250]. CPGs were also applied to study the crawling motions of infants [251]. Klein and Lewis [252] developed a biomimetic bipedal walking machine and controlled it with a SNN which implemented both cyclic and reflexive behavior in a reflex-driven CPG architecture. Research with focus on the exact modeling and simulation of the musculoskeletal structure of the human body with methods from robotics was published by Nakamura et al. [253]. The proposed musculoskeletal model was comprised of a musculotendon network attached to a rigid skeleton. Muscles, tendons, ligaments and cartilage were represented as wires with two or more contact points. By applying an inverse dynamics algorithm to motion capturing data, it was possible to compute muscle tensions completely non-invasively. In later work, the musculoskeletal model was

further refined and augmented by a neuromuscular network built from artificial neurons [254]. After learning to reproduce muscle tensions computed from motion capturing data, the network was successfully applied in a simulation of the patellar tendon reflex. Sreenivasa et al. [255] considered a considerably simpler musculoskeletal model with only one antagonist pair of muscles to model the human arm stretch reflex. However, differently from Murai et al. [254], the authors used spiking neurons organized in different pools to model the spinal reflex network. The neural parameters were inferred from physiological data and the results of the simulation were validated against real-world experiments.

Enactive Systems With the complete abandonment of representations in enactivism, neurorobotics is a key tool for research on enactive systems. Only when both the brain and the body can be fully observed is it possible to study how an enactive system develops while interacting with its environment. Floreano et al. [256] and Suzuki et al. [257, 258] conducted a series of experiments with physical robots on active perception to investigate the influence of embodied closed-loop perception on information processing in neural networks. Receptive field formation in a series of neural network controllers that were generated using genetic algorithms and a Hebbian online learning rule was shown to be strongly dependent on whether the robot's control output was based on the current camera input. Importantly, sampling of the visual input space was more focused when control was visually guided. A practical example of leveraging affordances is the *Differential Extrinsic Plasticity* learning rule by Der and Martius [259], which exploits the dynamics of mechanical systems to generate stable motion patterns. In an experiment with a MYOROBOTICS humanoid robot arm, the rule was capable of learning different motion patterns, whose shape depended directly on the form of interaction the robot engaged in (e.g. rotating motions when it was connected to a wheel) [260]. Remarkably, the underlying control mechanism was extremely simple and required neither complex modeling nor extensive computation, clearly highlighting the central role of the robot's embodied interaction with the environment. Closely related to enactivism is the field of *cognitive developmental robotics*, which aims to explain human cognition through the simulation of ontogenetic developmental processes [261]. This is very different from traditional approaches of modeling a fully developed system and in line with the enactive paradigm. Following this approach, Kuniyoshi and Sangawa [262] connected a simplified musculoskeletal model of an infant to a basic artificial nervous system comprised of spinal reflex circuits, CPGs and an abstract model of the cortex. In a series of synthetic experiments, the authors investigated the emergence of coordinated movement primitives from spontaneous interactions with the environment. In later work, the infant model was extended to include tactile sensors distributed over the skin [263] and a simulated nervous system of spiking neurons was used to study the formation of sensorimotor maps through simulated interaction of a fetus with its uterine environment [264, 265]. Remarkably, by varying simulation parameters, the authors demonstrated how abnormal preterm conditions can lead to degenerate sensorimotor maps. Other experiments with

the infant model include the investigation of the role of sensory constraints in the self-organization of sensorimotor behavior [266] and the analysis of the effect of development on the formation of neural circuits in the spinal cord [267]. Nagai et al. [268] proposed a model for the emergence of the mirror neuron system from sensorimotor maps. The mirror neuron system enables action recognition and is thus an important prerequisite for the development of empathy during interactive social development.

Deriving a Definition for Neurorobotics

The brief overview presented above aims at identifying relevant directions of research in neurorobotics. In-depth reviews are outside the scope of this work and have been published earlier [269, 270, 271, 272]. Moreover, there is not only a dedicated journal for neurorobotics [273] but also a technical committee in the IEEE Robotics and Automation Society [274]. It becomes evident from the huge variety of different approaches that neurorobotics research cannot be identified with a specific class of models or methods. The essence lies in the closed-loop connection of some type of brain to some type of body. A comprehensive definition needs to detail which types of brains and bodies are eligible for neurorobotics systems. Krichmar [270] proposed the following set of criteria:

“A neurorobot has the following properties:

1. It engages in a behavioral task.
2. It is situated in a real-world environment.
3. It has a means to sense environmental cues and act upon its environment.
4. Its behavior is controlled by a simulated nervous system having a design that reflects, at some level, the brain’s architecture and dynamics.”

While a system that satisfies these four properties obviously qualifies as a neurorobotics system, this definition is very constraining. First, the engagement in a behavioral task is common but not necessary. For example, the study of emergent phenomena like in the experiments by Yamada et al. [264] does not involve any goal-directed behavior. Likewise, also many other studies on enactive cognition are not covered by this definition. Second, the limitation of neurorobotics to real-world environments is too narrow. Krichmar [270] argues that “simulating an environment can introduce unwanted and unintentional biases to the model.” Even though this argument is valid and in line with Pfeifer and Bongard [198], state-of-the-art simulation tools make it increasingly less relevant. More importantly, the most advanced research topics in neurorobotics can only be investigated in simulations. This is best illustrated at the example of biological musculoskeletal systems that can only be simulated but not replicated in physical robots. Kaplan [275] suggested a less concise but also more general definition of neurorobotics that does not require engagement in a behavioral task and also includes simulations:

“At the interface of neuroscience and robotics, neurorobotics is the science and technology of embodied autonomous neural systems. Neural systems include brain-inspired algorithms (e.g. connectionist networks, artificial spiking neural nets),

computational models of biological neural networks (e.g. large-scale simulations of neural microcircuits) and actual biological systems (e.g. in vivo and in vitro neural nets). Such neural systems can be embodied in machines with mechanic, pneumatic, electromagnetic or any other forms of physical or virtual actuation. This includes robots, prosthetic, wearable systems, virtual reality environments but at also, at smaller scale, micro-machines and, at the larger scales, furniture and infrastructures. [...] The grand challenge of neurorobotics is to build a well-founded experimental science of embodiment.”

It is important to note that this definition, in contrast to the one by Krichmar [270], also includes systems such as neuroprostheses where the brain is not a simulation but actual biological tissue. Other examples include work on closed-loop control of a mobile robot based on recordings from biological neurons cultured on a high-density microelectrode arrays [276, 277].

Neither of the two definitions presented above captures the essence of current neurorobotics research. While the one by Krichmar [270] only addresses a rather narrow class of models, the one by Kaplan [275] covers the most relevant aspects of the field but seems too unspecific. We therefore propose a new definition that is motivated by the unique research opportunity of neurorobotics and grounded in the analysis of the research landscape from the last subsection:

Neurorobotics is a field of research at the intersection of robotics, neuroscience, cognitive science and AI that provides both the *theory* and the *tools* for connecting *brains* to *robotic bodies* in a closed loop of perception, cognition and action. A neurorobotics system is comprised of four components:

1. A brain, which can be any type of neural system that transforms perceptual input into motor output, ranging from simple artificial neural networks to highly detailed large-scale brain simulations and observable biological neural systems.
2. A robotic body, which can be any type of physical or simulated mechanical structure that can interact with an environment through sensors and actuators, ranging from simple mobile robot platforms to highly detailed simulations of biological musculoskeletal systems.
3. A mapping that establishes a closed loop between brain and body by specifying how perceptual input from the body is mapped to the brain and how motor output from the brain is mapped to the body.
4. A physical or simulated environment in which the system is situated.

The definition is different from the two before because it is explicitly centered around the establishment of a closed PCA loop, but does not require autonomy or leveraging

any effects of embodiment as is the case in the text by Kaplan [275]. This is not meant to imply that embodiment and enaction are not at the core of neurorobotics research. In fact, one of the main motivations of this work is to show that neurorobotics is a key tool for investigating them. However, the less restrictive formulation of the definition allows for the inclusion of the growing body of research on DNNs for robotics [278]. Furthermore, biological neural systems such as the neuron cultures investigated by Maruyama et al. [276] and Masumori et al. [277] are considered neurobotic systems but neuroprostheses are not. This is because the latter are “connected” to complete biological brains that can be neither fully inspected nor modified, which is in direct contradiction to the main goal of neurorobotics.²

Neurorobotics Experiments

With a comprehensive definition of neurorobotics in place, it is now possible to introduce *neurorobotics experiments*. The term *experiment* is motivated by the close relation of neurorobotics to neuroscience and the specific nature of closed-loop systems. Like in the case of large-scale brain simulations, the dynamics of neurorobotics systems with brain and body models of practically relevant detail and size are too complex for purely theoretical analysis. Conducting experiments to observe the resulting neural activity and its relation to behavior is therefore essential. Figure 3.6 depicts a schematic overview of all components involved in a neurorobotics experiment. At the core is a neurorobotics system which is comprised of a robot and a brain simulation. We will refer to the latter also simply as the *brain*. Both are connected to each other in a closed loop through a mapping that defines how sensory stimuli and motor commands are relayed to the brain and the robot, respectively. A second closed loop is established between the robot and the environment. Whereas the first loop is responsible for neural signaling, the second one enables physical interaction between the neurorobotics system and its surroundings. Only through the combination of both of these loops, can a closed PCA loop be established. In real-world experiments, the physical interaction loop is automatically present by the laws of physics as long as there are no artificially imposed constraints. Virtual experiments require a physics simulation that reflects the embedding in the environment at the required level of fidelity.

The development of a neurorobotics experiment encompasses five steps and the process is akin to traditional experiments in the natural sciences or in robotics. These are: *design*, *instantiation*, *setup*, *execution* and *evaluation*. Table 3.1 provides an overview. Every new type of experiment starts with the design of a general protocol that describes all of the components involved and the variables to be observed. In fact, this protocol does not only define a single experiment, but a whole class of experiments that, for

²This does, of course, not exclude the use of neurorobotics tools for research and development in neuroprosthetics. The main point is that the type of insight gained thereby is fundamentally different from actual neurorobotics experiments.

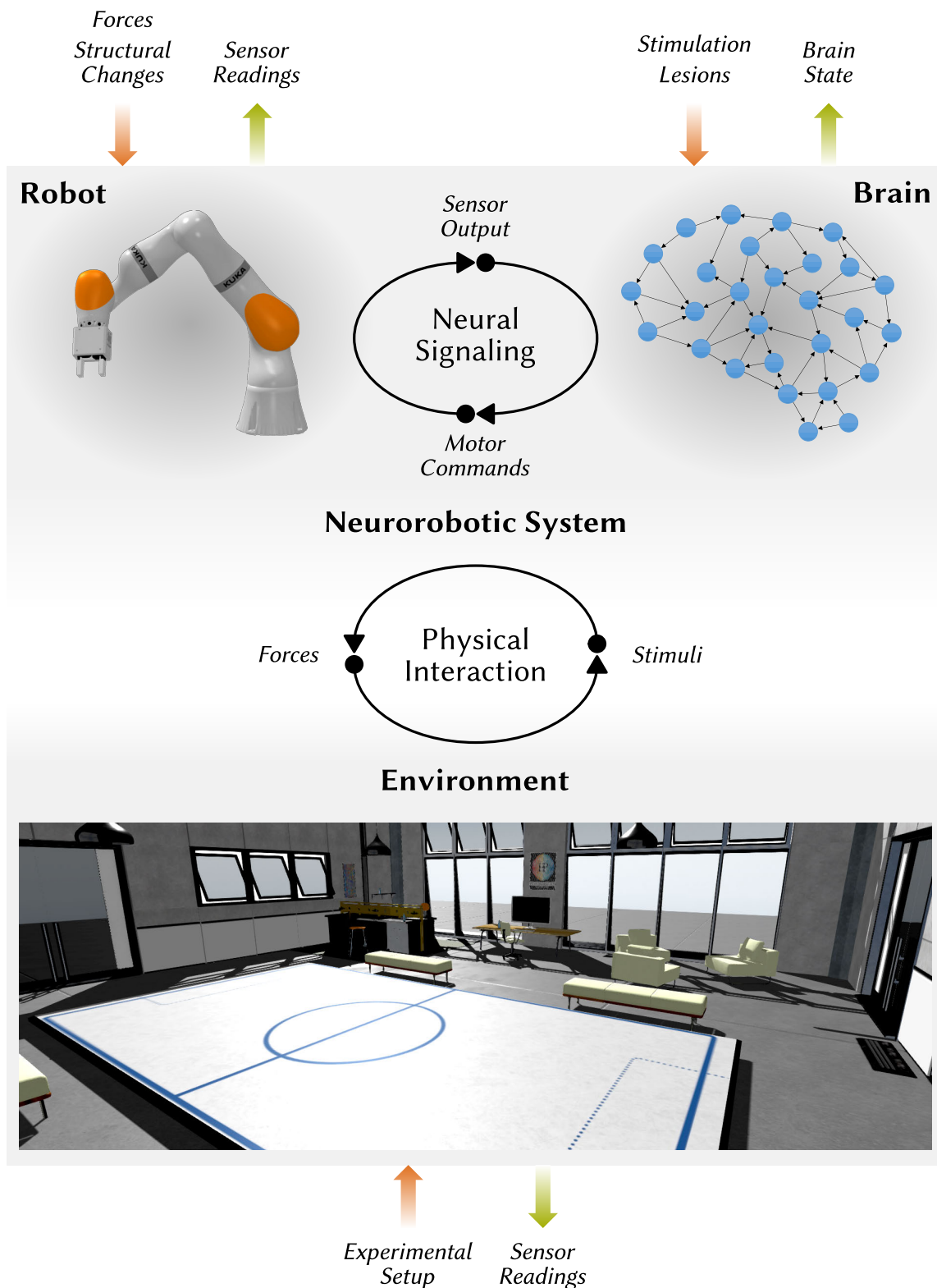


Figure 3.6: Schematic overview of a neurorobotics experiment. Designing an experiment starts with the definition of the neurorobotics system by specifying the routing of neural signals between the *robot* and the *brain*. The *environment* is comprised of all required objects external to the robot and coupled to it through physical interaction. Red and green arrows indicate parameters of the experimental protocol and measurements, respectively. Note that the schema applies equally to both simulated and physical robots.

Table 3.1: The five steps in the development of a neurorobotics experiment.

Step	Description	Example
<i>Design</i>	Specification of a general class of experiments based on a research goal, including descriptions of supported types of robots, sensors and environment layouts as well as the system architecture.	An experiment prototype for robot grasping tasks with a set of generic programming interfaces.
<i>Setup</i>	Implementation of a concrete physical or virtual experimental setup based on the design specification.	An industrial robot with a prismatic two-finger gripper is placed next to a set of objects. It contains a model of the motor cortex with a learning rule that enables it to grasp objects based on visual input from a camera.
<i>Instantiation</i>	Definition of experiment parameters.	The objects placed next to the robot are colored wooden bricks; the parameters of the cortical model are set according to data from a recent publication.
<i>Execution</i>	The experiment is executed. Depending on the design and setup, a pre-defined protocol is carried out. The user may interact with the system and visualize state parameters in real-time.	The robot arm is controlled by the cortical model to grasp one brick after another. A live visualization displays the neural activity of the cortical model.
<i>Evaluation</i>	Data recorded during experiment execution are evaluated.	Spike traces of neurons in the simulated cortical model are compared to data from the literature.

example, addresses a scientific question about grasping, navigation, vision, etc. From an engineering perspective, experiments of the same class will share a common system architecture and use similar software components. The concrete robot and brain models as well as the details of the environment are defined in the setup step, where the actual experiment is built. In the case of a physical setup, this step involves setting up the robot including auxiliary equipment, deploying all required software and preparing the brain simulation. Virtual setups are prepared by implementing or including the required models and integrating them into a physics simulation. Note that virtual setups can easily be shared with others, making conducting an experiment considerably simpler and faster. Before the experiment is ready for execution, the setup must be instantiated by setting the parameters of the robot, the brain model and the environment. In a grasping experiment, for example, this step includes preparing the object the robot should grasp. Virtual experiments make it possible to instantiate a single setup arbitrarily often in parallel and with different parameters. After instantiation, all aspects of the experiment are fully defined and the actual execution can begin. Depending on the type of study, the experiment can be interactive or fully automated according to a fixed protocol. The orange

arrows in Figure 3.6 provide examples of possible inputs that can be applied or changed during the runtime of the experiment. Data can be recorded just like in a traditional experiment for further analysis in the final evaluation phase, as indicated by the green arrows. A special feature of fully virtual experiments is that the neurorobotics system can be fully inspected at runtime. Since all components are simulated, the complete system state is transparent at any point in time, without any limitations of physical measurement devices.

3.3 The HBP Neurorobotics Platform

The HBP Neurorobotics Platform, which has already been introduced in Section 1.3, is developed as a task-tailored tool for the design and conduction of neurorobotics experiments. While there are other tools that connect brain models to some form of embodiment, none of them is as comprehensive as the NRP. For example, AI-SIMCOG [279], NeuVision [280], Simbrain [281] and SpikingLab [282] provide integrated toolkits that include both neural network simulators and a robot simulators. However, the latter support only extremely limited 2D environments without any advanced modeling of sensors, actuators and physics. The systems conceived by Gamez et al. [283, 284] and Cofer et al. [285] employ realistic simulations of the robot and its environment but are limited to specific robot models such as CRONOS and iCub or biomechanical systems, respectively. Finally, Weidel et al. [286], Voegtlin [287], and Jordan et al. [288] proposed interfaces for connecting neural simulators to robot simulators in a systematic way. But while such interfaces are at the core of every neurorobotics system, lots of additional software infrastructure is required to conduct actual neurorobotics experiments. The unique advantage of the NRP among these different approaches is that it combines high-fidelity simulations with maximum flexibility and scalability. This feature set is realized by a software architecture that has been optimized for the unique requirements of neurorobotics experiments.

System Architecture

A key motivation for the development of the NRP was to provide a tool for investigating embodied large-scale brain simulations. The realization of this specific feature entailed two design decisions that make the system architecture considerably different from other tools for neurorobotics research. First, the NRP needs to provide a comprehensive world simulation to ensure that the quality of the simulated robots and environments does not fall behind the brain simulation. Second, it must be possible to run experiments on high performance computing infrastructure that has enough resources to simulate models of the required size and at the targeted level of detail. To accommodate these requirements, the NRP is implemented as a web service. This makes it possible that experiments run in data centers that host the required computing resources while users can still directly interact with them through a web-based application. A schematic overview of the resulting system architecture with the main components and interfaces is depicted in Figure 3.7. Owing to

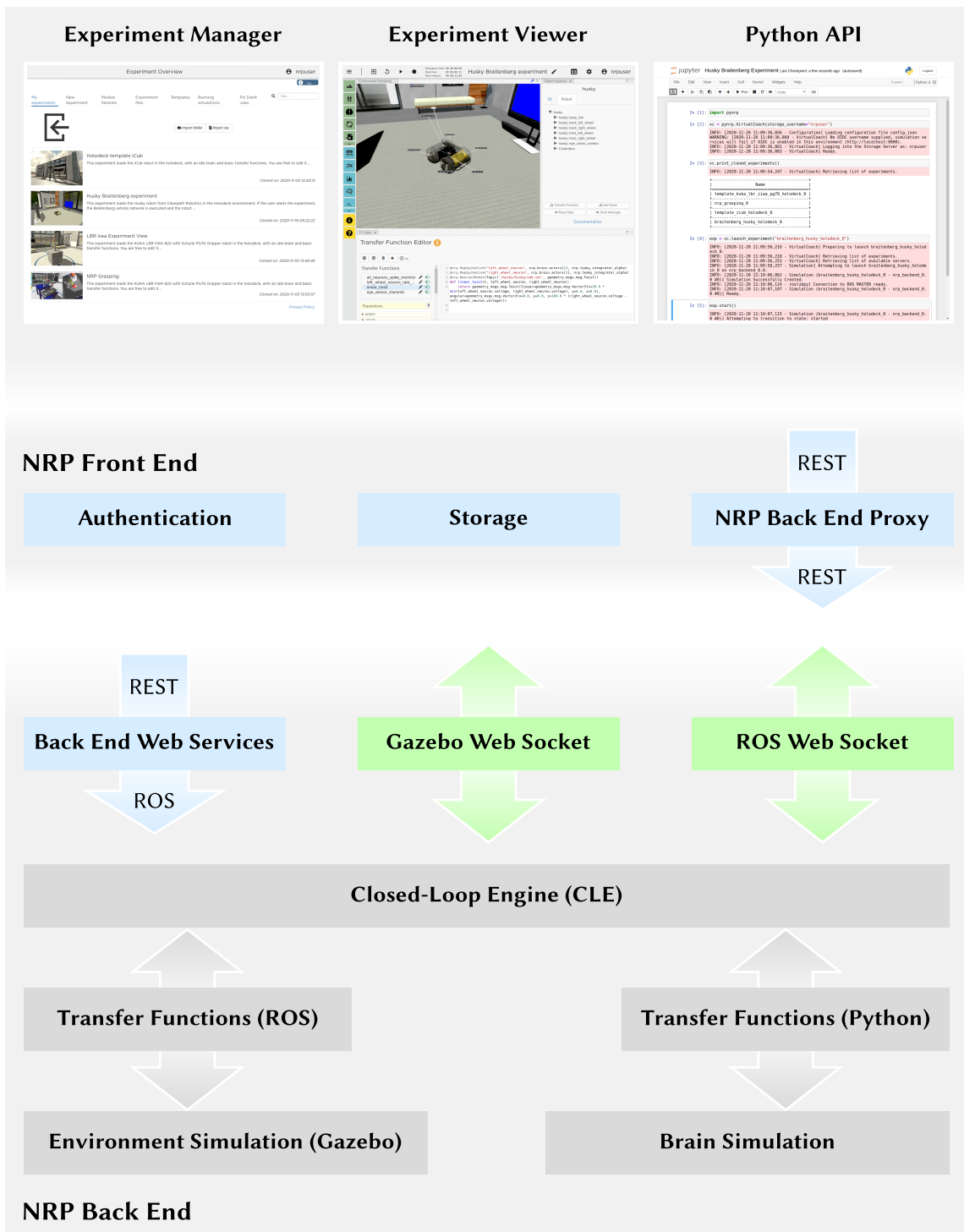


Figure 3.7: Schematic overview of the system architecture of the HBP Neurobotics Platform.

the realization as a web service, the architecture is layered according to the different types of compute loads and execution environments involved in a neurorobotics experiment running on the NRP.

NRP Back End The core of the NRP is the *NRP Back End* shown at the bottom of Figure 3.7. It hosts both the brain simulation and the environment simulation as well as a set of web-based communication and management services. The architecture is designed to support different implementations for both simulations so that users can integrate the tools that are best suited for their use cases. For example, while neural models are provided as *PyNN* [289] scripts and simulated with NEST [290] by default, there is also support for *SpiNNaker* (modeling with PyNN) [180] and *Nengo* (modeling with NEF) [291]. In Section 4.2, the mechanisms that enable this flexibility will be discussed in more detail for an NRP extension that adds support for the neuromorphic processor Intel Loihi as a brain simulator. Even though the interface to the environment simulation also supports modularity, the only tool that is currently supported is an extended version of the open-source robotics simulator *Gazebo* [292] with support for musculoskeletal simulations based on *OpenSim* [210]. In contrast to many other robotics simulators, Gazebo is an open-source project, which makes it possible that the complete NRP is released under an open-source license. The fact that the complete codebase is fully accessible is especially important for research projects where new models can only be investigated by extending the system's feature set and where large-scale simulations on high performance computers are only possible by adapting the components to match the requirements of the underlying software infrastructure.

Closed-Loop Engine Both the environment simulation and the brain simulation are external tools and, apart from some minor adaptations, not part of the NRP development. This is because the main contribution lies not in providing a simulation engine but a tool set for realizing closed-loop experiments based on a principled workflow. In this sense, the NRP acts as a middleware that connects existing simulation engines. The required application logic is implemented by the *Closed-Loop Engine (CLE)*, which was developed from the ground up in the HBP. It manages, connects and synchronizes the environment simulation and the brain simulation. This involves the launch and termination of instances of each simulator on start and stop of an experiment, as well as the exchange of data for closed-loop operation during runtime. The latter requires the translation between different interfaces and representations: Control commands and sensor readings from Gazebo are made available through the *Robot Operating System (ROS)* [293], whereas neural simulators are typically based on the Python programming language. To connect the two simulations, the CLE needs to act as a hub that translates between different communication protocols and data representations as illustrated in Figure 3.8. Both the environment simulation and the robot simulation run in parallel for a defined duration of *simulated time* Δt before they are paused. Depending on model size and complexity, the actual *simulation time* may be shorter or longer than Δt and is in particular different for both simulations. The step-wise execution of an experiment with fixed time steps

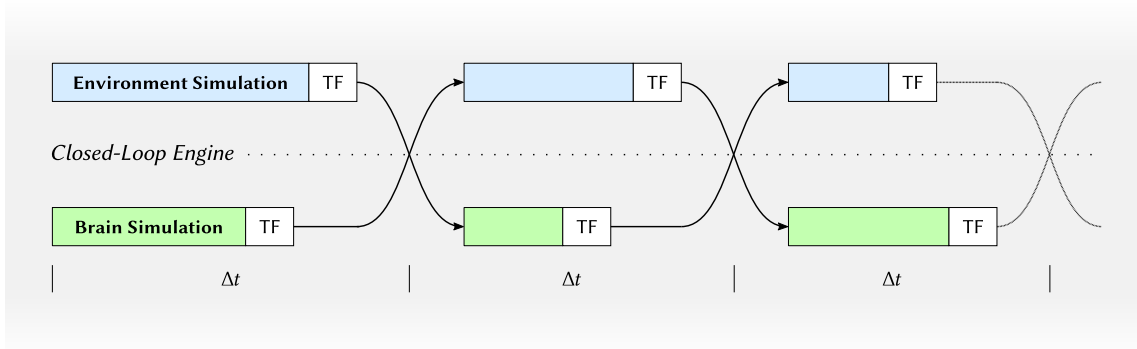


Figure 3.8: Synchronization of the environment simulation and the brain simulation by the CLE of the NRP. Each simulation runs independently until the simulated time has progressed by the global time step Δt . At the end of each step, the output of one simulation becomes the input of the other for the next time step. The encoding and decoding of data is handled by TFs.

enforces that both the brain and the environment models remain in *temporal synchrony* independently of their execution speed. In a sense, the closed loop is unrolled over time. As soon as both simulations have progressed independently by Δt , the CLE maintains *information synchrony* by forwarding the output of one simulation to the next timestep's input of the other. Unlike the conversion between different interfaces and the enforcement of temporal synchrony, information synchrony cannot be established automatically and depends on two specifications:

- *Sensorimotor Mapping*: The sensorimotor mapping determines how sensor information and motor commands are relayed between the brain model and the robot model. It provides an exact specification which neurons or regions of the brain receive the output of which sensors of the robot and, conversely, which actuators of the robot receive control commands from which neurons or brain regions.
- *Encoding of Information*: The robot and the brain model typically use different techniques to encode sensorimotor information. For example, common robot sensors provide measurements as real numbers while spiking neuron models communicate by exchanging digital pulses. Connecting both requires a specification of how to translate back and forth between the different representations of information expected by the robot and the brain.

It is these two specifications that define the closed PCA loop between the brain and the body. Their definition is at the core of neurorobotics research, which is why they are represented by a dedicated NRP concept called transfer function (TF). TFs enable the definition of both sensorimotor mappings and the conversion between different representations of information. They are based on the Python programming language, which allows for the implementation of any desired behavior [294]. As shown in Figure 3.8, the CLE invokes the TFs defined for an experiment on every synchronization of the two

simulations. Data exchange therefore only happens in between two simulation steps, accounting for a transmission delay of up to Δt for an individual signal. For this reason, Δt can be adjusted individually for an experiment to meet the required accuracy. The back end also contains a programmable state machine for automating control flows in NRP experiments. However, this component is not relevant in the context of this work and will therefore not be discussed in more detail. Instead, an alternative approach to the design of experimental protocols will be presented in Section 4.1.

NRP Back End Web Services The NRP Back End is designed to run fully independently on both local and remote computing resources. For this reason, it exposes only a small set of well-defined services that are accessible by other applications through a REST-based interface. Examples include commands for launching simulations, managing TFs or querying the system health. In addition, clients can connect directly to Gazebo and ROS through web sockets for live visualization and control of running experiments.

NRP Front End The *NRP Front End* depicted in the upper part of Figure 3.7 encompasses all components and services that mediate between the user and the experiments running on the *NRP Back End*. Technically, it is comprised of two sub-layers that run on the client computer of the user and the server in the data center, respectively. As can be seen at the top of the figure, the NRP can be accessed through three tools that realize two different user interfaces. The main entry point is a web application with an *Experiment Manager* and an *Experiment Viewer* that run in the browser without any installation. With the manager, users can upload new experiment descriptions, create experiment instances from templates and launch existing experiments on a back end. When an experiment is launched, the tool automatically switches to the viewer depicted in Figure 1.6 and the middle inset at the top of Figure 3.7. It is the NRP's main graphical user interface and combines tools for editing (Environment Editor, Brain Editor, etc.) and visualizing (3D environment rendering, plotting, etc.) neurorobotics experiments. Both the Experiment Viewer and the Experiment Manager are designed for direct user interaction with individual experiments. The *Python API* provides an alternative interface to the NRP that can be accessed from other applications. This makes it possible to automate the execution of experiments, which is an important prerequisite for parameter optimization or testing a brain model in many different environments.

NRP Front End Services All services required by the web applications and the Python API are implemented in the lower sub-layer of the front end and run on a remote host. Figure 3.7 lists some of the main components such as services for user authentication and storage. One of the most important of them in the context of this work is the *NRP Back End Proxy*, which manages all available NRP Back Ends. This is necessary because a typical NRP installation in a high performance computing center comprises only one front end but multiple back ends. The NRP Back End Proxy is therefore essential for running multiple experiments in parallel. In particular, the number of back ends available determines the number of experiments that can run at the same time.

Virtual Neurorobotics Experiments

The NRP combines all features and components required to conduct *virtual neurorobotics experiments* within a fully digital integrated tool chain that covers all characteristics outlined in Figure 3.6. An important precondition for this approach to work is a digital representation of all relevant aspects of an experimental setup. In the NRP, this representation is realized as shown in Figure 3.9 by a set of configuration files.

Every NRP experiment is defined by an *Experiment Configuration* file that contains meta information such as name, author and description, general settings such as robot and camera configurations, as well as references to further model configuration files. Among those is the *Environment Model* that needs to be provided in the Simulation Description Format (SDF), a file format for robots and 3D environments that is used by Gazebo. The environment model file may itself again reference further external content such as other model files, 3D object meshes and simulation plugins. In general, all model definitions and corresponding resource files are not stored along with the experiment but in a central repository so that they can be shared across different experiments.

At the core of the experiment is the actual embodied system that is comprised of a brain model, a robot model and a mapping between them. All three of them are defined in the *CLE Configuration* file, which is the second mandatory external reference in the experiment configuration. Like the environment model, the *Robot Model* needs to be provided as a Gazebo-compatible SDF description and is by default stored in a shared repository. The *Brain Model* is provided as a Python script and stored inside the experiment directory. Its content is typically fully user defined with the structure depending largely on the employed brain simulator. Code for TFs can be provided directly inside the CLE configuration file or inserted as references to external Python code. Like the file format for the experiment description, that for the CLE configuration is based on the Extensible Markup Language (XML).

Besides the environment model and the CLE configuration, the experiment description can optionally contain two further file references. The first is a *ROS Launch Script* which lets users automatically start ROS components on launch of an experiment. As will be shown in Chapter 4, this feature makes it possible to extend experiments with features and control logic beyond the functionality offered by TFs and ensures compatibility to commonly used tool chains and software stacks in robotics. Finally, it is also possible to specify experimental protocols as configurations for the state machine execution environment that is included by default in every NRP installation.

The NRP's experiment description format covers the *Setup* phase from Table 3.1. Its structure provides effective guidance for the configuration of closed-loop systems and is at the same time open enough to accommodate a huge variety of different experimental setups. However, it does not yet offer any support for advanced experiments with possibly hundreds of parallel simulations, which are mandatory for replicating the perceptual experience of living creatures within a practical time frame. For this reason, the following chapter will introduce an extension of the NRP for parallel distributed learning.

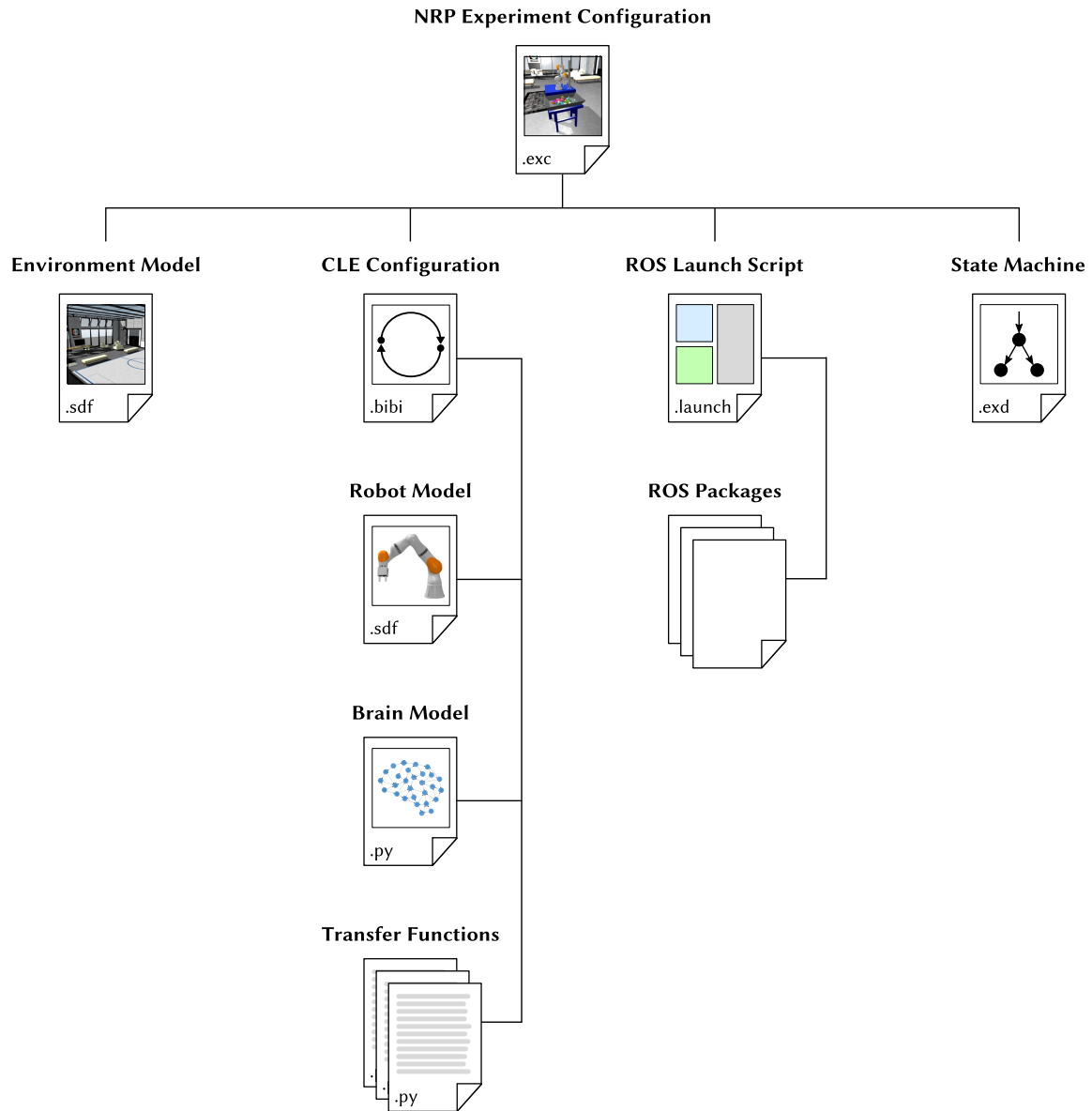


Figure 3.9: Configuration file hierarchy of an NRP experiment. The annotations at the bottom of each file indicate the file type. Typically, not all configuration and model files of an experiment reside in a common directory. Instead, objects that are shared between several experiments such as models or ROS packages are stored in globally accessible repositories. Lines between files therefore indicate references. Some of the depicted file types such as the Environment Model may contain further references to external sources that are not shown in the figure.

4

A New Tool Set for Neurorobotics Experiments

The HBP Neurorobotics Platform provides a powerful framework for virtual neurorobotics. It already contains a baseline set of models and experiments that are readily available after installation by default. They range from simple tutorials for phototaxis on a mobile robot with only a few neurons to a complex show case experiment with a musculoskeletal simulation of the mouse body that is based on a neuroscientific study [295]. Creating variations of these templates or extending them is to a large extent possible directly from within the NRP using the tools of the front end's web applications. However, the design of new types of experiments requires extending the underlying simulation framework. This chapter introduces two novel extensions for the NRP that add support for a completely *new experiment class* and take advantage of *hardware acceleration*. Section 4.1 describes a framework for parallel distributed learning that supports running hundreds of experiment instances at the same time to accelerate training and expand the space of collected experiences. It is universal and not limited to a specific application or area of research. Section 4.2 targets the acceleration of experiments with brain models that are based on SNNs by providing a new interface to the neuromorphic processor Intel Loihi that is fully integrated into the NRP. Real-world validation is an essential part of virtual experiments. In Section 4.3, we present a novel biomimetic mouse robot that complements the aforementioned musculoskeletal simulation of the mouse body to enable knowledge transfer between virtual NRP experiments and validation studies in real-world environments.

4.1 Massively Parallel Neurorobotics Experiments

The goal of virtual neurorobotics experiments is to reproduce real-world sensorimotor experience as closely as possible, both in terms of *quality* and *quantity*. High quality refers to the fidelity of the simulation as well as the underlying robot and environment models. Compared to previous tools, the NRP sets a high standard in this regard by adopting Gazebo for the environment simulation and adding support for musculoskeletal modeling. A downside of this approach is the high computational complexity, which slows down the experiment simulation. As a result, the quantity of sensorimotor experience that can be generated in the course of an experiment within practical time frames is limited. However, this quantity is essential because it is directly related to the coverage of the sensorimotor space. As von der Malsburg [296] points out, the description of the human genome corresponds to about one gigabyte of data and the environment can be described by only a few gigabytes of 3D models. By contrast, the description of the synaptic connectivity in a developed brain amounts to a petabyte of data. This complexity emerges during ontogeny from countless unique interactions with the environment. In the following subsections, we develop an approach to reproducing this rich body of sensorimotor experience with the NRP. It takes advantage of state-of-the-art cloud computing to run hundreds or even thousands of experiment simulations concurrently. The development of the required framework, models and algorithms is guided by a real-world case study on robot grasping by Levine et al. [297] that relied on physical robots for parallelization. An important goal of this section besides the extension of the NRP is therefore also to highlight the advantages of virtual over physical robotics. The results reported in this section have been partly published in [298].

A Case Study on Data-Driven Robot Grasping

The reliable grasping of objects is a long-standing research topic in robotics with a huge variety of different approaches ranging from analytical methods [299] to DNNs in the cloud [300]. Typically, they incorporate a huge body of implicit or explicit knowledge about the problem domain. Levine et al. [297] proposed a purely data-driven method that is capable of learning how to grasp objects completely from scratch without any prior knowledge by replacing manual engineering with massive amounts of data. While not taking explicit advantage of embodiment, the method requires training data of both high quality and quantity. The exclusive focus on data makes it an ideal prototypical use case for guiding and validating the development of the technical infrastructure for massively parallel experiments on the NRP. In the remainder of this subsection, we therefore provide a brief summary of the experimental setup based on the description by Levine et al. [297, 301, 302]. The findings will later guide the development of a simulation of the experiment in the NRP.

Figure 4.1 provides an overview of the experimental setup with a total of 14 identical 7-DoF robots. Each of them is equipped with a compliant underactuated two-finger gripper and a camera that is mounted to a fixed stand behind the robot. In front of every

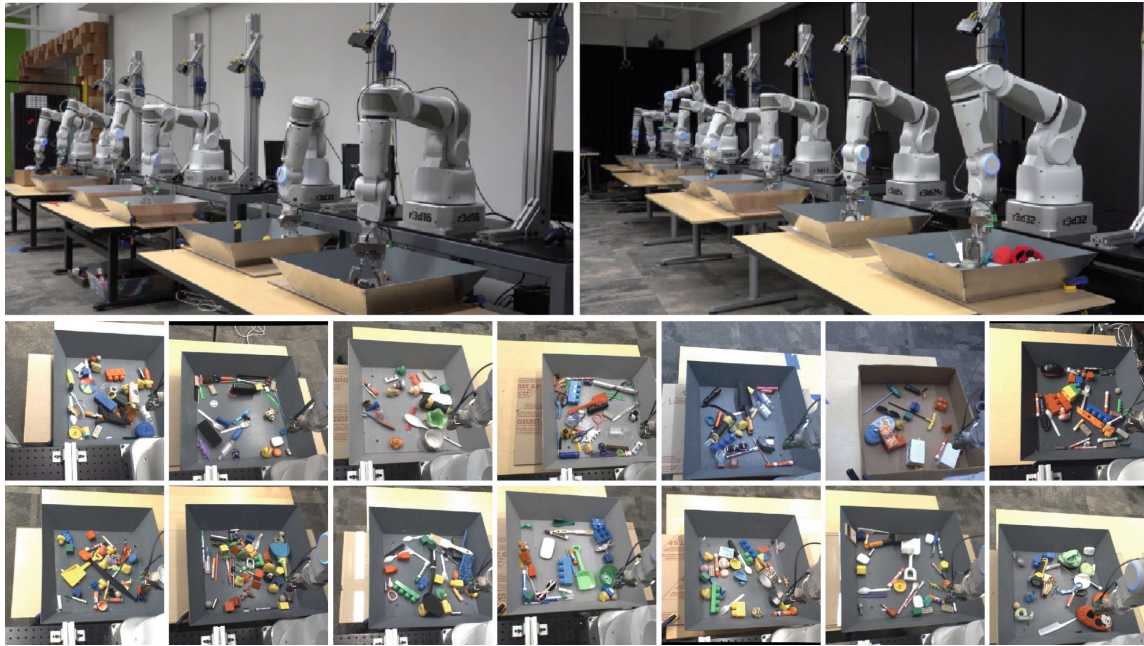


Figure 4.1: Parallel data collection for learning to grasp objects on multiple robots. *Top:* Overview of the experimental setup. The robots are placed in front of a trays filled with objects. Each of them is equipped with a compliant underactuated two-finger gripper and a fixed mounted monocular RGB-D camera. *Bottom:* The trays of all 14 robots that were used in the experiment. Adapted from Levine et al. [302] by permission from Springer Nature Customer Service Centre GmbH, ©2017.

robot is a tray filled with objects. There are no details provided about the object types considered in the experiment except that they were “chosen among common household and office items, and ranged from a 4 to 20 cm in length along the longest axis” [297]. It is further stated that complexity was increased in a later set of experiments, where “around 1100 different objects” [297] were used. This figure provides an upper bound for the initial experiment phase. To increase diversity, the camera position slightly differed from robot to robot as well as the objects in the trays. During the runtime of the experiment, additional variation between the robots was introduced through individual wear and tear of the grippers. Moreover, the content of the trays was replaced manually from time to time. The amount of objects in a single tray and the replacement frequency are not communicated by the authors. An important constraint enforced through the encoding of robot motion commands is that the gripper is always facing downwards at an angle that is perpendicular to the base of the tray. Movement is only possible around z -axis. Some example configurations are shown in Figure 4.1. As a result, objects in the tray can only be grasped from the top with pinch-type grasps [303]. It is important to note that this implies a drastic reduction of the action space compared to unconstrained grasping with multiple fingers.

With the limitation of the experiment to a single predefined grasp type, the only task remaining is to move the robot to a configuration where closing the gripper will most likely result in the successful grasp on an object. A key feature of the experiment is that

the control of the robot is implemented by a single DNN N that computes grasp success probabilities solely based on monocular RGB images from the cameras mounted behind the robots¹ and a relative motion command defined in task space with respect to frame of the robot base. More concretely, the network approximates the following probability function:

$$N(\mathbf{I}_0, \mathbf{I}_t, \mathbf{v}) \approx P(\exists o \in \text{Tray}(t). \text{robot_ttp}(p, t) \wedge \text{move}(\mathbf{v}, t) \implies \text{grasp}(o, t + 1)) \quad (4.1)$$

In the equation above, \mathbf{I}_0 , \mathbf{I}_t and \mathbf{v} denote an unobstructed camera image of the tray, a camera image of the tray taken from the same perspective but also showing the current position of the robot arm in time step t , and a task space motion vector. $\text{Tray}(t)$ is the set of all objects in the tray in front of the robot in time step t , $\text{robot_ttp}(p, t)$ expresses that robot's tool tip is at position p in time step t , $\text{move}(\mathbf{v}, t)$ denotes that the tool tip center of the robot moves and rotates along \mathbf{v} in time step t and $\text{grasp}(o, t + 1)$ is true iff the robot successfully grasps object o when closing the gripper in time step $t + 1$. Deriving an analytical solution for the right hand side of Equation 4.1 requires exact knowledge of all objects and their positions in the tray, as well as an algorithm to compute whether closing the gripper at task space position $p + \mathbf{v}$ will result in a successful grasp. While the former is a computer vision task that is particularly complex when there are many objects that can occlude each other, the latter is only possible with a physics model that is capable of computing contact forces between the gripper and the objects. N needs to solve both of them at the same time solely based on visual information from the camera images \mathbf{I}_0 and \mathbf{I}_t . Figure 4.2 shows the network architecture.

Training the prediction network effectively comes down to solving a binary image classification problem. The difference lies in the structure of the input which encompasses two images and a vector instead of only a single image. Collecting a corresponding data set D is only possible with a baseline control algorithm that executes grasp attempts before N is available. To resolve this issue, Levine et al. [297] apply a bootstrapping procedure that starts by issuing random motion vectors \mathbf{v} to collect a base data set D^0 . This enables the training of a first version N^1 of the grasp prediction network, which then serves as a basis for iterative data collection and training. In total, the parameters of N are re-trained four times with additional data:

$$D = D^0 \cup D^1 \cup D^2 \cup D^3 \cup D^4 \quad (4.2)$$

In the above equation D^i denotes the data set collected with network N^i for $i > 0$. The final grasp prediction network results from training on the full data set D . The samples in every set D^i are collected over multiple time steps $t \in (1, \dots, T)$, the last of which always corresponds to closing the gripper. Every time step t of an episode e yields a sample $S_t^e = (\mathbf{I}_0^e, \mathbf{I}_t^e, p_T^e - p_t^e, l^e)$. p_t^e denotes the robot's current configuration and p_T^e its final configuration in that episode. In particular, the last sample from an episode always has the

¹Actually, the cameras captured RGB-D images that are available in the data set released along with the experiment [304]. Nevertheless, the depth information is not used in the control loop of the robots.

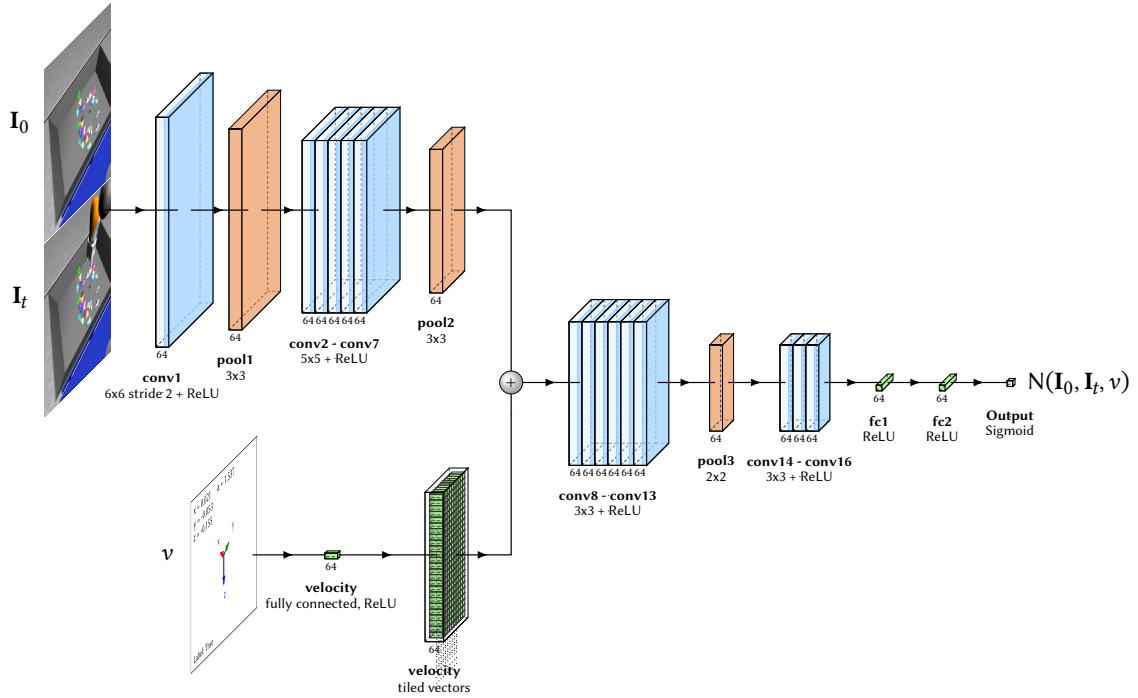


Figure 4.2: Architecture of the grasp prediction network $N(\mathbf{I}_0, \mathbf{I}_t, \mathbf{v})$. **conv**, **pool** and **fc** denote convolution layers, max pooling layers and fully connected layers. If not stated otherwise, the stride size of all convolution and max pooling layers is 1. Layer **conv1** is assumed to be applied without padding while all further convolutions use padding to preserve the feature map dimensions. Furthermore, all convolution layers apply batch normalization. The motion vector is mapped to a fully connected layer that is then spatially tiled across all feature map dimensions to enable direct addition to the output of **pool2**. **fc2** is connected to a single neuron with a sigmoidal activation function to map the network output to a probability value. Adapted and extended from [297].

form $S_T^e = (\mathbf{I}_0^e, \mathbf{I}_T^e, 0, l^e)$, which means that no motion command is issued and the gripper closes. If the grasp is successful the label l^e is set to **true**. In the original experiment, the episode duration was increased from $T = 2$ to $T = 10$ during data collection, resulting in an increment of two after each training phase.

What is still missing yet is a method for leveraging the predictions of N in the robot's control loop. The network alone can only assess the quality of a task space motion \mathbf{v} with respect to grasp success. Determining a motion vector \mathbf{v}^* that maximizes $N(\mathbf{I}_0, \mathbf{I}_t, \mathbf{v}^*)$ is implemented in a separate control function that samples candidate solutions based on the Cross-Entropy Method (CEM), a gradient-free optimization algorithm [305]. While the robot moves and \mathbf{I}_t changes, \mathbf{v}^* is continuously updated to compensate for model errors by generating new samples around the current robot configuration for the CEM optimization. N thereby becomes the model M of a closed PCA loop as illustrated in Figure 3.2. As Levine et al. [297] point out, the control loop is solely based on visual input. The approach can therefore be described as special type of visual servoing, which is a common task in traditional robotics. But unlike earlier work in the field, there is no camera calibration required. Therefore, N effectively learns hand-eye coordination.

Virtual Experimental Setup for the Neurorobotics Platform

Creating a digital twin of the robot grasping experiment involves not only the virtualization of the complete physical setup but also of the experiment protocol that is applied for data collection. This subsection addresses the *Design* and *Setup* phases as introduced in Table 3.1. The instantiation and execution phases will be discussed in the next two subsections. An overview of the final NRP experiment is depicted in Figure 4.3.

Models As the model library of the NRP is primarily aimed at use cases from neuroscience, most NRP-related components of the virtual experiment need to be designed from the ground up. A natural starting point is the modeling of the robot along with an appropriate gripper and control interfaces. Like the latest version of the original experiment, the digital version on the NRP is based on a *KUKA LBR iiwa* robot. This specific model not only has seven degrees of freedoms (DoFs), but it is also widely used in both research and industry, which makes it a valuable addition to the NRP’s model library for a broad range of applications. 3D models and a URDF description of the robot are readily available as of part the ROS Industrial Program [306]. The gripper of the robot is based on a URDF model of a *Schunk WSG 50* parallel gripper with two fingers [307] and 3D meshes of a similar *Schunk PG 70*. The fingers of the gripper were designed with the 3D modeling software *Blender* [308]. In summary, the overall setup is comprised of the robot, the gripper and the tray containing the objects to be grasped. As can be seen in



Figure 4.3: Digital twin of the robot grasping experiment in the NRP “Holodeck”, a digital lab space for virtual neurorobotics experiments. A KUKA LBR iiwa robot equipped with a Schunk PG 70 parallel gripper is placed next to a tray with random objects. The content of the tray is captured by an RGB camera mounted behind the robot.

Figure 4.3, there is a close correspondence to the setup of the physical experiment shown in Figure 4.1. Since both the robot and the gripper model are specified as URDF macros, replacing them by other models is possible with little effort.

What is still missing yet are 3D models of the objects to be grasped by the robot. Even though selecting and adding them may appear to be a trivial task at first glance, it is actually one of the major challenges in creating the virtual experiment. This is due to the large number and wide variety of objects required. Not only do they have to have shapes and sizes that are compatible with the capabilities of the robot's gripper, but they must also represent a diverse range of geometries and appearances. Also, their 3D meshes need to meet a number of common quality criteria, such as watertightness and correct orientation of surface normals. While a large number of 3D models are freely available online, selecting, checking and adapting them involves time-consuming and cumbersome manual work. For this reason, the setup includes a set of about 1000 procedurally generated random objects that have been published specifically for applications in robotics [309, 310]. Some of them are depicted in Figure 4.3. As an alternative, the experiment also contains a small selection of everyday objects.

Controllers The modeling efforts discussed so far only cover the robot's kinematics. In general, it is also necessary to define all parameters governing its dynamics, such as frictions, forces and inertia. To accurately reproduce the robot's dynamics, careful calibration is required and the final results depend significantly on the quality of the physics simulation. However, this effort is only required when the goal of the experiment is to develop a model or algorithm that takes into account the robot's dynamics. This is, for example, the case when training a low-level force controller. Since the control algorithm described in the last subsection solely outputs relative motion vectors in task space, no detailed modeling of the system dynamics is required as long as the robot has an *inverse kinematics model* and supports *position control* in joint space. To realize the latter, the robot's model was extended to work with the *ros_control* framework [311], which provides standard control interfaces for ROS and corresponding reference implementations of joint position controllers. In a sense, position control hides possible inaccuracies of the dynamics simulation and thereby makes the simulated robot highly accurate with respect to the ground truth of the actual physical experiment. Inverse kinematics transformations are computed by *MoveIt*, a software framework for robot motion planning that interfaces with *ros_control* [312]. The only exception where force control is applied instead of position control is the robot's gripper, in order to ensure that objects can be grasped even when their exact geometry is unknown. Because the fingers of the simulated gripper are rigid, this approach can be seen as an approximation of the compliant gripper used in the physical experiment.

Programming Interfaces In terms of Table 3.1, the virtual model of the experiment described so far corresponds to the *Setup* phase, where a general experiment class from the *Design* phase is instantiated with concrete models for the robot, the brain and the

environment. In a more general view, the experiment class can be defined as grasping. It describes the overall domain addressed by the experiment and experiments from the same class naturally share many common features, such as the presence of some form of hand or gripper and a set of objects that the robot should grasp. Likewise, there are common commands, such as closing the gripper and checking whether an object was successfully grasped. This makes it possible to define a general application programming interface (API) for an experiment class that abstracts common functionality from specific implementations. Individual experiment classes are thus effectively determined by their unique APIs.

The definition of an API is not only of conceptual relevance, but has two important practical implications. First and foremost, prospective users who want to test a brain model in a specific experimental setup only need to know the high-level API rather than the low-level interfaces from the NRP, Gazebo, ROS and other components that might be involved. Conducting experiments thereby becomes considerably easier and requires less technical knowledge. Secondly, a standardized API makes it possible to switch seamlessly between different setups of the same class to evaluate the performance of a brain model in different settings. The experiment API for grasping is implemented in Python and comprises all functionality that is required for the experiment protocol outlined in the last subsection. Similar approaches to standardizing the interfaces of virtual environments have been published earlier, with prominent examples being *OpenAI Gym* [313] and *Reinforcement Learning Coach* [314]. Most related to this work are attempts to provide OpenAI Gym-compatible interfaces for Gazebo [315, 316, 317]. However, they exclusively target reinforcement learning and lack the software infrastructure required for massively parallel experiments, which will be introduced in the next subsections.

Complete Experiment Figure 4.4 summarizes all components of the *NRP Grasping Experiment* based on the schema from Figure 3.9. The arguably most salient difference is that many components defined by the latter are missing, particularly the brain model and the TFs. For this reason, the CLE configuration only needs to be included in order to specify the robot model and simulation parameters, which is why it is grayed out in the schema. Both the brain model and the TFs are implemented externally and connect to the experiment through the *Experiment API*, which, together with the robot model, forms the system's core. As can be seen in the figure, there are actually three different model files that are related to the robot. The main specification that also hosts the configuration for the `ros_control` framework is contained in the robot's URDF description. A corresponding SDF model for Gazebo can be automatically generated from it. Finally, the SRDF configuration contains additional parameters required for computing the robot's inverse kinematics. All three components located in the lower part of the right branch of the schema are implemented as ROS packages. They are managed by a ROS launch script which is referenced in the global experiment description file.

The NRP Grasping Experiment is designed to reflect all relevant features of the original physical setup as closely as possible. Nevertheless, there are still qualitative and quantitative differences. Table 4.1 provides an overview. One of the main deviations is

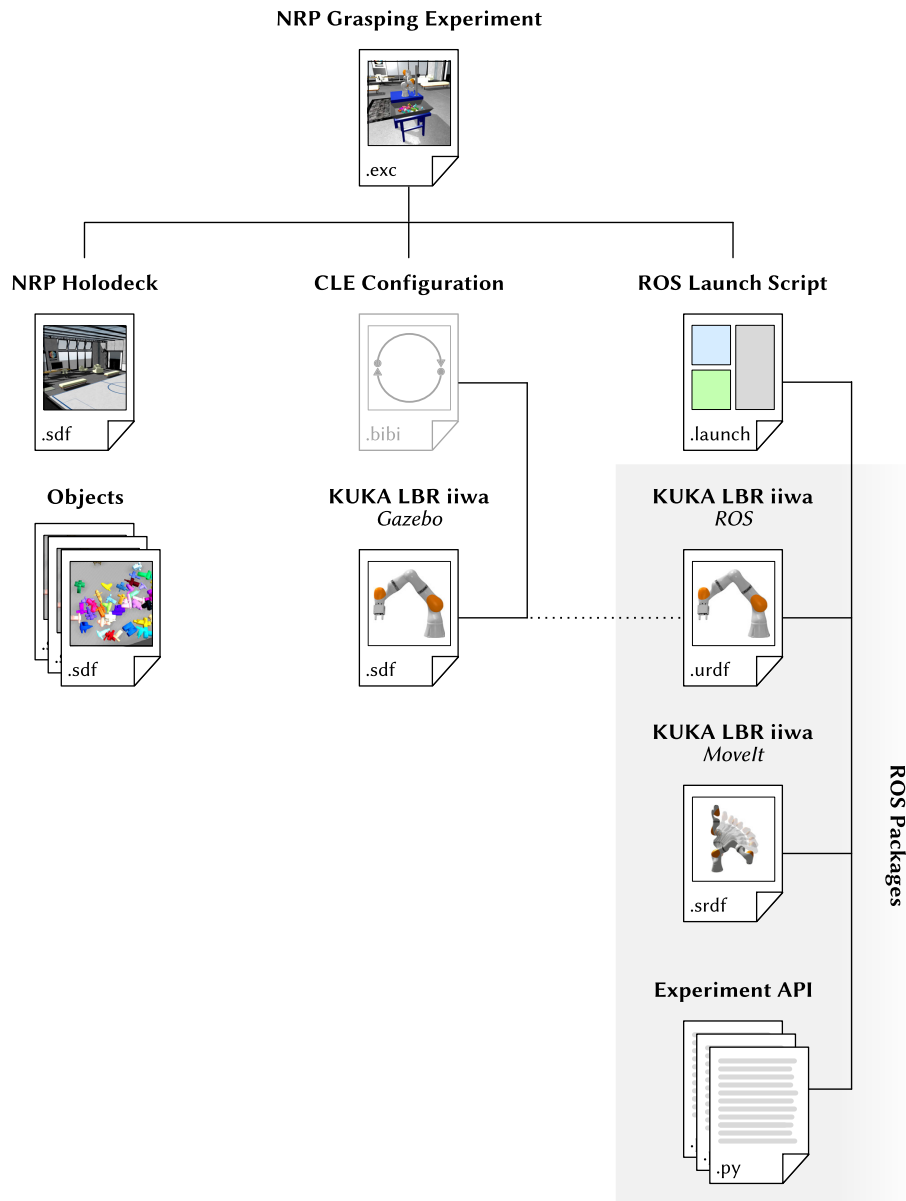


Figure 4.4: Overview of all components of the NRP Grasping Experiment. The schema instantiates the general structure of NRP experiments from Figure 3.9. Unused components are left out. The *CLE Configuration* is grayed out because it is required technically, even though the experiment uses an external brain model and therefore does not include any TFs. Even though they are not part of the actual experiment definition but added during runtime through the *Experiment API*, the object models are shown since they are constituent parts of the experiment. The dotted line between the *Gazebo Robot Model* and the *ROS Robot Model* indicates that the SDF model is generated automatically from the URDF model.

Table 4.1: Comparison of the virtual and physical experiment setups for robot grasping. The properties of the physical experiment are based on the description by Levine et al. [297] of the more recent setup with the KUKA LBR iiwa robot.

	Virtual NRP Experiment	Physical Experiment
<i>Robot</i>	KUKA LBR iiwa	KUKA LBR iiwa
<i>Gripper</i>	Schunk PG 70 two-finger gripper, fully acutated	Compliant two-finger gripper, underactuated
<i>Objects</i>	≈1000 procedural rigid objects	≈1100 “large, small, hard, soft, deformable, and translucent” [297] objects
<i>Physics</i>	Rigid body dynamics	Soft body dynamics
<i>Performance</i>	Scales with computing power	Fixed to real time
<i>Costs</i>	Usage-dependent cloud fees	> 60 000 EUR for the robot alone [318]
<i>Setup Time</i>	Seconds to minutes	Hours to days

the limitation to rigid body dynamics imposed by Gazebo. Consequently, there are also differences in the gripper and the types of objects considered. It is planned that future releases of the NRP will address this point by adding support for alternative simulation engines. Nevertheless, the main characteristics of the physical setup are conserved and there are no conceptual limitations. The second major point of distinction is the effort required to install and run the experiment. Adding another virtual robot only takes seconds to minutes while scaling up the physical experiment is not only very expensive but also takes considerably longer. Moreover, when running the NRP on cloud infrastructure, there are no fixed costs and the experiment can be scaled to an arbitrary number of robots working in parallel. Virtual experiments such as the NRP Grasping Experiment thereby augment physical experiments not only quantitatively but also qualitatively by making it possible to address new research questions that would otherwise involve high investments in infrastructure or take too long to be practically feasible.

Extended NRP System Architecture

So far, the NRP Grasping Experiment comprises only a single robot. Extending the configuration files to include additional ones is technically feasible, but generally unfavorable in terms of complexity and scalability. Both Gazebo and ROS are designed to simulate and control a single experiment and cannot be easily parallelized in a cluster. It is therefore not possible to take advantage of distributed computing infrastructures and the maximum experiment size becomes effectively limited to the capacity of a desktop computer. Further challenges include the separation of all robots within a single ROS name space and the high volume of data transfer that would need to be handled by a single ROS core process. For this reason, parallelization in the virtual experiment is realized by launching multiple experiment instances rather than adding multiple robots to a single instance. This is made possible by the NRP’s distributed system architecture that is in the following extended to support large-scale experiments with potentially thousands of parallel instances.

Default System Architecture The NRP bundles many interdependent software components, ranging from the simulation engines in the back end to the web applications in the front end. Installing them individually is error-prone and not practical for large setups. This is why the front end and the back end are also available as preconfigured images for the container-based OS-level virtualization framework *Docker* [319, 320]. They are self-contained and run on any platform with support for Docker and the x86-64 instruction set architecture, independently of the installed operating system and library versions. Importantly, the encapsulation inside containers makes it easy to scale up simulations and launch multiple NRP instances on a single machine, making the system completely independent of the underlying computing infrastructure. In the default distribution of the NRP, a single front end container manages multiple back end containers to support parallel access by different users. A simplified schema of the underlying system architecture is depicted in Figure 4.5. Depending on its performance, a single compute node can host multiple back end containers. All of them operate completely independently and are assigned dynamically as soon as a new experiment is launched from the front end. As user data, models and experiments are provisioned through a shared storage on the front end compute node, every back end can execute every experiment.

Extensions for Parallel Experiments The default system architecture of the NRP supports the parallel execution of multiple experiments, but not the execution of parallel experiments. The difference between the former and the latter lies in the dependences between the individual experiment simulations running on the back ends. In a parallel experiment, they run in a common context, process data that belongs to a single neurorobotics experiment and possibly share data with each other. This is fundamentally different from a number of independent simulations that run concurrently and motivates the extension of the NRP system architecture presented in Figure 4.6. The arguably most salient feature is the increased number of components located on the front end node, which now hosts additional data storage servers and management interfaces. Table 4.2 lists the software packages chosen for each of the services.

At the core of the extended system architecture is a central data base that is accessible by all NRP instances. It not only stores metadata of experiments and collected raw data but also serves as a central parameter server that distributes experiment settings to all

Table 4.2: Selected implementations of the additional services defined in the extended NRP system architecture. All chosen frameworks are open-source and freely available.

Service	Implementation	Source
<i>Object Storage</i>	MinIO	[322]
<i>Data Base</i>	MongoDB	[323]
<i>Data Base GUI</i>	mongo-express	[324]
<i>Dashboard</i>	Metabase	[325]

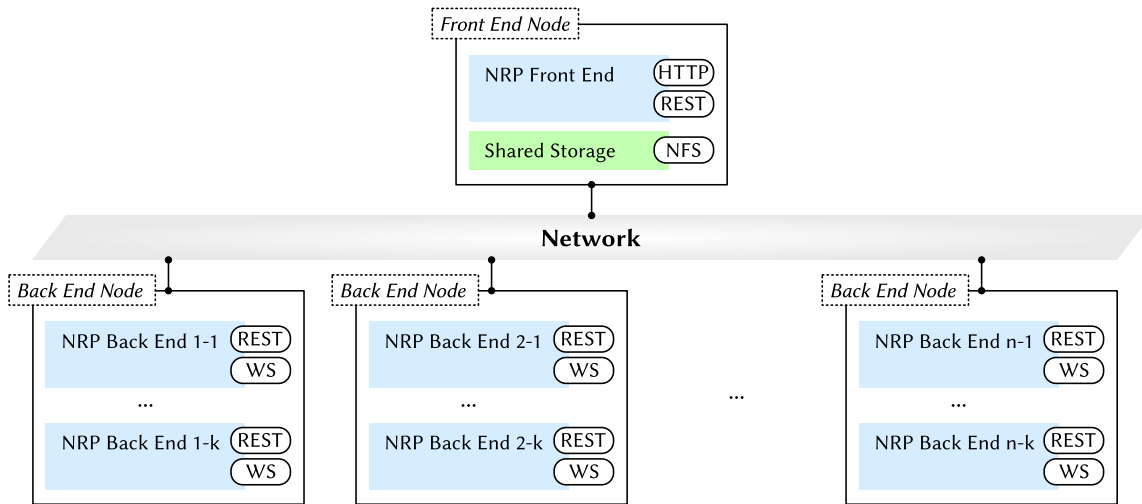


Figure 4.5: Default system architecture of the NRP with Docker-based deployment. The outer boxes correspond to compute nodes. Docker containers and storage are highlighted in blue and green, respectively. Tags in rounded boxes indicate exposed interfaces. In particular, HTTP refers to a web application and WS denotes a web socket.

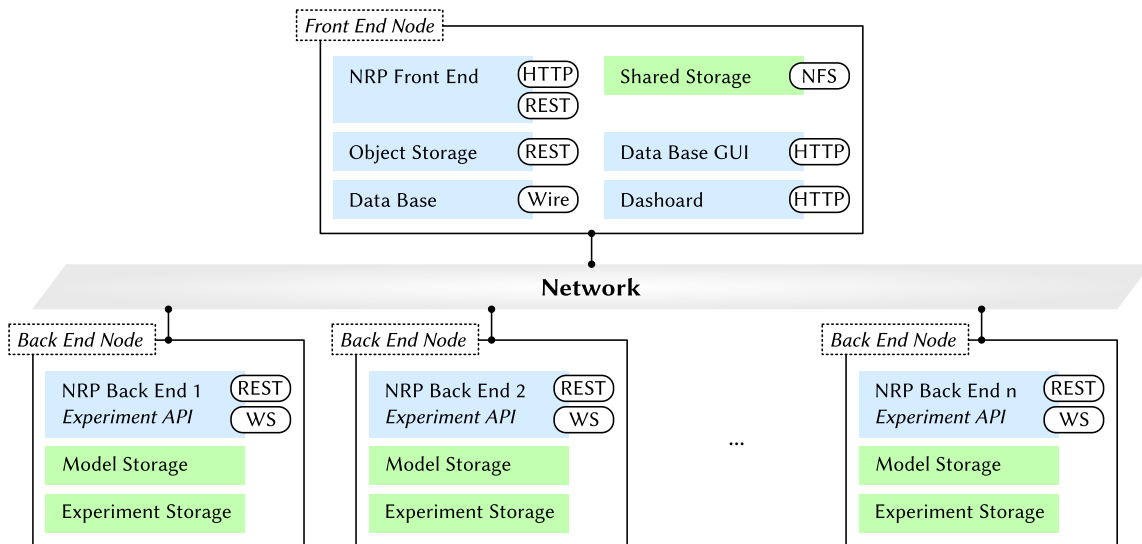


Figure 4.6: Extended system architecture of the NRP for parallel distributed learning. Symbols and abbreviations are identical to Figure 4.5. Storage in the back ends is only available to local containers. The interface Wire refers to the communication protocol of the MongoDB document data base [321].

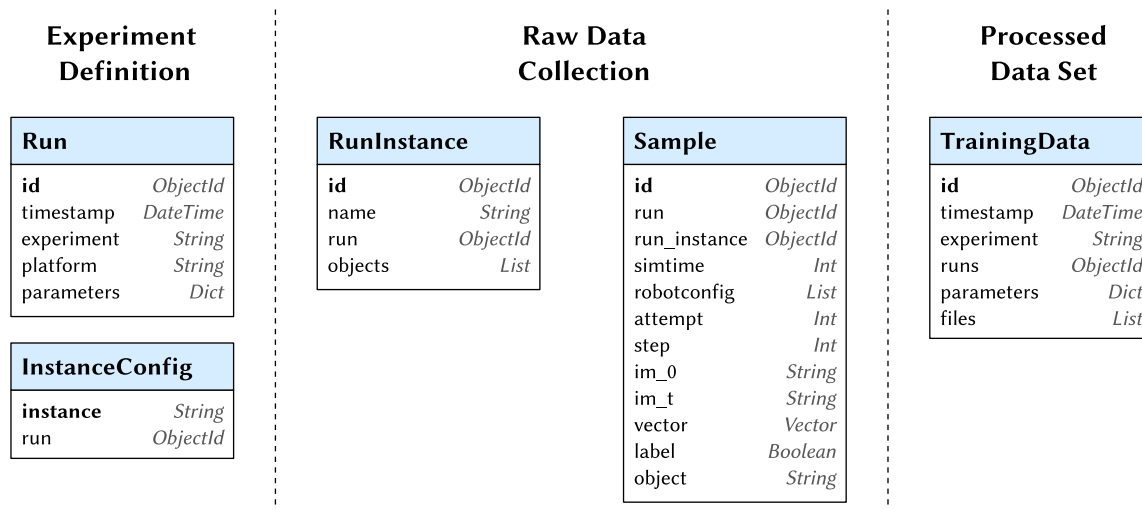


Figure 4.7: Document type definitions for the data base of the extended NRP system architecture. The schema only depicts the most relevant parts of the actual implementation. Variables printed in bold letters at the top of every document definition denote primary keys. Every conducted experiment is identified by a document of type Run that is referenced by all other documents related to this experiment. The document types are grouped into three different categories for the definition of experiments, the raw data generated during runtime and the post-processed data set.

back ends. To account for the rather dynamic nature of the data generated in robotics experiments, a document-oriented data base was chosen instead of a traditional relational data base. This allows for an easy adaptation of the data scheme when the experimental protocol changes or additional data needs to be stored. Moreover, document-oriented data bases are better suited for storing raw data, which fits very well to the requirements of neurorobotics experiments. Figure 4.7 provides a summary of the document structure defined for the NRP Grasping Experiment. Every run of the experiment is represented by an instance of the document type Run, which is referenced by all data base entries resulting from its execution.² It points to one or more documents of type RunInstance that represent individual experiment simulations running on NRP back ends and, for example, store the object models the robot is supposed to grasp in the corresponding instance of the experiment. Analogously, individual grasp attempts are stored along with all parameters in documents of type Sample. To keep the memory footprint of the data base low for increased responsiveness to queries, image data is stored separately in an object storage. Consequently, the individual samples only contain file references. Taken together, RunInstance and Sample are the results of the experiment *execution phase* as defined in Table 3.1. The *instantiation* and *evaluation* phases are represented

²While redundancies are typically avoided in traditional data base schemes, accessing the corresponding Run from every document considerably simplifies the most common queries and is therefore advantageous for this specific use case.

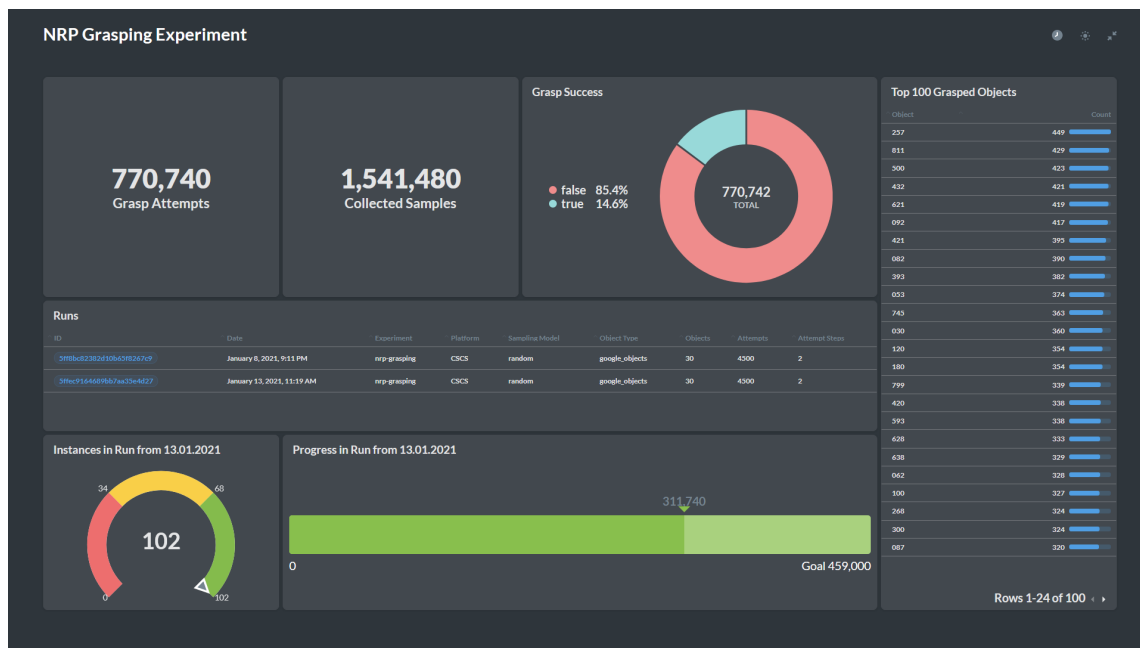


Figure 4.8: Dashboard for monitoring the data collection in parallel distributed NRP experiments. The depicted view displays not only the progress of the current experiment run and the number of executed grasp attempts but also statistics of individual objects.

by InstanceConfig and TrainingData, respectively. The former maps back ends to runs for parameter distribution at launch. The latter stores post-processed raw data from the execution phase, which is the main result of an experiment run in the case of the NRP Grasping Experiment. Actual metrics for a quantitative evaluation of the robots' performance can be computed based on the raw data from the execution phase. It is important to note that all document definitions can be easily extended to support other types of data and experiments. Even though the developed framework is motivated by the NRP Grasping Experiment, it is therefore nevertheless universal and flexible. A major advantage of using a data base to store the results of experiments is the possibility to query and analyze data. This makes it possible to create interactive dashboards with live statistics from running experiments. Figure 4.8 depicts an example from the NRP Grasping Experiment with detailed information about the experiment progress and grasp success rates.

To realize massively parallel experiments, the NRP system architecture has not only been extended but also optimized for performance. This is why the model and experiment storage now resides locally in every back end node to mitigate bandwidth bottlenecks during the launch of hundreds of simulations at the same time. Configuration and setup files are still located on the front end to enable the fast deployment of updates for components such as the Experiment API, which is installed automatically when a new back end Docker container is created. A new experiment launch script supports starting all simulations of a run within minutes through parallel calls of the NRP's Python API.

Parallel Distributed Learning

The virtual experiment setup enables the collection of training data analogously to the physical experiment. In tests on a compute cluster, the new architecture easily scaled to more than 100 parallel experiment simulations that collected data and stored it in the data base. Runs across instances were randomized in terms of the objects in the tray and the camera position. In the following, we will present concrete results from the NRP Grasping Experiment before the section closes with a discussion of how the developed system can be applied to other experimental setups.

NRP Grasping Experiment Following the design of original physical experiment, the NRP Grasping Experiment is split into two iterative phases for data collection and training. Figure 4.9 provides an overview. In the data collection phase, all experiment simulations perform grasp attempts and store them in the data base. Every attempt is split into multiple time steps T , after each of which a new motion command generated. At the beginning of the training process, samples are collected randomly with $T = 2$. In the first time step, the gripper moves along a randomly generated motion vector before it stops and closes in the second step. Figure 4.10 shows an example from the NRP Grasping Experiment. The top row corresponds to time step $t = 1$, the bottom row to time step $t = 2$. Each sample not only contains a view of the tray at the corresponding time step but also an unobstructed camera image that was captured at the beginning of the attempt before the robot moved into the field of view. The three components of the motion vector in the rightmost column are drawn individually at fixed angles to visualize their relative lengths.

As soon as enough samples have been collected, they can be exported for training. This step not only includes the conversion into an appropriate file format such as TFRecord

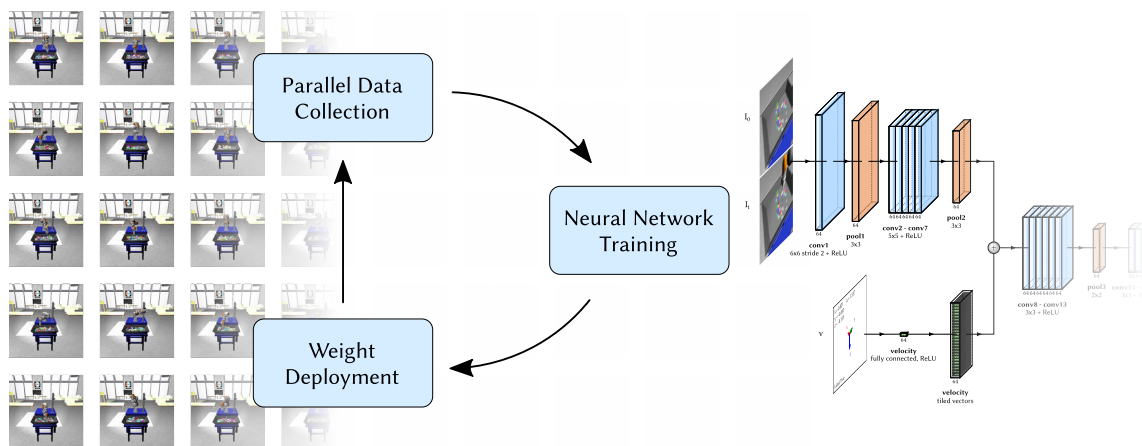


Figure 4.9: Parallel distributed learning in the NRP Grasping Experiment. Grasping data collected in the parallel experiment instances are used to train the grasp prediction network N . After every training step, the updated weights are deployed to the simulations on the back ends to collect further data with the improved network model.

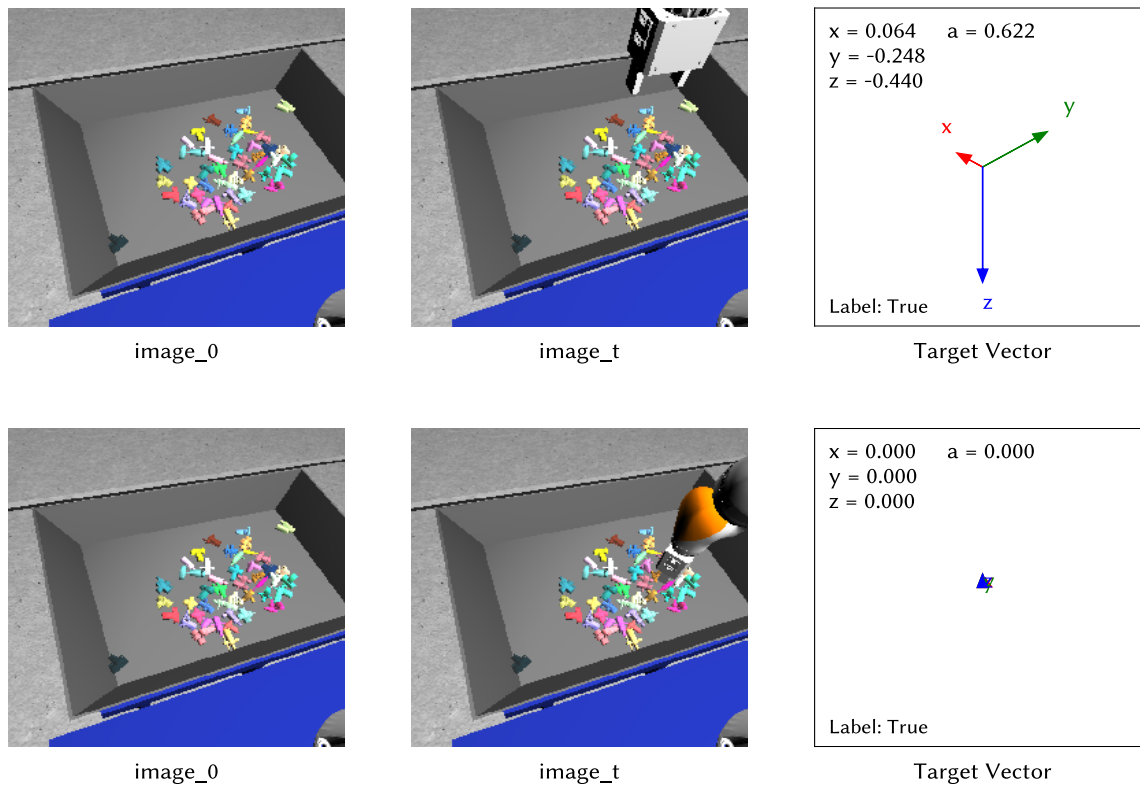


Figure 4.10: Samples of grasp attempts collected with the NRP Grasping Experiment. The format of the samples follows the definition from the physical experiment [297]. Each sample is comprised of two camera images with an unobstructed view on the tray and with a view of the current position of the robot’s gripper, respectively. The components of the generated motion vector and the label of the sample are depicted in the rightmost column. Both samples belong to a single grasp attempt with two steps, in the second of which the gripper stops moving and closes.

for TensorFlow [135], but also allows for additional pre-processing, such as sub-sampling the data set to guarantee an even distribution of positive and negative grasp attempts during the training phase. The actual success rate before sub-sampling can also serve as an indicator of how closely the overall dynamics of the physical experiment are captured by the virtual one. Figure 4.11 summarizes the results of hundreds of virtual experiment runs with random grasp attempts for three different numbers of objects. As expected, grasp success increases with a growing amount of objects in the tray. Not shown in the plot is the simulation time trade-off that occurs due to the increased complexity of the physics simulation for larger numbers of objects. The overall range of success rates is in line with the numbers reported by Levine et al. [297]. That the maximum rate of about 20 % is lower than the 30 % in the physical experiment can be explained by the fact that the simulation only supports rigid body dynamics and therefore requires a more exact positioning of the gripper compared to a compliant system. After exporting the collected data and training the grasp prediction network N , the network weights can be deployed to all experiment simulations to collect further training data.

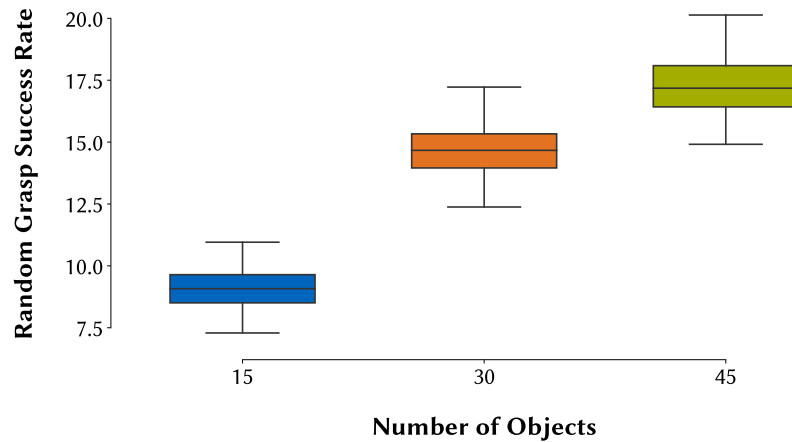


Figure 4.11: Box plot with success rates of random grasp attempts for different object numbers in the NRP Grasping Experiment. Data for each number of objects were collected from 100 simulations that executed 450 000 grasp attempts. More objects increase the success rate at the cost of simulation time. Success rates fall in a similar range as the 10 % – 30 % reported for the physical experiment. The lower maximum rate is most likely due to the simulation’s limitation to rigid body dynamics.

Support for other Experiment Types The extensions introduced in this section provide the basis for parallel distributed learning with the NRP. Even though some of the components such as the data base have so far only been discussed in the context of the NRP Grasping Experiment, they can be easily adapted to support other types of experiments. In particular, in Chapter 5 the system will be applied to a setup with a modified environment and a completely different learning task. Finally, the overall architecture also supports other types of distributed learning where a model is trained online during data collection such as *A3C* or *IMPALA* [326, 327]. Even though these methods do not build up sets of training samples, the data base can be still used for parameter distribution and storing meta data.

4.2 Neuromorphic Neurorobotics Experiments with Intel Loihi

The system developed in the previous section enables accelerated virtual neurorobotics experiments through massively parallel scaling. Especially applications where system performance is mainly limited by the speed of the environment simulation can benefit from this approach. One of the arguably most common use cases that falls into this category is the control of robots with DNNs, where a single GPU is often sufficient to serve multiple simulations at the same time [328]. This is fundamentally different for SNN models that support online learning and have a complex internal dynamic state, which

cannot be shared across several experiment instances.³ Moreover, simulating them is often more computationally demanding than the environment simulation. The acceleration of neurorobotics experiments that are based on SNNs is therefore a strong use case for the neuromorphic processors introduced in Chapter 2. By default, the NRP supports SpiNNaker [180] through its PyNN interface [289]. In this section, we introduce an extension for the neuromorphic processor Intel Loihi and its Python-based programming interface *Nx Net API* [329]. The new framework works with both locally attached Loihi devices and off-site systems in the cloud.

Requirements and Architecture

The *Nx Net API* is part of *Nx SDK*, Intel’s software development kit for Loihi. Even though it is based on Python, the required software environment differs considerably from that provided by the NRP, which precludes a direct integration. For this reason, the preferred way to connect the two systems is to provide a network-based interface that handles all communication between the robot simulation and the brain simulation. This also makes it possible to connect to remote Loihi systems in the cloud. The interface needs to support two types of operations:

- *Synchronization*: The neural network simulation on Loihi needs to be synchronized with the NRP’s global time step Δt to ensure that both simulations progress at equal speed independently of the actual computation time. As explained in Chapter 3, this is accomplished by executing each of them in chunks of length Δt .
- *Data Transfer*: Input and output of the neural network simulation must be made available in the TFs of the NRP experiment.

Both features have been realized based on a client-server architecture that is depicted in Figure 4.12. In the schema, there is only a single NRP node for both the front end and the back end because this is the most typical setup for local installations. Support for the distributed setups from Figures 4.5 and 4.6 is technically feasible but requires additional user management to control access to Loihi, which is outside the scope of this work. The core component of the *Nx Net API* is the NRP Loihi server application on the Loihi compute node that contains the description of the brain model, controls the simulation and exchanges data with the NRP node through a Python socket. A client application running on the NRP node connects to the server to send data from the robot and receive the output of the brain simulation in TFs. All communication between server and client is channeled through an SSH tunnel which secures the connection and implements the user authentication for accessing the Loihi system.

³This argument does not apply, for example, to setups where DNNs are approximated by SNNs. However, approaches in this direction typically do not fully exploit the computational features of SNNs and can therefore be considered a specific implementation technique for DNNs. It is further important to note that there are also DNNs that have an internal state due to recurrent connections or specific neuron models.

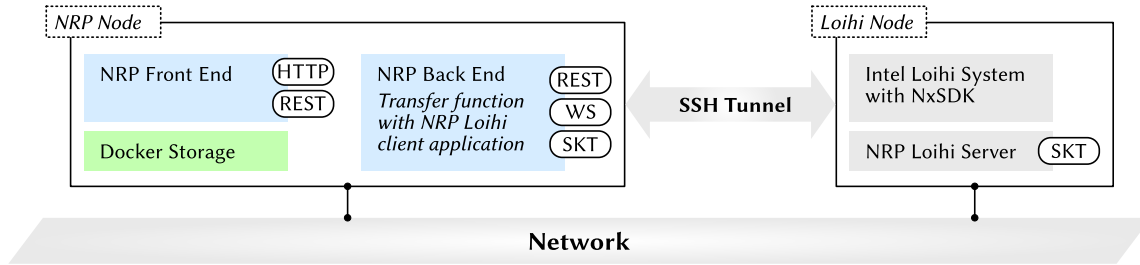


Figure 4.12: Extended system architecture of the NRP with support for Intel Loihi. Symbols and abbreviations are identical to Figure 4.5. SKT refers to a socket connection. The Loihi system is connected to a different node, possibly at a remote site. Communication with the NRP is handled by a transfer function that connects to a server application running on the Loihi node through a Python socket.

NRP Extensions for Nx SDK

The client required for the communication between an NRP experiment and Loihi can be set up in two different ways. In the following two paragraphs, we first develop a method that leverages the TF framework and does not require any changes in the NRP’s code base. Based on this initial prototype, we then introduce an extension of the NRP CLE that adds full-featured native Loihi support.

Custom Integration with Transfer Functions Since the TFs of the NRP are based Python, they can implement virtually any desired algorithm. More concretely, they are not limited to simple basic numerical operations on the data exchanged between the robot and the brain model but can contain complex control flows and access functions from imported Python modules. It is therefore possible to implement the complete Loihi client application within a TF. This approach was realized as a proof of concept for the *Braitenberg Vehicle Experiment* that is part of the NRP’s showcase experiments. The setup is comprised of a mobile robot platform placed in between two screens with adjustable colors. It is controlled by a neural network which implements phototaxis so that the robot always drives towards red objects. By adjusting the screen content appropriately over time, the robot can be made to drive back and forth within its workspace. The neural network model is extremely simple and contains only eight neurons (Nx Net *compartments*) and six synapses (Nx Net *connections*). It receives two input signals that encode the presence of red color in the left and right half of the camera image and outputs rotational speeds for the left and right wheels of the robot. Both client and server exchange the corresponding values directly as spikes rates and wheel speeds, which means that no spikes are transmitted. Since all client code is contained within a single transfer function, no modifications of the NRP’s code base were required. However, the overall setup is highly specific to both the NRP experiment and the neural network model. Changes in any of the two will in most cases also require changes in the code for client and server. Flexibility is thereby traded off with comparatively high manual implementation effort. Moreover, some of the NRP’s features such as the spike train monitor are not available with this method.

Extension of the NRP’s Closed-Loop Engine As outlined in Section 3.3, the CLE of the NRP connects and synchronizes the environment simulation and the brain simulation. Because both of them are accessed through open APIs, it is possible to replace them by alternative software packages as long as the required interfaces are implemented. For example, the CLE by default supports not only NEST but also Nengo and SpiNNaker. Neural simulators integrated this way provide full support for all features of the NRP, which makes using them a lot more convenient compared to the custom TF from the last paragraph. Figure 4.13 provides a schematic overview of the interfaces implemented for Loihi. The CLE accesses every simulation via two interfaces, the *control adapter* and the *communication adapter*. They are responsible for controlling the execution of the simulation and the exchange of data through *devices*, respectively. Devices are accessed by transfer functions and include, for example, spike detectors and spike generators. All functionality for providing these features needs to be implemented by the server. As a result, the Loihi control and communication adapters only contain client code for relaying requests from the CLE to the actual Loihi endpoint. This was realized by replacing the basic server application from the last paragraph with a dispatcher that serves both adapters through a Python socket [330]. A special challenge turned out to be the mapping of the Nx Net API to the NRP’s communication adapter interface, which was designed based on the feature set of PyNN. Differently from the initial prototype, the new server application implements efficient synchronization based on *Sequential Neural Interfacing Processes (SNIPs)*, an interfacing mechanism of Nx SDK.

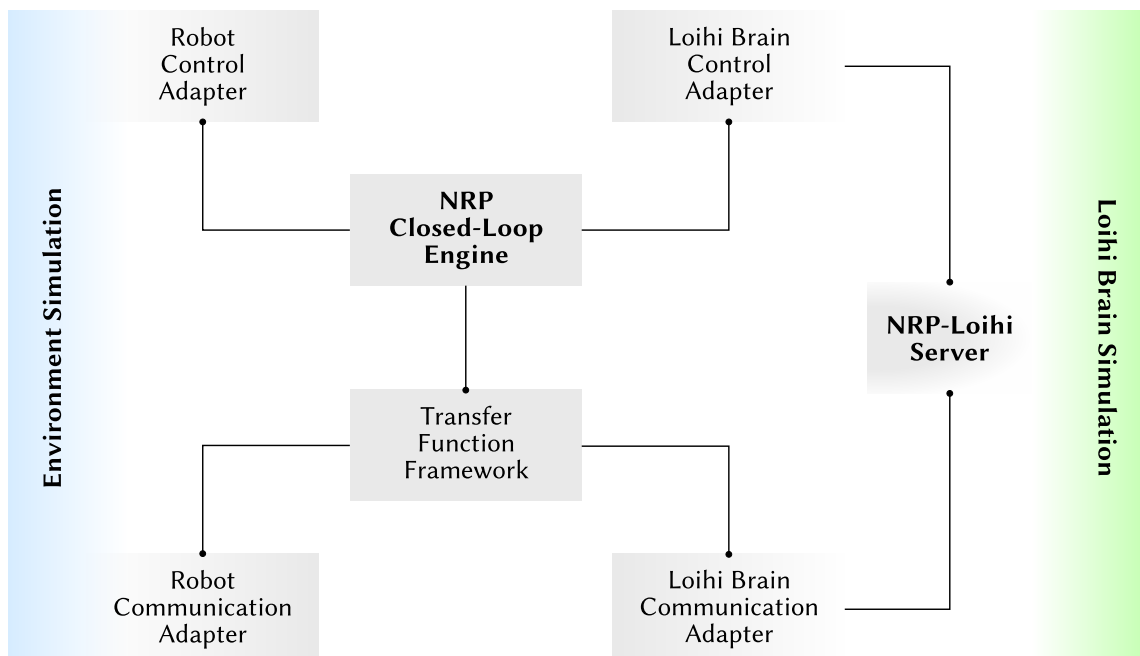


Figure 4.13: Native support for Intel Loihi in the NRP CLE. The adapter components for the brain simulation forward CLE commands and requests to a server component with direct access to Loihi. Only the most relevant components and dependences are shown. Adapted and extended from [330].

With the new extension, Loihi can be used without any restrictions and with the same set of features available for the other brain simulators of the NRP. The barrier synchronization implemented by the chip [62] allows for an efficient stepwise progression of the simulation as required by the CLE. Experiments can therefore run in biological real time if the connection to the server has low latency. This is especially the case when the Loihi system is directly attached to the NRP node and the network socket can be replaced by a Unix domain socket.

4.3 Biomimetic Neurorobotics with the TUM Robot Mouse

The virtual development of models and algorithms with the tools introduced in this chapter augments the exploration space and accelerates development in research and engineering alike. In particular, designing a model or algorithm in a virtual NRP experiment is more efficient than on a physical system, even if fine-tuning is still required to compensate for slight differences between the simulation and the real world. A major challenge, however, is the transfer of results from neuroscience to robotics. Whereas experiments with brain simulations require body models that capture the kinematic and dynamic properties of the bodies of living creatures as closely as possible, work in robotics typically targets simplified mechanical structures that are technically feasible. As discussed in Section 3.1, these differences can have a huge impact on the structure and functioning of the connected brain models. There is consequently a huge potential in optimizing knowledge transfer from neuroscience to robotics by providing a set of complementary body models that cover a full range of use cases from virtual neuroscience to neuromorphic engineering. The NRP includes a realistic musculoskeletal model of the mouse that mainly targets applications in neuroscience. This not only because a huge body of findings in neuroscience is based on the mouse [331], but also because highly detailed atlases of the mouse brain are available [332, 333, 334]. When this model is complemented by an appropriate robot as illustrated in Figure 4.14, a principled workflow for transferring results from neuroscience to robotics becomes available. This section introduces the design goals and main components of the first version of the *TUM Robot Mouse* that laid the foundations for a full series of small-sized biomimetic robots with mouse-like morphology.

Design Goals

A central design goal for the TUM Mouse Robot was to build a *small* and *inexpensive* research platform from easily accessible components that captures the most characteristic morphological features of the mouse. The rationale behind this approach is twofold: Aiming for a small overall size not only makes the robot more similar to its biological counterpart but also constrains the design space in a favorable way. Focusing on low-cost components allows for rapid design iterations and makes it possible to build multiple

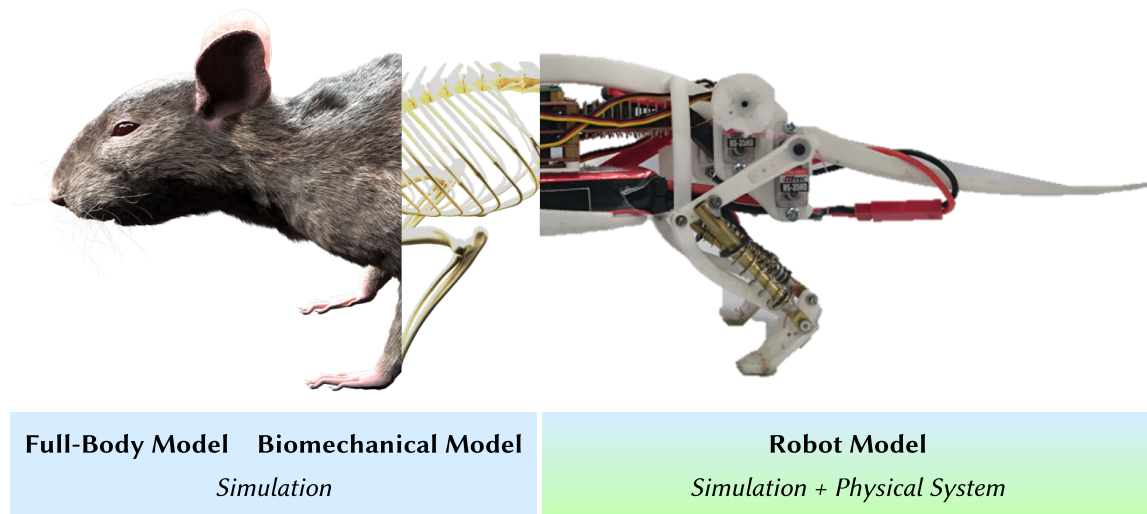


Figure 4.14: Vision for the TUM Robot Mouse. The robot complements the musculoskeletal mouse body simulation provided by the NRP and can be made available both as a physical device and as a simulation model, enabling a principled workflow for knowledge transfer from neuroscience to applications in robotics and neuromorphic engineering. Photo of the robot adapted from [335].

robots with custom modifications. In summary, these considerations lead to the following strategy for the development of the robot mouse:

- *Biomimetic Design:* The robot not only mimics the size and appearance of the mouse body but also selected features of its biomechanics. This makes it possible to leverage embodiment effects and provides guidance for reducing the size of the robot’s actuation system.
- *Legged Locomotion:* To reduce complexity and achieve the targeted size of a mouse, the initial version of the robot is designed with a focus on basic legged locomotion. This will provide a basis for later iterations to add sensors and refine mechanics for other tasks such as grasping and jumping.
- *Rapid Manufacturing:* All custom parts of the robot are built with rapid manufacturing techniques such as 3D printing to enable the implementation of innovative designs with low manual assembly effort and high reproducibility. Another opportunity of additive manufacturing is the procedural design of kinematic structures [336].

The use of rapid manufacturing methods opens up an interesting perspective for the study of embodiment. Low material costs and the simplicity of generating new parts on demand enables research on how morphological changes interact with neural processes directly on a physical system. Examples include the modeling of developmental progression (e.g. body growth) or abrupt changes (e.g. lesions).

Keeping the size of the robot mouse small and lightweight offers many advantages beyond a close correspondence to the body morphology of biological mice. It provides the

prerequisites for high agility and versatility, making the robot suitable for a broad range of tasks and environments. Importantly, these specific features are also relevant in many industrial applications, such as inspection tasks in rough terrain. The size constraints also have immediate effects on the system's electronics. Traditional microcontroller units (MCUs) with appropriate specifications in terms of both size and power consumption offer only very little computing power far below the requirements of typical AI models and brain simulations. This makes the robot mouse a unique testbed for neuromorphic systems like SpiNNaker or Intel Loihi.

Core Components and Initial Version of the Robot

The basic conceptual design of the robot mouse as a biomimetic legged walking robot sets a clear direction for the initial development phases. In an extensive study of the limb kinematics of small mammals such as rats, Fischer et al. [337] highlight that besides the legs also spine movement plays an essential role in walking. This insight sets the frame for the development road map of the robot, which is centered around the design of a biomimetic hind leg and a flexible actuated spine.

Design of the Leg There is a plethora of research on biomimetic legged locomotion ranging from insect-like hexapods to mammalian and amphibian quadrupeds to humanoid bipeds [338, 339]. Not less diverse are the mechanisms and materials that have been proposed for mimicking the properties of biological musculoskeletal systems. Of special relevance for the leg of the robot mouse is the *pantograph* design conceptualized by Witte et al. [340], which is based on observations from the study of gaits of small mammals. Of particular relevance is the finding that the *femur* (proximal/upper link) is in *matched motion* with the *metatarsus* (distal/lower link), which means that they are effectively parallel to each other [337]. This kinematic invariant can be leveraged to reduce the complexity of the mechanical design and makes it possible to build a complete leg from only three links and two actuators. The parallel configuration of femur and metatarsus can be realized with passive elastic elements that add compliance to the system, which, for example, improves walking on uneven ground. The starting point for the design of the mouse leg is the pantograph leg realized for the robot *Cheetah-cub* [341]. Figure 4.15 depicts three substantial development stages. Early prototypes were built from laser cut acrylic plastic and rubber bands [342]. Movements are generated by motors that twist a set of strings around each other to shorten their overall length and in turn apply a retracting force to the leg at their attachment point. These *twisted string actuators* are specifically designed for lightweight robots and were applied earlier in robotic hands [344, 345]. With the twisting of the string mainly requiring high rotational speed rather than high torque, this actuation principle seems very well suited for size-constrained applications but is challenging in terms of control. The initial version of the leg depicted in the left part of Figure 4.15 suffered from issues with the routing of the rubber bands that could be mitigated in a second 3D printed prototype that is shown in the middle. In summary, however, the overall design turned out to be challenging to realize at the desired level of

robustness and maturity. For this reason, the third version of the leg shown on the right of Figure 4.15 is based on a more traditional mechanical design with springs, servomotors and a force sensor at the bottom of the foot. It was successfully integrated in the first version of the robot mouse [335].

Design of the Spine As noted earlier, the spine plays an essential role in biological locomotion. So far, detailed biomimetic models have been mainly developed for humanoid robots such as ECCEROBOT, Roboy or Kengoro [346, 347, 348]. Karakasiliotis et al. [349] developed a salamander-like robot with an artificial spine that was optimized to reproduce biological motion data from cineradiographic recordings. A drastically reduced design was proposed by Weinmeister et al. [350] who extended the Cheetah-cub robot with a spine comprised of a single compliant joint for improved turning motion. The goal for the robot mouse was to identify a mechanism that strikes the right level of balance between this reductionist approach and the highly detailed spine models developed for humanoid robots.

One of the first prototypes is shown in Figure 4.16. It is built from 3D printed vertebrae that are lined up on a steel cable routed through holes in their centers. The shapes of the individual vertebrae are derived as approximations of the actual biological morphology with variations that occur along the extent of the spine from head to tail being neglected. Of special importance are the vertebral discs placed between adjacent vertebrae because they act as passive joints and make the spine compliant. In the artificial spine, they are replaced by silicone that is cast between the vertebrae. This approach is highly advantageous because the silicon not only mimics the functioning of the vertebral discs but also of the ligaments that surround the spine and provide both compliance and structural integrity. The fully assembled spine is actuated through nylon wires that are routed through holes at different positions in the vertebrae. Besides the servomotors depicted in the figure, additional tests were conducted with Nitinol-based *shape memory alloys* that change their

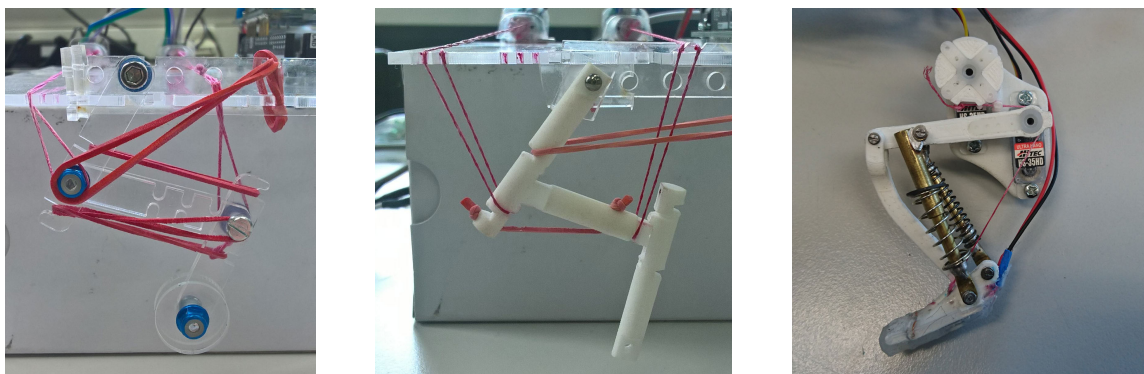


Figure 4.15: Hind leg designs for the TUM Robot Mouse. *Left:* An early prototype made from laser cut acrylic with rubber bands and twisted string actuators [342]. *Middle:* Refined design with 3D printed components [342]. *Right:* Final design for the first version of the robot mouse with springs and servomotors. [343]. Images adapted from [342] and [343].

shape depending on the temperature. Even though movements with this type of actuator are too slow to actively support walking, the extremely compact size of the overall system make this design an interesting option for other applications where speed is less relevant.

Initial Version of the Robot and Outlook The leg design presented in the last paragraph forms the basis of the first version of the TUM Robot Mouse shown in Figure 4.17. While there is no spine yet, there are forelegs with a dedicated design. Walking is implemented with an open-loop controller that runs directly on the robot. One of the main challenges in the completion of this first prototype turned out to be the assembly of the compliant components. While the main parts of the leg are 3D printed, the spring mechanism is manually crafted and cannot be easily replicated. The same applies to the spine model, where the casting of the silicon and the routing and fixation of the nylon wires require lots of care to achieve good results. Moreover, the durability of the overall mechanism is not clear yet since the prototype was deformed after a force test. A highly promising direction for the future development of the robot is to print components that not only have the desired geometry but also the required material properties. This idea has been realized in two further prototypes of the leg and the spine that are depicted in Figure 4.18.

The new leg is a single monolithic component that is ready for use as soon as the printing process is finished. A new design replaces the springs and joints by 3D printed compliant structures. The same principles also enable the design of an alternative mechanism for the spine, which is shown on the right side of the figure. Of course, there are many more opportunities to enhance the TUM Robot Mouse. The initial version presented in this thesis provides a promising foundation for future developments. In

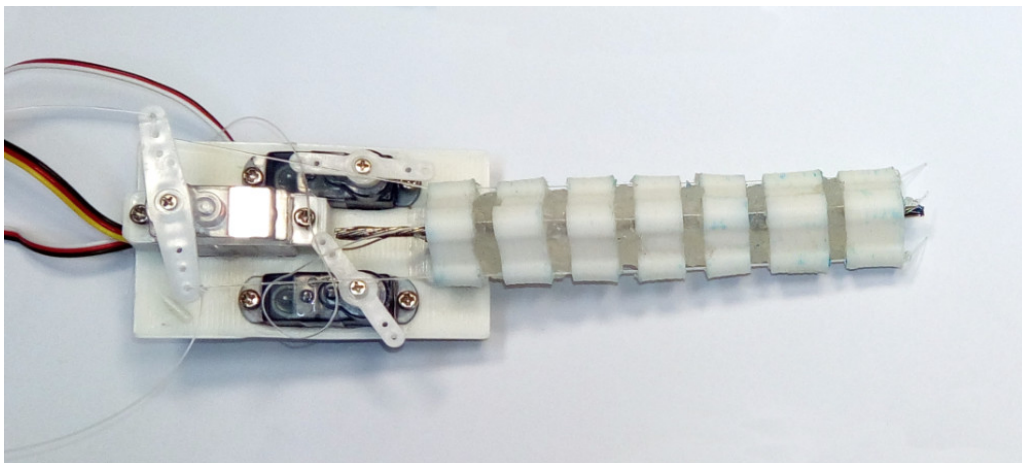


Figure 4.16: Prototype of the spine for the TUM Robot Mouse. Individual vertebrae are 3D printed and lined up on a steel cable that is routed through holes in their centers. Smaller nylon threads routed through the outer parts are attached to servomotors. Silicone casted between the vertebrae fixates the spine's structure and makes it flexible. Image adapted from [351].

fact, there are already new versions of the robot available that incorporate improved components based on those from Figure 4.18 [352]. Further directions of research include the integration of advanced sensors and the development of sophisticated perception and control algorithms with support for learning. First work in this direction has been already completed. Figure 4.19 depicts a prototype of an actuated head with cameras.

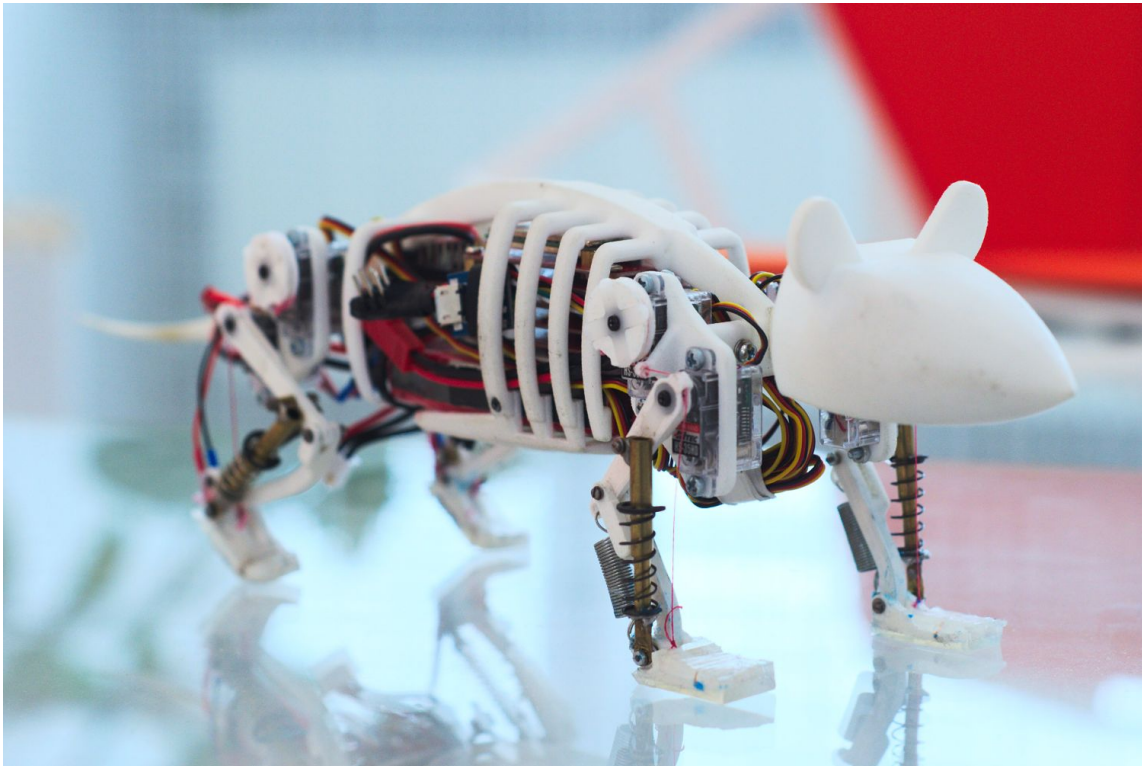


Figure 4.17: Initial version of the TUM Robot Mouse. The photo depicts the first version of the robot which does not contain a spine yet [335].

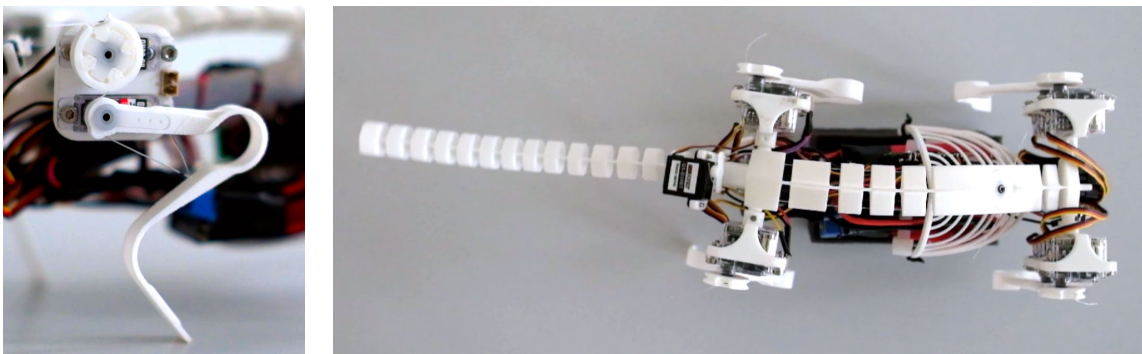


Figure 4.18: Fully 3D printed leg and spine in a refined version of the TUM robot mouse. The images are adapted from a video by Kok Choong Ng.

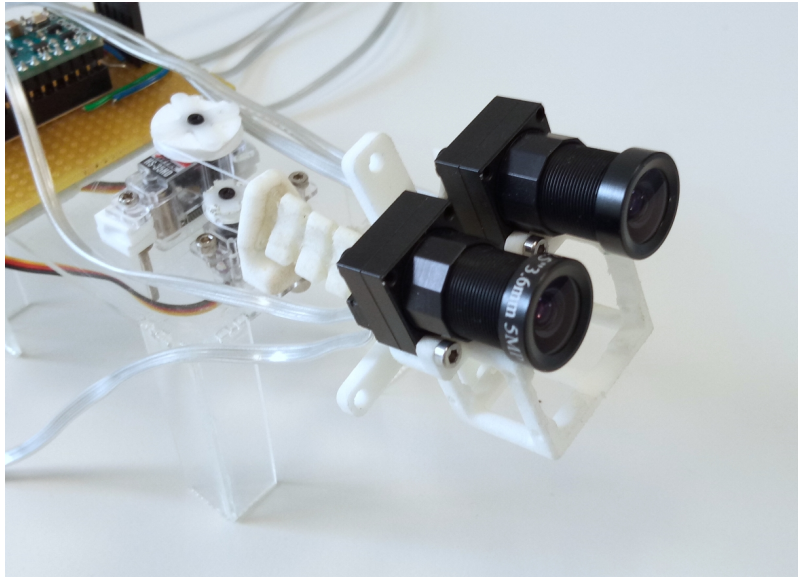


Figure 4.19: Prototype of an actuated head with cameras for the TUM Robot Mouse.

5

A Brain-Derived Modular-Hierarchical Neural Network Architecture

Neurorobotics provides the scientific and technological framework that enables the study of brain function in the context of directly observable behavior of an embodied system in a concrete environment. Connecting a simulated brain to a simulated body in a neurorobotics experiment greatly enhances neuroscientific studies and is the foundation for fully virtualized closed-loop neuroscience. The main motivation underlying this type of study is to investigate the *emergent* properties of a brain-body system. Results can be compared to biological ground-truth data to validate the underlying brain simulation or to predict the influence of a specific set of parameters. While this approach can also yield insight for new brain-derived models and methods in AI and robotics, the brain simulations considered in neuroscience are in general too complex and too specific to be used directly in technical applications. In particular, much of the detail especially in data-driven neuroscientific brain models might not always be required to achieve the desired functionality.

The complexity of biological neural networks makes meaningful abstractions essential when transferring insights and principles from neuroscience to AI. As already mentioned in Chapter 1, this is best illustrated at the example of an aircraft: Building a machine that can fly was not accomplished by replicating the full morphology of a bird's body. Instead, the unique profile of the wings turned out to be the most essential feature that enables both birds and aircrafts to fly. ANNs and in particular DNNs are an outstanding example of how the adoption of only a few basic principles from brain organization can already result

in an extremely powerful computational tool. At the same time, the open challenges in terms of efficiency and robustness faced by state-of-the-art DNNs discussed in Chapter 1 also highlight that biological brains feature many other mechanisms that have so far been largely neglected. While it can be argued that research in machine learning still takes some inspiration from neuroscience [54], neural networks in the brain are nevertheless organized in a fundamentally different way. Identifying the computationally relevant principles underlying these differences is an open research question.

A striking feature of biological brains that has so far received only little attention in the design of ANNs is their highly diversified anatomical and functional architecture with a wealth of different neuron morphologies and synaptic connection patterns. In this chapter, we argue that this architecture is a key principle of distributed modular information processing in the brain and that ANNs can considerably benefit from incorporating it. The first sections provide a principled review of the architectural design features that allow biological neural networks to process information in a robust modular way and link these insights to current shortcomings of ANNs. Based on these findings, a novel brain-derived modular-hierarchical neural network (MHNN) architecture is introduced in Section 5.2. Drawing on this architecture, we develop a novel training procedure for the self-supervised learning of multisensory maps that is based on a topographic loss function in Section 5.3.

5.1 Functional Specialization and Modularity in the Brain

Table 2.1 highlights that the brain is not an amorphous mass of nervous tissue but an intricately structured and highly interconnected information processing system with multiple levels of organization. Each of them comprises a wealth of further sub-structures and sub-systems, all of which contribute together to the overall cognitive performance. One of the most diverse organizational layers is the level of single cells, which comprises a huge wealth of diverse neuron morphologies. The identification and classification of different neural cell types has therefore become a major endeavor in neuroscience that has started with the seminal work of Cajal and is still ongoing today [353]. An obvious question arising in this context is what the purpose of each of these cell types is or, more concretely, how a neuron's computational properties are related to its physical morphology. This applies analogously to the different brain regions, many of which had already been mapped at the beginning of the 20th century [80, 107]. But even though considerable knowledge has accumulated about the building blocks of the brain and their functional purpose, it is still an unsolved problem how they interact to construct a unified coherent perception of the world. Solving this *binding problem* [354] will not only provide insight into one of the key principles of information processing in the brain, but also shed light on how to design technical systems that are as robust to perturbation and structural damage.

Linking Cell Types to Computational Functions

The classification of different types of neurons in the brain is a complex task. At the most basic level, it is possible to distinguish three different neuron types based on the number of processes emerging from the soma: unipolar neurons, bipolar neurons and, most importantly, multipolar neurons [355]. Especially the latter, however, are highly diverse and require more fine-grained classification schemes. These include not only cell morphology but also electrophysiological and genetic criteria. For example, a state-of-the-art in-depth analysis revealed 38 morphological and 17 electrophysiological cell types in the mouse visual cortex alone [356]. Even though the total number of distinct types of neurons in the human brain is still unknown, it is assumed that there might be hundreds [357].

Neuron Morphology In some cases, the specific properties of a neuron are directly related to its function, which especially applies to retinal neurons or sensory receptors in the skin. But besides these rather obvious cases, most diversity can be observed in multipolar neurons of the central nervous system. A salient feature of these neurons are their complex dendritic trees that relay stimuli towards the soma. Figure 5.1 depicts four examples. The neurons are sampled from four regions of the mouse brain. While the cell bodies seem very similar, the dendrites considerably differ from each other, which

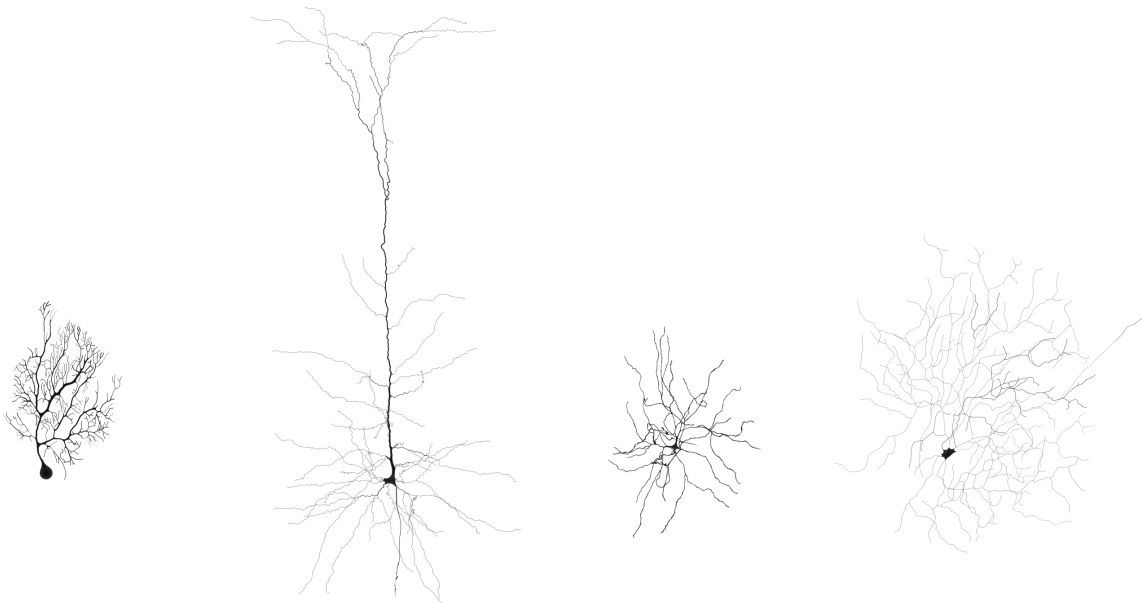


Figure 5.1: Examples of different neuron morphologies in the mouse brain. The most salient feature of every neuron type is its unique dendritic tree structure. *From left to right:* cerebellar Purkinje cell [358], neocortical pyramidal cell in the occipital lobe [359], medium spiny neuron in the basal ganglia [360], retinal ganglion cell [361]. Neuron morphologies were retrieved from *NeuroMorpho.org* [362] and visualized with *NeuroMorphoVis* [363].

raises the questions if dendritic morphology serves any computational purpose. In a simulation-based study, Stiefel and Sejnowski [364] generated dendrite models with genetic algorithms that were tuned to compute the linear summation or determine the order of incoming signals, respectively. Each of the two optimization goals yielded dendrite structures that resembled those of different types of biological neurons. Similar results were reported by Torben-Nielsen et al. [365]. Concrete experimental insights on the role of dendrites have accumulated for cortical pyramidal neurons as the one shown in Figure 5.1. These cells have a distal apical dendrite that receives feedback information from within the brain whereas feedforward signals are received closer to the soma. Since these two signal streams can amplify each other, pyramidal cells are hypothesized to implement an associative learning mechanism that relates internal feedback signals to feedforward sensory perception [366]. Drawing from this insight, Urbanczik and Senn [367] proposed a synaptic plasticity rule where weight updates depend on the electric potential in an extremely simplified single-compartment dendrite model. It is also worth noting that this rule later was employed to derive a biologically plausible approximation of the backpropagation algorithm [368].

Synaptic Connectivity An arguably more evident purpose of diversified dendrite morphology is the facilitation of different patterns of synaptic connectivity. But spatial proximity is only a necessary precondition for the formation of a synaptic connection and not a sufficient one because molecular processes make neurons selective for specific targets. Even the spatial location on the target neuron where the actual synapse is formed is not chosen arbitrarily, which becomes most evident in layer-specific connectivity patterns in the cortex [369]. This directed neural growth is mainly guided by molecular processes that dominate early developmental phases [370]. In the large-scale cortical microcircuit reconstruction by the BBP, the diversity of neurons and connections was reconstructed digitally by first capturing exact cell morphologies from biological tissue. They were then cloned with validated statistical variations of branching angles and section lengths. The actual circuit model was built by placing predefined ratios of different neuron types inside a volume element before assigning appropriate morphological and electrophysiological profiles [59]. Synapses were finally created algorithmically by enforcing constraints that resulted from a set of interrelated anatomical properties which had been identified in the cortical microcircuit [371]. While this reconstruction has been shown to yield realistic synaptic connectivity, it does not capture the actual biological processes that give rise to it. In the brain, the first phase during which molecular processes drive the formation of neurons and synapses is followed by an activity-driven phase, where sensory perception “initializes” this scaffold connectome by adapting connection strengths and pruning synapses [370, 372]. These observations give clear proof of the fact that embodiment is an essential non-negligible constituent of learning processes in the brain, an insight that will crucially guide the methodology developed in this and the next chapter. They also play a key role in the theory of *neuronal group selection* or, as it is often referred to, *neural Darwinism* put forward by Edelman [373]. The first two of the organizational principles that it postulates for the formation of biological neural circuits closely correspond to the

molecular phase and the activity-driven phase, respectively. Remarkably, Edelman [373] recognized the importance of simulations to validate his theoretical predictions and even conducted neurorobotics experiments.

Receptive Fields The synaptic connectivity of a neuron largely determines its *receptive field*, the “portion of sensory space that can elicit neuronal responses when stimulated” [374]. Receptive fields establish a direct connection between a neuron’s responses and the sensory input space. More concretely, this means that, for example, a sensory neuron responds to mechanical stimulation of a certain area of the skin. This concept also translates to cells that are more distant from sensory input. One of the arguably most well-known studies on this topic was a series of experiments by Hubel and Wiesel [97], who recorded activity in the visual cortex of the cat. They identified neurons with receptive fields that are sensitive to light stimulus patterns with specific shapes and directions. Receptive fields therefore give clear proof of functional specialization in single neurons. It was shown that near sites of sensory input the structure of receptive fields is closely related to the morphology of the neurons’ dendritic trees [375, 376, 377]. According to measurements from dendrites of cortical neurons, however, there is no universal correspondence between structure and function. In fact, the observed neurons received input that would have allowed for the formation of whole range of different receptive fields, which suggests a significant computational role of individual neurons [378]. There is also evidence that sensory stimuli perceived during a critical period after birth have strong influence on receptive field formation in cortical neurons [379].

Grandmother Cells One of the probably most compelling evidences for functional specialization of single neurons is the giant Mauthner cell of the teleost fish. When this neuron is removed, animals lose a very specific escape behavior and have less chances of survival when exposed to predators [380]. This result is insofar remarkable as that it confirms the existence of mappings between neurons and very concrete functions. Such correspondence is also often claimed to exist in the brains of primates in the form of the “*Grandmother Cells*” [381]. As indicated by the name, these cells refer to a hypothesized special class of neurons that encodes the presence of a high-level concept, such as a specific face. Experimental evidence in favor of this hypothesis was reported Quiroga et al. [382] who indeed identified neurons in the human cortex that were activated by specific persons and objects. Intriguingly, they responded independently of the visual appearance of the stimulus and also recognized drawings or written names. On lower levels of abstraction, other neurons have been found that are sensitive to object classes [383], color [384] or orientation [97]. Similarly, also neurons in the auditory system specialize for preferred stimuli [385] and the human auditory cortex has been shown to be tonotopically organized [386]. Another seminal contribution was the discovery of *place cells* [387], *head direction cells* [388] and *grid cells* [99] that are assumed to encode space for orientation and navigation. One must, however, not conclude from these results that the representation of information in the brain is fully understood. They are rather small pieces that still need to be fit together by an integrative theory.

Segregated Information Streams across Brain Regions

That specialization plays a role in the overall architecture of the brain already becomes evident from its diversified anatomical structure. Many of its components are discernible by mere visual inspection, and in some cases it is even possible to draw direct conclusions about their functions. This is best illustrated at the example of the *medulla oblongata* that connects the brain and the spinal cord. Another salient component of the brain is the *cerebellum*, which plays an important role in motor control.¹ Even though there are many more anatomically distinct components, the mapping to corresponding functions is usually less clear and in particular not evident from the anatomical structure. This is especially true for the mammalian cortex, which is thought to be the center of all higher cognitive functions.

Cortical Regions The first in-depth analysis of the cortex was provided by Brodmann [80], who created the cell type atlas presented at the beginning of Chapter 2. Remarkably, some of the brain regions that he determined solely based on microscopic anatomical analysis later turned out to be centers of specific functions. But the discovery of these correspondences required methods that allowed for relating observed behavior to those parts of neural tissue that give rise to it. In the early days of neuroscience, the only source for this type of information were *brain lesions*. In lesion studies, the parts of the brain that are involved in a cognitive function are inferred indirectly by determining the behavioral effects of the loss or damage of a part of the brain. The starting point for this area of research was the identification of a brain region involved in language processing by Pierre Paul Broca in 1861 and was later followed by similar findings on “memory, hemispheric specialization, emotion, vision and motor control” [390]. Even though modern neuroimaging techniques such as fMRI and other recent methodological developments have the potential of replacing lesion studies, it is argued that lesions still can provide unique insights that can complement alternative measurements [391].

The overall architecture of the cortex shares the same basic structure across all mammals [392]. Nevertheless, the concrete parcellation into differently sized cortical areas is individual to every species. Even though the cytoarchitectonic properties applied by Brodmann [80] to identify distinct regions are still valid today, more recent mapping efforts leverage multimodal data including connectivity, neural receptive fields, topographic input space representation and function [70, 393]. Notably, connectivity data have turned out to be particularly suited for this task. A general observation in cortical organization is the spatial proximity of functionally related areas [394]. The main areas are spatially grouped by modality into the *visual cortex*, the *auditory cortex*, the *somatosensory cortex* and the *motor cortex* [395]. Each of these cortical structures is subdivided in several regions that

¹It should be noted that it has more recently been hypothesized that the cerebellum also contributes to cognitive tasks. Interestingly, Koziol et al. [389], the authors of a review that summarizes support for this claim, note that the confinement of the cerebellum to movement tasks also reflects traditional thinking that separates mind and body.

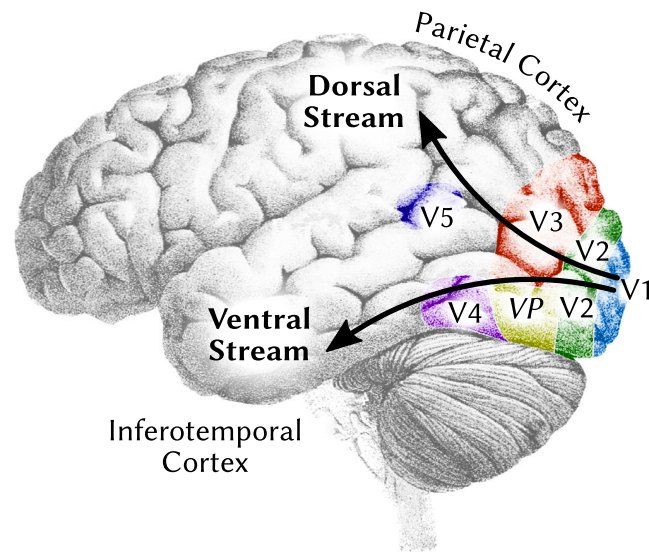


Figure 5.2: Illustration of the dorsal and ventral visual pathways. The ventral stream processes information related to object appearance (*what?*) and flows through $V1 \rightarrow V2 \rightarrow V4$ towards the inferotemporal cortex. *VP* is only shown for completeness. The dorsal stream process information related to object location and motor control (*where?* and *how?*). It flows through $V1 \rightarrow V2 \rightarrow V3 \rightarrow V5$ towards the parietal cortex [397]. Only feedforward connectivity is shown. Cortical regions are highlighted qualitatively based on Tootell et al. [398] and Logothetis [399], the drawing of the brain is a modified version of the original version by Brown [72] which is in the public domain of the United States (PD-US-expired).

process increasingly abstract information. This is best illustrated at the example of the visual cortex that is partly sketched in Figure 5.2: Information enters in area *V1* and then proceeds to subsequent areas, including *V2* and *V3*. In general, receptive field sizes of cells increase and represented stimuli become more abstract along the path [394]. As described in the last section, neurons in early processing stages of the visual cortex respond to very basic forms and shapes while neurons in higher cortical areas may even be sensitive to specific faces. Another characteristic of cortical regions distant from sensory input is the processing of multiple sensory modalities, which is reflected in more diverse connectivity. These parts of the cortex are commonly referred to as *association areas* [394].

Cortical Pathways A direct consequence entailed by the subdivision of the cortex into multiple regions is the existence of corresponding pathways that interconnect them. One of the most prominent findings in this direction was the discovery of a *dorsal stream* and a *ventral stream* in the visual system by Mishkin et al. [396]. Figure 5.2 depicts them together with the involved brain regions. Both pathways originate at the primary visual cortex *V1* and start to diverge after *V2*. On the way towards inferotemporal cortex and parietal cortex, respectively, they pass through different cortical areas with specific functional properties. As a result, each pathway specializes on processing a different type of information: The dorsal stream processes position, motion and speed for enabling the localization of objects, whereas the ventral stream processes form, color and texture for

object perception [394]. Like for individual brain regions, lesion studies played a central role in the identification of these functions. Ventral and dorsal pathways have also been identified in other cortical systems such as the auditory cortex [394, 400]. Moreover, there is also evidence for even more fine-grained specialization in the visual system with separate processing of form, movement and other properties [401]. It is interesting to note that sensory input seems to be converted from physical measurements to qualitative perceptions in the hierarchies along the streams. A study of the human visual system with fMRI has revealed that color as a subjective perception only occurs in higher cortical areas whereas those close to sensory input only represent the actual physical input stimuli [402]. Clearly, light has no color in a physical sense just like pain does not exist outside the living creature that feels it. Moreover, all information processed by the brain is encoded in spike trains independently of its qualitative criteria. From this, one may conclude that information streams encode the actual meaning of a stimulus, a concept that is akin to that of *labeled line codes* [403].

Information Segregation and Binding The findings about regions and pathways highlight two computational paradigms implemented in the mammalian cortex: *hierarchical* and *parallel* processing [393]. This means that information streams along pathways are concurrent and representations become increasingly abstract within each of them on the way from early sensory areas towards association regions. As exemplified by the dorsal and ventral portions of the visual system, every stream computes only a subset of the features that can be perceived from the sensory input. Information thereby becomes *segregated*. Nevertheless, the resulting perception of the external world is unified. This is even more remarkable when considering that we can perceive a multitude of different things at the same time and still each of them is assigned the matching information from the individual streams (e.g. cars with different colors along a street). The question of how this is accomplished has come to be known as the *binding problem* [404]. An evident solution to this issue would be combination-coding cells that encode the presence of a specific subset of stimuli (e.g. one cell for each combination of car, color and position). However, such an approach would, of course, only be viable for limited domains since the whole space of perception can be impossibly covered by allocating specialized cells for every combination of attributes [354]. Other theories therefore stress the role of the interactions between multiple neurons to bind features. Binding by synchrony posits that the synchronized firing of neurons links information across different processing streams [405]. Even though experimental data has indeed proved the occurrence of synchronized neural firing, von der Malsburg [354] argues that the temporal resolution of biological neurons is too coarse to rely on synchrony as the only mechanism of binding and proposes that information is represented by interconnected subsets of neurons called *network fragments* [406, 407]. In this model, it is the active connectivity between neurons that defines the binding, from which it follows that connections can be switched instantaneously to activate different network fragments.

The Connectome The theory put forward by von der Malsburg [406] highlights the importance of the brain’s connectivity, the *connectome* [408]. So far, pathways have been treated as discrete entities that operate largely independently to compute distinct functions. The actual anatomical connectivity scheme is considerably more complex. Connection patterns between cortical regions are typically symmetric and there is considerable cross talk between different processing streams [393]. As DeYoe and van Essen [409] point out, there is furthermore no direct mapping between basic sensory features and individual streams, since every of them processes data that contribute to the function it computes. This observation is supported by more recent experimental findings which revealed that tactile input stimuli can be decoded equally well by neurons in both the somatosensory cortex and the primary visual cortex [410]. It is also known that the cortex can compensate for damage of neural tissue (e.g. after a stroke) or changes in sensory input (e.g. because of blindness) through dynamic reorganization [411]. From these examples it becomes clear that the connectome is by no means a simple structure. Its complex anatomical architecture enables the emergence of diverse activity patterns, which are also referred to as *functional connectivity*. In fact, the functional specialization of brain regions can be seen as a consequence of the underlying anatomical connectivity: As Passingham et al. [412] have shown, brain regions have unique *connectional fingerprints*, i.e. connections from and to other cortical regions. As we will elaborate in more detail in the next sections, the intricate connectome of biological neural networks is one of the most important features that distinguishes them from ANNs.

5.2 Design Principles of Modular-Hierarchical Neural Networks

Most state-of-the-art neural network models are based on largely monolithic architectures that lack a clear separation of pathways and functions. Even though it has been demonstrated that different sub-tasks of a problem can be localized in distinct parts of a network, there is no reuse of functionality and identical sub-tasks are learned multiple times at different locations [413]. These results highlight a fundamental difference between information processing in the brain and in ANNs: While DNNs can learn *hierarchical* presentations that bear similarities to those in the mammalian cortex [414], they lack the brain’s functional specialization and *modularity* outlined in the section before. A plethora of different *modular neural network (MNN)* architectures have been proposed to address this shortcoming, but none of them has come even close to the success of DNNs. In this section, we develop a novel brain-derived neural network architecture that is both modular and hierarchical. It adopts a modular computational paradigm that is centered around task-based sensor data fusion and combines it with the powerful hierarchical feature representations of DNNs. As a preparation, we first provide an overview of the main features of common MNN architectures.

Modular Neural Networks

Like in the case of ANNs, the development of MNNs was initially motivated by brain research with published work dating back to the end of the 1980s [415]. Gallinari [416] defines MNNs as “systems in which several NN modules cooperate with each other or with other techniques to complete a global task.” Neural network modules are independent sub-networks, each of which realizes a specific functionality such as the computation of a function or the detection of a class of patterns. In particular, individual modules can be based on different models and trained with different learning algorithms, which enables the design of hybrid systems that apply several types of computation and learning at the same time. Practical motivations for investigating MNNs include complexity reduction, model reuse and the incorporation of existing knowledge, fusing data from multiple sources, multi-task learning, and increased robustness [416]. In addition, the localization of concrete functions in distinct modules provides a direct insight into the operation of the system, facilitating the interpretation of its output. Especially interpretability is a key issue in current neural network research and one of the main challenges in the development of safety-critical systems such as automated driving functions.

Design Principles There is a huge body of research on MNNs that has been regularly covered by extensive reviews [415, 416, 417, 418, 419, 420]. In the following, we will therefore only focus on the main design principles for MNNs that have been identified in these contributions. Based on a general conceptual framework proposed by Auda and Kamel [415], the development of a MNN can be divided into three steps:

1. *Decomposition*: The task to be solved is split into sub-tasks for the neural network modules. The decomposition can be defined manually based on a global system architecture (user-defined functions are mapped to individual modules), derived from intrinsic task properties (structure of the input data), or by specifying an algorithm that will be executed before or during training (e.g. evolutionary algorithms [420]).
2. *Training*: The modular component networks are trained individually (loosely coupled) or jointly (tightly coupled) [419]. Depending on the type of modularization, the network topology might be adapted during the training process.
3. *Fusion*: The output of all modules is merged through competitive and cooperative processes that may be realized as simple arithmetic operations or as trainable models [420]. Depending on the type of architecture, modules might be connected directly with each other so that the output of one module becomes the input of another.

At first glance, modularity adds additional complexity to the network design. Rather than focusing on the development and training of a single model, it is now required to specify multiple networks along with their mutual interactions. However, network modules that implement a comparatively narrow sub-task can be typically made considerably smaller

than a monolithic network which learns a complex task *end-to-end*. Moreover, training is faster, requires less data, and every module can be implemented with the architecture and learning rule that is best suited for its task [415]. For example, when parsing the content of a website, images are best processed with a CNN while text is typically parsed with recurrent neural networks. Another advantage of modularity is the clear separation of different domains within the network. This opens up a promising direction of research on continual learning [421] without catastrophic forgetting [51]. In terms of practical implementation, MNNs enable a direct mapping of computational tasks to the target hardware, which is especially important for neuromorphic processors and distributed heterogeneous computing systems. The huge success of DNNs has recently also led to the adoption of MNNs as modeling tools in neuroscience, where they outperformed traditional methods [422].

Basic Modular Neural Network Architectures The main step in the design of a MNN is the decomposition of the task as outlined above. It not only determines the overall architecture of the network but also applicable methods for the subsequent training and fusion phases. *Ensemble models* constitute the arguably most basic class of MNN models. They are based on the fact that when training a set of neural networks on an identical problem, each of them will arrive at a different solution due to variations during the learning process (e.g. random weight initialization, stochastic weight updates, etc.). Depending on the task, the aggregated ensemble output is then computed from the responses of the individual networks by applying operations such as majority voting or averaging. It has been shown that the resulting generalization error is never larger than the average error of the ensemble networks [423]. Clearly, this approach is only effective as long as there is a sufficient amount of diversity within the ensemble. The *Mixtures of Local Experts* model by Jacobs et al. [424] addresses this issue with a training procedure that assigns different sub-tasks to expert networks. A key element of this architecture is a separate gating network which combines the outputs of the individual experts. It could be shown that the architecture is capable of automatically separating the *where* and *what* components of an object localization and detection task [425]. Remarkably, each sub-task was mapped to the expert network with the most appropriate architecture. Both ensemble and mixtures of experts models form the basis of many other architectures. Another important direction of research is the application of evolutionary algorithms to automatically synthesize MNNs for a given task. Examples include the use of graph grammars to generate MNNs for gait control of six-legged robots [426] and the learning of modular action policies for the computer game *Ms. Pacman* [427].

Modular Deep Neural Networks Even though DNNs are mainly based on monolithic architectures, the hierarchical feature representations learned by them can be seen as a basic form of modularity. Since especially low-level features are identical across related tasks, the corresponding network layers can be reused to accelerate training [428]. More fine-grained reuse within network layers is realized in a modular layer proposed by

Kirsch et al. [429]. It is comprised of a library of modules and a controller that activates a subset of them depending on the current input. In the work by Tani [245] on *Multiple Timescales Recurrent Neural Networks*, individual modules in a hierarchy of recurrent neural networks are coordinated by the relative time scales of their internal dynamics. Modules representing higher levels of abstraction run at slower time scales and provide context for lower levels. The concept was implemented in a neurorobotics experiment where a humanoid was able to learn new high-level action sequences without changing connection weights in a subordinate network for basic motion primitives. Addressing static image data rather than dynamic motion patterns, Cireşan et al. [430, 431] applied traditional ensemble methods with averaging to form committees of CNNs. At the time of publication, their model achieved best-in-class results in handwritten character recognition and traffic sign classification. A fundamentally different method for designing modular DNNs was introduced by Andreas et al. [432]. Their *Dynamic Module Neural Network* dynamically composes modules to compute answers for natural language questions on a data base or on images. Based on a fixed set of module networks that correspond to sub-tasks of the question answering process (e.g. locating an object in an image), the system combines traditional semantic parsing with reinforcement learning to automatically generate a MNN that computes the answer to the input question. Another line of research addresses the incremental modular augmentation of neural networks by adding modules as new tasks are learned. The *Progressive Neural Networks* introduced by Rusu et al. [433] are extended by adding new columns as laterally connected parallel layers next to the ones of the original network. When learning a new task, only the weights of the corresponding column are adapted. The method was successfully applied to transfer a model trained in simulation to a physical robot [434]. A similar model was developed by Terekhov et al. [435]. Modularity also plays an important role in reinforcement learning for robotics. For example, Devin et al. [436] presented a method for learning modular action policies which enables reusing learned task knowledge on different robots by replacing a corresponding task-specific part of the network. It is important to note, however, that DNNs in reinforcement learning can in general be replaced by other function approximators. The modularity is therefore not a particular feature of the underlying network architecture.

Multimodal Modular Neural Networks Systems that process information from multiple sensory modalities such as vision, hearing or proprioception provide a natural starting point for the definition of MNN architectures. The separation of individual modalities into different modules results in unimodal areas that need to be integrated by an association mechanism. More technically, this means that multimodal MNNs implicitly or explicitly realize some form of data fusion. A key research question in this context is the learning of joint internal representations for multiple modalities. One of the first DNN models addressing this question was conceived by Ngiam et al. [437] who constructed a *bimodal deep autoencoder* model from a *bimodal DBN*. The latter in turn was comprised of two modality-specific restricted Boltzmann machines (RBMs) that were trained individually on video and audio data. While the system could not outperform an architecture optimized

for a single modality, performance was better compared to the fully unimodal RBM baseline. Interestingly, the authors could demonstrate cross-modal interaction by replicating the *McGurk effect* [438]. One of the main reasons for the limited performance of the model was that not all of the learned features were multimodal. *Correlational neural networks* address this shortcoming with a similar autoencoder architecture but employ standard backpropagation with a loss function that maximizes the correlation between the latent space representations of the involved modalities [439]. Droniou et al. [440] proposed an architecture that clusters the input data and at the same time learns a low-dimensional manifold representation for each cluster. By sharing the network layers for clustering and manifold projection among multiple modalities, the system could learn multimodal representations. Feature extraction was realized with an autoencoder that preprocessed the input data. A widely used approach that can be seen as a direct extension of standard deep CNNs are architectures with multiple parallel processing streams which merge near the output layer. For example, Eitel et al. [441] presented a late fusion object recognition network comprised of two separate streams for image and depth data. Each of them was trained individually before joint training in the final network. Rathi and Roy [442] developed a multimodal SNN architecture with an unsupervised STDP-based learning rule. Cross-modal correlations were detected by connections between the inner layers of two three-layered unimodal networks.

Multimodal Architectures Based on Self-Organizing Maps SOMs are an alternative to autoencoders that is especially common in bioinspired models [127]. They are typically implemented as two-dimensional grids of neurons that learn a topological mapping of the input space by means of a competitive process with local facilitation and global inhibition. In this sense, they can be interpreted as cortical maps that self-organize to represent their respective sensory input modalities. Lalleo and Dominey [443] proposed *multi-modal convergence maps* as an extension of SOMs to implement the convergence-divergence zone theory by Damasio [444] and study the learning of body schemata in robotics. Individual sensory modalities were represented by unimodal maps that projected onto a shared amodal map. In a series of experiments, they demonstrated the elicitation of motor response after stimulating the learned maps and cross-modal effects such as improved visual recognition of the robot's hand through the inclusion of proprioceptive data. Visual input was encoded as a raw vector of pixels or preprocessed by a pattern matching system. The model by Axenie et al. [445] employs SOMs to learn individual maps for sensory input channels. The maps are fully connected with each other and learn correlations with a Hebbian plasticity rule. Based on information theoretic measures, an algorithm automatically determines which modalities are associated with each other directly from raw input signals. The system was evaluated on ego-motion estimation of a quadrotor. More recently, Khacef et al. [446] proposed a new SOM variant that is inspired by the theory of reentry by Edelman and Gally [447] that was mentioned earlier in Section 2.4. It is based on two SOMs and an algorithm that associates correlated units by managing connectivity and updating cross-modal weights with Hebbian learning.

The system was evaluated on a classification task where multimodality was exploited to reduce the required amount of labeled data during training and to increase accuracy during inference.

Architecture Definition

From all the different architecture concepts outlined above, it is evident that multimodal MNNs fit best to the characteristics of both artificial and biological embodied systems. From a robotics perspective, individual modules can be directly matched with the sensors available in the system. From a neuroscience perspective, the identification of modules with sensory modalities reflects the organization of the sensory cortex into distinct modality-specific areas. The three main types multimodal of MNN architectures discussed before can be summarized as follows:

- *Multimodal Deep Autoencoders*: Self-supervised learning of latent representations of the input space with support for feature hierarchies. Learning a concrete task requires additional processing stages.
- *Multimodal SOMs*: Unsupervised learning of topological representations of the input space without direct support for hierarchical feature representations. Learning a concrete task requires additional processing stages.
- *Multistream DNNs*: Supervised learning of concrete tasks. Input data is first processed in separate streams of network layers that merge into a single output stream.

Each of these architectures implements functionality that captures some aspect of cortical processing. Deep autoencoders can learn hierarchical feature representations, SOMs form topological representations of the input space and multistream DNNs fuse sensory information to achieve a specific task. However, there is no architecture yet that implements all three of these characteristics at the same time. Therefore, in this section, we introduce a novel brain-derived neural network architecture called *modular-hierarchical neural networks (MHNNs)* that combines modular and hierarchical processing with task-based data fusion. After providing a formal description of the architecture in terms of its individual network modules and their synaptic connectivity with each other, we present an algorithm for learning lateral connections between network modules based on multimodal input data. A novel algorithm for training the modality-specific component networks will be introduced subsequently in Section 5.3.

Component and Hub Networks The key concept of the new network architecture is an explicit separation between the processing of *sensory input* and the generation of *control output*. While these terms are motivated by robotics applications, there is, in general, no restriction to a particular type of data. Thus, in the case of a standard image recognition task, the output can also be a classification result. Figure 5.3 provides an overview of the main components of the architecture. The separation of input and output

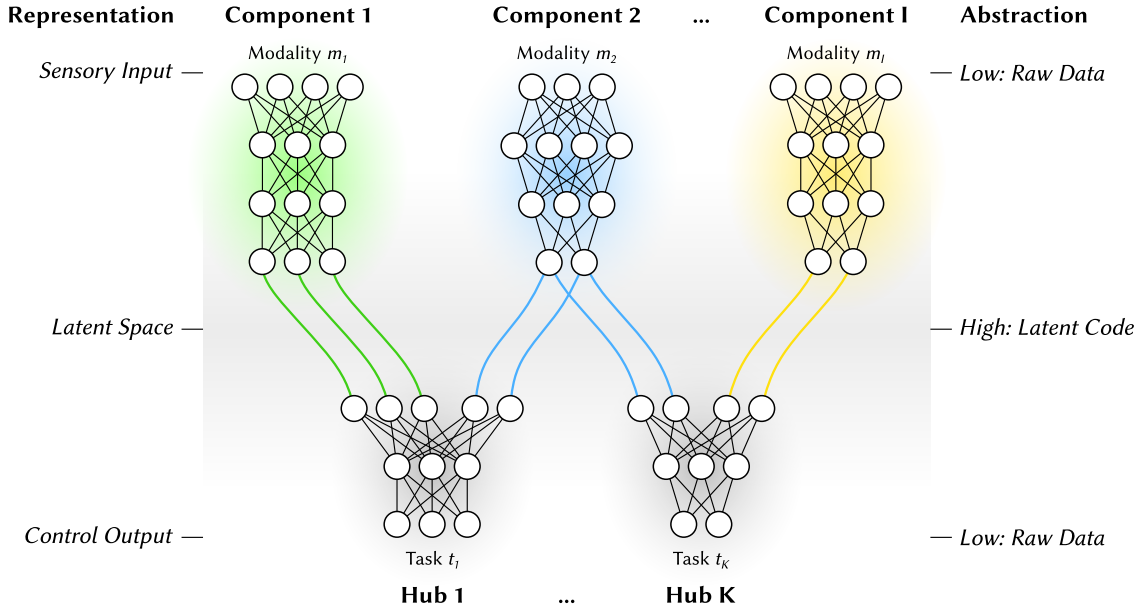


Figure 5.3: Example instantiation of the proposed MHNN architecture. *Component networks* process external inputs (e.g. vision, hearing, proprioception, etc.) and transform them into abstract latent representations. Colors indicate the different modalities processed by the networks. Task-specific *hub networks* fuse latent space data in order to compute the system’s output (e.g. motor control, pattern recognition, etc.). The visualizations of the individual networks serve as illustrations.

leads to a bipartite network topology with twofold modularity. It is therefore possible to decompose the set $\text{Modules}(N)$ of all modules in a MHNN N as follows:

$$\text{Modules}(N) = C \cup H \quad \text{with} \quad C = \{C_1, C_2, \dots, C_I\} \quad \text{and} \quad H = \{H_1, H_2, \dots, H_K\} \quad (5.1)$$

C denotes a set of *component networks*, each of which processes a stream of sensory input data. Following the definition from the problem statement in Section 1.4, the set of all input streams corresponds to a system’s modalities M . A key concept of MHNNs is that there is a bijective mapping $c : M \rightarrow C$ between modalities and component networks, which means that each modality is processed by a dedicated neural network. This specific design choice is actually less restrictive than it might first appear. Each modality can still be processed along different parallel processing streams in a single component network. For example, Rueckl et al. [448] developed a simple neural network model of the dorsal and ventral streams in the visual cortex that predicted both object location and identity. More generally, it is also possible to process a single modality by two completely different component networks at the same time by inserting the corresponding branches as separate streams at the input layer. But, by definition, component networks that process more than one modality at the same time are not allowed. This is because data fusion is implemented by the hub networks H that compute the system’s output. In terms of the definition from Section 1.4, there is exactly one hub network for each task $t \in T$. Analogously to the component networks, this relation can be expressed as a bijective mapping $h : T \rightarrow H$.

The output of each hub network corresponds to a command for the system's actuators A. As stated earlier, the definition is also applicable to non-robotic use cases where A simply denotes the model's output. As illustrated in Figure 5.3, every hub network can be connected to any number of component networks. Solving the task assigned to the network, therefore, implicitly entails fusing the data from all connected modalities. Due to the direct correspondence between hubs and tasks, the sensor fusion is inherently task-based and fulfills all criteria formulated in Section 1.4.

Modular and Hierarchical Processing Despite its conceptual simplicity, the network architecture defined in the last paragraph incorporates many findings from neuroscience. While there is, in principle, no restriction on the implementation of the component networks, the main idea behind the architecture is that they are realized as DNNs that automatically learn hierarchical feature space representations of the input modalities. Unlike in common end-to-end learning tasks, where a monolithic network directly computes the target output from raw input data, the component networks considered here only need to convert the input into an abstract and more compact latent space representation, as indicated in Figure 5.3. Possible network types include not only autoencoders, but also the feature layers of DNNs. Appropriate network architectures can be selected based on the type of input data (e.g. CNNs for images, simple multilayer ANNs for proprioception). The mapping of high-dimensional data streams on the input side of the component networks to compact representations in a latent space corresponds well to hierarchical cortical information processing. CNNs in particular have turned out to be powerful models for predicting the processing of visual input along the ventral stream [53, 449]. The data processed at each layer of a component network becomes increasingly abstract towards its output layer. This direction is reversed in the hub networks, where latent internal representations are transformed to concrete control output. In summary, data processed by the complete network is first converted into an abstract internal representation in the component networks before it is projected back to the system's output space in the hub networks. This hierarchical scheme captures basic principles of cortical processing. At the same time, the combination of multiple component and hub networks allows for the implementation of parallel processing streams akin to the cortical pathways discussed in Section 5.1. As a result, the proposed architecture combines both modular and hierarchical processing in a single brain-derived model.

The notion of hub networks or *hubs* is motivated by findings about the brain's connectome. Studies with tools from graph theory have revealed that networks in biological brains exhibit a *small-world* structure with clusters of highly interconnected areas and sparse connectivity between them [450]. A specific feature of these networks is the existence of *communities* and *hubs*. While the former can be identified with independent modules for segregated information processing, the latter play an essential role in relaying information between different parts of the brain [451]. In a theoretical study of the cortical connectivity in the cat brain, Zamora-López et al. [452] found evidence that hubs also perform multisensory integration or, in technical terms, data fusion. It is important to note

that the nodes of brain network models in neuroscience do not necessarily correspond to individual neurons, but often relate to specific measurement sites or brain regions [451]. At this point, the correspondence between the proposed network architecture and cortical networks becomes evident. The layers of all network modules can be interpreted as cortical regions that, in turn, form the nodes of the graph model. Within in each module, the layers form a community. While this applies to both component and hub networks, only the latter receive and fuse information from multiple network modules, which is very closely analogous to the cortical hubs that have been identified in biological brains.

Connectivity of Hub Networks What is still missing is a specification of the connectivity between component and hub networks. An overview of the general scheme is depicted in Figure 5.3. The shape of the input layer $\text{Input}(H_k)$ of a hub H_k is determined by the output layer shapes of the component networks $\text{Components}(H_k)$ from which it receives data:

$$|\text{Input}(H_k)| = \sum_{C^* \in \text{Components}(H_k)} |\text{Output}(C^*)| \quad (5.2)$$

As a result, every input unit of a hub network can be matched to an output unit of a component network. All subsequent layers of H_k can be freely defined based on the assigned task. The definition of connectivity between component and hub networks can thereby be reduced to the identification of relevant input modalities for each task. Depending on the type of task and the domain knowledge available, there are two different approaches for determining $\text{Components}(H_k)$:

- *Manual Mapping*: Meaningful mappings between components and hubs are obvious in many settings and $\text{Components}(H_k)$ can be specified manually. In a visual servoing task, for example, camera input is mandatory and proprioceptive state information about the robot will considerably ease control.
- *Learning*: Especially for complex compound tasks, the determination of relevant input modalities is not trivial. In particular, there might be more than one viable choice for $\text{Components}(H_k)$ with the best one being dependent on the quality of the data retrieved from the individual component networks. Therefore, the model needs to learn its configuration automatically.

A promising approach for learning the component network set of hub H_k is to start training H_k on its associated task with all component networks C connected. This means that H_k initially receives input from all available modalities. During learning, the weights of the synapses originating from those component networks that are most informative for the task can be expected to grow more than those of synapses that do not provide relevant input. Synapses from components to H_k that are below a defined threshold can then be removed from the network. In fact, *network pruning* is a separate research topic that has gained momentum with the growing size and complexity of DNNs. Blalock et al. [453] conclude in an extensive review that the pruning of network parameters

based on their magnitudes as suggested here “substantially compresses networks without reducing accuracy.” By pruning, component networks with weak connectivity to H_k are automatically removed from $\text{Components}(H_k)$. At the end of the process, H_k is only connected to the most relevant modalities. With the definition of its component network set $\text{Components}(H_k)$, every hub H_k is assigned a connectivity profile that is specific for its associated task. This is in line with findings from the cortex where the “connectional fingerprints” of a region assume a primary role in the definition of its function [454]. Also the synaptic pruning process is directly motivated by findings from neuroscience: Huttenlocher [455] found that the synaptic density in the brains of infants between one and two years is up to 50 % higher than in adult brains. Furthermore, “immature synaptic profiles had a nearly symmetrical appearance” [455]. These results underpin that the connectome of the brain is not a fixed and purely genetically encoded structure, but undergoes considerable development, during which neural pathways are shaped and synaptic density decreases as a result of individual experience.

Lateral Connectivity of Component Networks Data fusion in the hub networks starts from the latent representations generated by the component networks. But as already indicated before in Section 5.1, there is accumulating evidence for cross-modal interactions in early sensory cortices [456, 457]. These findings are in line with the concept of reentrant connectivity between neuronal groups in the group selection theory by Edelman [373]. One of the main features of reentry is that it enables the detection and enforcement of correlations between different sensory streams. Correlated activity contains low-level information about the relationships between modalities. If the typical pattern of correlations during normal operating conditions is known, it becomes possible to detect anomalies and possibly compensate for unreliable input caused by occlusions, changing environmental conditions or sensor failure. A potential drawback of reentry is its strong reliance on recurrent connectivity. On the one hand, this enables rich network dynamics and reciprocal regulation of different regions of the network. On the other, both connections and parameters must be carefully balanced and adjusted, respectively, to maintain a stable level of network activity without exploding bursts of neural activation that self-amplify through recurrent connections. To mitigate these issues, MHNNs are based on a simplified version of reentry without recurrence that is still capable of correlation detection. By omitting recurrent connections, dynamic instabilities such as exploding network activity are completely avoided. Figure 5.4 schematically illustrates the main concept in an example system where a component network for modality m_1 receives lateral input from the networks for m_2 and m_3 . We denote the lateral input to a component network as $\text{Lateral}(\cdot)$. In the context of Figure 5.4, this expression evaluates as follows:

$$\text{Lateral}(m_1) = \{m_2, m_3\}, \quad \text{Lateral}(m_2) = \text{Lateral}(m_3) = \emptyset \quad (5.3)$$

Each synaptic connection from m_2 or m_3 to m_1 can be thought of as a correlation indicator, which means that only neurons with highly correlated activity are connected. Since correlations are therefore considered only between pairs of neurons from two modalities,

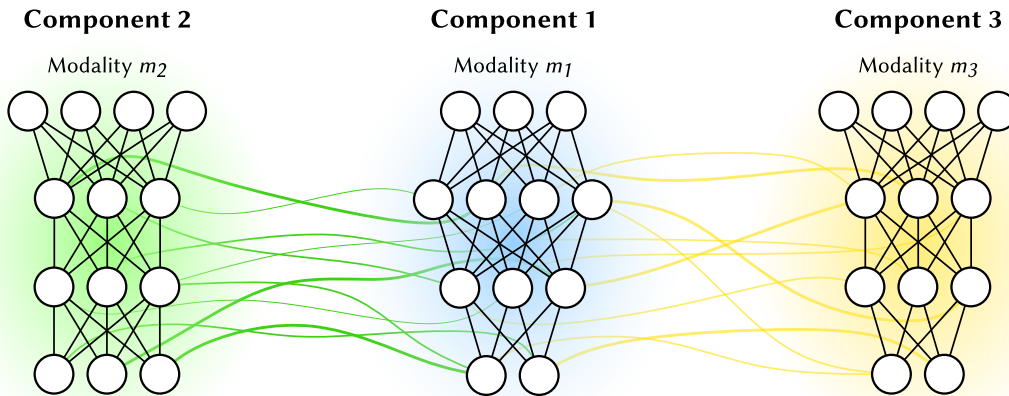


Figure 5.4: Schematic illustration of lateral connectivity in MHNNs. The component network in the center receives input from two other modalities. Synaptic strengths are indicated by the thickness of the connections. Note that input layers are not connected to limit the detection of correlations to deeper layers that are more likely to encode abstract features.

the proposed method is applicable to any component network i with $|\text{Lateral}(m_i)| \geq 1$. As indicated in the figure, lateral connections are defined exclusively for component networks. This design choice is not guided by neuroscientific findings but rather limits complexity compared to unconstrained connectivity between component and hub networks. With every lateral connection representing a high correlation between two neurons, one can identify two possible modes of operation:

- *Anomaly Detection:* The activities of two highly correlated neurons that are laterally connected can be predicted from each other. An anomaly can therefore be interpreted as mismatch between the predicted and the actual activity for a high number of neurons in a component network. The threshold for identifying a mismatch can be scaled by the correlation coefficient of the corresponding pair of neurons. Examples of common sources of anomalies include sensor failure or decreased sensor performance due to adverse environmental conditions.
- *Early Data Fusion:* The additional information provided by laterally connected neurons can be leveraged for *early data fusion*. Re-training a component network after adding lateral connectivity enables the integration of information from related modalities already at an early stage. However, cyclic dependences between different modalities need to be avoided in order to prevent self-amplifying activities due to recurrent feedback.

Early data fusion at component network level is considerably different from late data fusion in the hub networks. Whereas the former is intended to associate sensory streams that process related information (e.g. position estimate of an object based on visual and auditory information), the latter is task-based and can combine streams that encode fundamentally different types of information (e.g. object position and object identity). The

concrete instantiation of the architecture with components, hubs and their connections directly depends on the targeted system and will therefore not be considered further at this point. In the following, we instead propose an algorithm for establishing lateral connectivity between component networks based on correlated neural activity.

Correlation-Based Learning of Lateral Connections with Synaptic Pruning As discussed before, the lateral connections between a pair of component networks encode correlated activity of the corresponding neurons. In the example from Figure 5.4, the correlation strength is indicated by the line width of the synaptic connections. Especially in the case of very deep component networks with many layers and possibly millions of neurons, the identification of correlated neural activity becomes a major computational challenge. Not only are there millions of potential synapses but the activity profiles also need to be tracked over thousands of input patterns. Models that compute a full correlation matrix between all pairs of neurons like the one presented by Axenie et al. [445] are therefore not applicable. Moreover, the assumption of full synaptic connectivity is also unrealistic from a biological perspective. In this light, it is important to revisit the development of the connectome in the brain outlined in Section 5.1. On a conceptual level, synaptic development encompasses two stages that will in the following be referred to as *growth* and *pruning*. In the first stage that occurs during early development, an abundance of synaptic connections are made [455]. The formation of the connectome does not occur until the second stage of ontogenetic development, when unneeded synapses are pruned. These seminal findings not only show that the development of the brain is shaped by interaction with the environment, but they can also be transformed into an efficient algorithm for creating lateral connectivity between component networks.

The complete algorithm for the synthesis of lateral connections between two component networks is outlined in Algorithm 1 and encompasses three steps. At the beginning, a set of undirected lateral connections L is created randomly by sampling pairs of neurons from both networks. The probability distribution that controls the sampling process must not necessarily be uniform. For example, lateral connections between the input layers have been completely omitted in Figure 5.4 to focus connectivity on deeper network layers. In general, those can be expected to represent more abstract features of the input signals and in turn are more likely to be invariant across modalities. The number of lateral connections can be controlled by a parameter ρ as the fraction of the maximum number of lateral connections between the two component networks.

After initialization, the synaptic weights are still undefined. Learning the correlations between connected units requires comparing neural activities in response to corresponding multimodal input patterns that are presented to both component networks at the same time. For this reason, the outer loop in the second step iterates over all samples in the data set D and computes the activations \hat{N}_1, \hat{N}_2 of all neurons in the component networks. When applied to all samples in D , this yields a set of pairs of network activation patterns:

$$D = \{(m_1^1, m_2^1), \dots, (m_1^K, m_2^K)\} \mapsto \{(\hat{N}_1^1, \hat{N}_2^1), \dots, (\hat{N}_1^K, \hat{N}_2^K)\} \quad (5.4)$$

Algorithm 1: Learning Lateral Connections between Component Networks

Input:

- C_1, C_2 : Two component networks $C_1 = (N_1, S_1)$ and $C_2 = (N_2, S_2) \in C$ with neurons N_i and synapses $S_i \subseteq N_i \times N_i$ for modalities m_1 and m_2
 D : A multimodal input data set $D = \{(m_1^1, m_2^1), \dots, (m_1^K, m_2^K)\}$ with corresponding samples from modalities m_1 and m_2

Parameters:

- ρ : Random lateral connectivity ratio
 θ : Synaptic pruning threshold

Output:

- L : A set of lateral *undirected* synaptic connections $L \subseteq \{\{n_1, n_2\} \mid n_1 \in N_1, n_2 \in N_2\}$
 $w_L(\cdot)$: A synaptic weight function $w_L : L \rightarrow [-1, 1]$

Step 1: Formation of Random Lateral Synapses

```

1 L ← ∅
2 while |L| < ρ|N1||N2| do
3   | Sample random n1 ∈ N1 and n2 ∈ N2
4   | L ← L ∪ {n1, n2}
5 end
    
```

Step 2: Learning of Correlated Activity

```

6 k ← 1; μ̄(·) ← 0; σ̄2(·) ← 0; cov̄(·) ← 0
7 foreach (m1k, m2k) ∈ D do
8   | Compute all neuron activations N̂1k ← C1(m1k) and N̂2k ← C2(m2k)
9   | foreach l = {n1, n2} ∈ L do
10    | Update neuron activity means, variances and covariance
11    | μ̄k(n1) ← SampleMean(k, n̂1k, μ̄k-1(n1))
12    | μ̄k(n2) ← SampleMean(k, n̂2k, μ̄k-1(n2))
13    | σ̄k2(n1) ← SampleVar(k, n̂1k, μ̄k(n1), σ̄k-12(n1))
14    | σ̄k2(n2) ← SampleVar(k, n̂2k, μ̄k(n2), σ̄k-12(n2))
15    | cov̄k(n1, n2) ← SampleCov(k, n̂1k, n̂2k, μ̄k(n1), μ̄k(n2), cov̄k-1(n1, n2))
16    | Compute new activity correlation for weight update
17    | wL(n1, n2) ← cov̄k(n1, n2) / √(σ̄k2(n1)σ̄k2(n2))
18   | end
19   | k ← k + 1
20 end
    
```

Step 3: Correlation-Dependent Pruning of Synapses

```

19 foreach l ∈ L do
20   | if |wL(l)| < θ then L ← L \ l
21 end
    
```

The activations of individual neurons $n_1 \in N_1$ and $n_2 \in N_2$ can then be defined accordingly as follows:

$$\hat{n}_i = \{\hat{n}_i^1, \dots, \hat{n}_i^K \mid \hat{n}_i^k \in \hat{N}_i^k\} \text{ for } i \in \{1, 2\} \quad (5.5)$$

\hat{n}_1 and \hat{n}_2 provide the basis for computing the sample means $\bar{\mu}(\cdot)$, the sample variances $\bar{\sigma}^2(\cdot)$ and the sample covariances $\bar{\text{cov}}(\cdot)$ for connected pairs of neurons in the two component networks:

$$\bar{\mu}(n_i) = \frac{1}{K} \sum_{k=1}^K \hat{n}_i^k \quad \text{for } i \in \{1, 2\} \quad (5.6)$$

$$\bar{\sigma}^2(n_i) = \frac{1}{K-1} \sum_{k=1}^K (\hat{n}_i^k - \bar{\mu}(n_i))^2 \quad \text{for } i \in \{1, 2\} \quad (5.7)$$

$$\bar{\text{cov}}(n_1, n_2) = \frac{1}{K-1} \sum_{k=1}^K (\hat{n}_1^k - \bar{\mu}(n_1))(\hat{n}_2^k - \bar{\mu}(n_2)) \quad (5.8)$$

Note that Equations 5.7 and 5.8 include *Bessel's correction* to compensate for bias in the sampling variance and covariance [458]. The weights $w_L(\cdot)$ between pairs of laterally connected units $\{n_1, n_2\} \in L$ can now be defined as the statistical correlation of their activations on D [459]:

$$w_L(\{n_1, n_2\}) = \frac{\bar{\text{cov}}(n_1, n_2)}{\sqrt{\bar{\sigma}^2(n_1)\bar{\sigma}^2(n_2)}} \quad (5.9)$$

In the following, n will refer to an arbitrary neuron in one of the two component networks. Computing lateral weights with Equation 5.9 from Equations 5.7 and 5.8 involves storing all activation patterns observed for all lateral synapses L on data set D . Depending on the size of L and D , this might require very large amounts of storage and thus can be very inefficient. Algorithm 1 therefore implements an incremental method that was originally proposed by Welford [460] to compute sample means, sample variances and sample covariances in a single pass over D . The online update for the sample mean $\bar{\mu}(\cdot)$ can be directly derived from Equation 5.6:

For $k \geq 1$ and $\bar{\mu}_0(n) := 0$:

$$\text{SampleMean}(k, \hat{n}^k, \bar{\mu}_{k-1}(n)) = \frac{1}{k} (\hat{n}^k + (k-1)\bar{\mu}_{k-1}(n)) \quad (5.10)$$

The sample mean is required to compute both sample variances, covariances and correlations in the subsequent steps. At the core of Welford's incremental online algorithm for the sample variance is the following identity [460]:

$$\begin{aligned} S_k(n) &= \sum_{l=1}^k (\hat{n}^l - \bar{\mu}_k(n))^2 \\ &= S_{k-1}(n) + \frac{k}{k-1} (\hat{n}^k - \bar{\mu}_k(n))^2 \end{aligned} \quad (5.11)$$

$$\begin{aligned} \bar{\sigma}_k^2(n) &= \frac{1}{k-1} S_k(n) \\ &= \frac{k-2}{k-1} \bar{\sigma}_{k-1}^2(n) + \frac{k}{(k-1)^2} (\hat{n}^k - \bar{\mu}_k(n))^2 \end{aligned} \quad (5.12)$$

Equation 5.12 already includes Bessel's correction. The online update function for the sample variances in Algorithm 1 can therefore be computed as follows:

For $k \geq 1$, $\bar{\mu}_0(n) := 0$ and $\bar{\sigma}_0^2(n) := 0$:

$$\text{SampleVar}(k, \hat{n}^k, \bar{\mu}_k(n), \bar{\sigma}_{k-1}^2(n)) = \frac{k-2}{k-1} \bar{\sigma}_{k-1}^2(n) + \frac{k}{(k-1)^2} (\hat{n}^k - \bar{\mu}_k(n))^2 \quad (5.13)$$

The update equations for the sample covariance can be derived analogously:

$$\begin{aligned} C_k(n_1, n_2) &= \sum_{l=1}^k (\hat{n}_1^l - \bar{\mu}_k(n_1))(\hat{n}_2^l - \bar{\mu}_k(n_2)) \\ &= C_{k-1}(n_1, n_2) + \frac{k}{k-1} (\hat{n}_1^k - \bar{\mu}_k(n_1))(\hat{n}_2^k - \bar{\mu}_k(n_2)) \end{aligned} \quad (5.14)$$

$$\begin{aligned} \overline{\text{cov}}_k(n_1, n_2) &= \frac{1}{k-1} C_k(n_1, n_2) \\ &= \frac{k-2}{k-1} \overline{\text{cov}}_{k-1}(n_1, n_2) + \frac{k}{(k-1)^2} (\hat{n}_1^k - \bar{\mu}_k(n_1))(\hat{n}_2^k - \bar{\mu}_k(n_2)) \end{aligned} \quad (5.15)$$

For $k \geq 1$, $\bar{\mu}_0(n_1) := 0$, $\bar{\mu}_0(n_2) := 0$, $\bar{\sigma}_0^2(n_1) := 0$ and $\bar{\sigma}_0^2(n_2) := 0$:

$$\begin{aligned} \text{SampleCov}(k, \hat{n}_1^k, \hat{n}_2^k, \bar{\mu}_k(n_1), \bar{\mu}_k(n_2), \overline{\text{cov}}_{k-1}(n_1, n_2)) &= \\ &= \frac{k-2}{k-1} \overline{\text{cov}}_{k-1}(n_1, n_2) + \frac{k}{(k-1)^2} (\hat{n}_1^k - \bar{\mu}_k(n_1))(\hat{n}_2^k - \bar{\mu}_k(n_2)) \end{aligned} \quad (5.16)$$

All estimates of sample means, sample variances and the sample covariance are updated online for all laterally connected neurons as new samples are processed. In particular,

Algorithm 1 computes mean and variance updates each time a neuron is visited, which does not change the result but becomes increasingly inefficient the more neurons are participating in multiple lateral connections. In practice, however, this can be easily avoided by maintaining a list neurons that have already been processed in the current iteration. As soon as all values for a connection l have been updated, the resulting weight $w_L(l)$ can be computed according to Equation 5.9. In the last step of the algorithm, all synapses with an absolute weight $|w_L(l)|$ below a user-defined threshold θ are finally pruned from the network. As a result, only neurons with high directly or indirectly correlated activity remain laterally connected. The final connection weights store the correlation coefficients of neuron activities with respect to the multimodal input data set D . This means that the lateral connectome is shaped by the system's experience and depends on the structure of both its body and the environment.

Links to Hebbian Synaptic Plasticity Even though the computation of lateral weights as described in Algorithm 1 does not include any direct reference to neuroscientific models of synaptic plasticity, it can actually be interpreted as a normalized and covariance-based extension of Hebb's well-known rule "*what fires together wires together*" [461]:

$$\Delta w_L^k(\{n_1, n_2\}) = \eta \hat{n}_1^k \hat{n}_2^k \quad (5.17)$$

In the equation above, the weight update in step k is defined as the product of neuron activities scaled by a learning rate η . Weight changes are thereby effectively driven by correlated activity of the presynaptic and the postsynaptic neuron. However, the simplicity of the rule imposes narrow constraints on the dynamics of the weight update. In particular, weights can only increase. For this reason, many variations of the basic Hebb rule have been proposed. Sejnowski et al. [462] introduced a covariance-based version as a model of synaptic plasticity in the hippocampus:

$$\Delta w_L^k(\{n_1, n_2\}) = \eta (\hat{n}_1^k - \bar{n}_1)(\hat{n}_2^k - \bar{n}_2) \quad (5.18)$$

\bar{n}_1 and \bar{n}_2 denote the average firing rates of neurons n_1 and n_2 . One can easily see that Equation 5.18 is similar to the update term in Equation 5.15 when the coefficients are neglected. For large values of k , the left term's coefficient converges to 1 while that of the right one converges to 0. Equation 5.18 can therefore be thought of as an approximation of Equation 5.15 when assuming a decaying learning rate η . The main difference between the two update rules is that the two terms in Equation 5.15 are weighted by the number of samples and therefore converge to a limit, while the average neural activity in 5.18 is computed only for a certain sliding time window. Furthermore, the final learning rule in Algorithm 1 from Equation 5.9 also applies normalization based on neuron activation variances. This is required because the activation statistics of individual neurons in the component networks may differ considerably from each other. In case they can assumed to be similar for all neurons, the denominator in Equation 5.9 can be approximated by a constant factor that can be included in the learning rate η . Interestingly, normalization is hypothesized to play an important role in the brain [463].

5.3 Self-Supervised Learning of Deep Multisensory Neural Maps

The neural network architecture developed in the previous section requires that component networks map raw input to a compact abstract representation, but is completely agnostic of the actual implementation. Possible choices include autoencoders or the feature layers of pre-trained DNNs. The most appropriate architecture can be selected independently for each component network. As a result, the generated latent space representations are specific to each modality and need to be fused individually in each hub network. While this general approach is fully in line with the design principles of the architecture, it does not exploit the complementarity of multiple correlated sensory streams that are available in embodied systems. In this section, we introduce a training procedure along with a novel loss function for learning a joint sensory map across two different modalities. After the specification of the underlying experimental setup in the NRP in the next subsection, we first present an algorithmic framework for the unconstrained self-supervised learning of common latent space representations across different modalities. In the main part of this section, this framework is extended with a novel topographic loss function that enables the learning of cortex-like topographic maps. The models and algorithms developed in this section have been implemented based on an extended version the framework presented in [464].

Experimental Setup and Data Set

The development of machine learning models that process data from multiple input modalities is very challenging due to the requirement for corresponding multimodal data sets. Each sample contained therein must be of the form $((m_1^k, \dots, m_l^k), l^k)$, where m_i^k is the current input from modality m_i and l^k is an optional label. While multimodal data sets are less common in traditional machine learning domains such as image processing, they are easily available in embodied systems, where sensors automatically produce correlated streams of data for multiple modalities. Neurorobotics is therefore an essential tool for the study of multisensory phenomena. This subsection introduces an experimental setup for the NRP that has been designed to generate multimodal data sets with both visual and proprioceptive information.

NRP Experiment The experiment is set up in the virtual lab environment of the NRP and is centered around the model of the KUKA LBR iiwa robot introduced in Section 4.1. Figure 5.5 depicts the complete system in the NRP Web Front End. The robot is placed on a platform in the virtual lab environment of the NRP to enable free and unconstrained movement in all directions. Differently from Section 4.1, there is no tool attached because the data collection process does not involve any object manipulation. Instead, the tip of the outermost joint is highlighted by a green sphere. This is because the main objective of the experiment is to enable the recording of correlated multimodal data streams related to the



Figure 5.5: NRP experiment for the collection of multimodal training data as required by deep multisensory neural maps. The setup is based on the KUKA LBR iiwa robot model introduced in Section 4.1. Instead of a gripper, a green sphere is attached to the robot’s last link in order to provide a salient visual stimulus that indicates the workspace position. The blue dots are a visualization of the sampling space and correspond to samples from the multimodal data set D_{map} . The insets on the right display additional camera views of the scene. During data collection, joint angles (*proprioception*) and camera data (*vision*) are recorded simultaneously.

robot’s current joint configuration and workspace position. Towards this end, the setup provides not only sensor readings of the current joint angles but also includes a camera with a complete view of the scene from the front. An inverse kinematics model enables the direct movement of the robot to target workspace positions, which is required for the data collection procedure discussed in the next paragraph. The experiment supports parallel distributed execution with the framework from Section 4.1. This is an important prerequisite for the rapid generation of new samples without extended processing time.

Generation of the Data Set The design of the experimental setup directly determines the structure of the data captured from it. There are two complementary sensory modalities, which both implicitly encode the workspace position of the robot’s outermost link: Sensors in the individual joints measure angles and the camera in front of the robot captures images of the scene including the green sphere at the last joint’s tip. With both the joint angle readings and the camera images being recorded simultaneously from the system, they are automatically correlated. New samples can therefore be easily retrieved by changing the robot configuration and capturing the resulting sensor output. However, unlike the joint angles, the monocular camera images do not provide a full spatial

description of the scene. While this can in principle be easily addressed by adding a second camera or a depth sensor to the experiment, capturing the full three-dimensional workspace of the robot is not essential for the development of the models and algorithms presented in this section. For this reason, we will only consider robot configurations with the tip located on the plane $y = 0$ that is parallel to the image plane of the camera. We denote the robot's configuration space by \mathbb{C}_{KUKA} . Based on the specification provided by the manufacturer, it can be described as follows [465]:

$$\mathbb{C}_{KUKA} = [\pm 170^\circ] \times [\pm 120^\circ] \times [\pm 170^\circ] \times [\pm 120^\circ] \times [\pm 170^\circ] \times [\pm 120^\circ] \times [\pm 175^\circ] \quad (5.19)$$

The motion ranges of the individual joints are listed starting with the base joint at the bottom. Moving the robot tip randomly to different positions $p_k \in \mathbb{R}^2$ on the plane and capturing the corresponding joint angles $q_k \in \mathbb{C}_{KUKA}$ and camera images $\mathbf{I}_k \in \mathbb{R}^{512 \times 512 \times 3}$ yields a multimodal data set D_{map} :

$$D_{map} = \{((q_0, \mathbf{I}_0), p_0), \dots, ((q_K, \mathbf{I}_K), p_K)\} \quad (5.20)$$

Algorithm 2 provides a formal description of the generation of D_{map} . To create a new sample, the algorithm generates a random workspace position p_k on the half-annulus covering the portion of the plane $y = 0$ that is reachable by the robot. The corresponding robot configuration is then determined using the *MoveIt* motion planning framework. Targets p_k for which the motion planner fails are discarded. Furthermore, self-collisions of the robot with the green sphere at its tip are neglected because it is only visual marker and not a structural part of the robot. Since only target positions are specified, the system can in principle produce arbitrary orientations. As it turns out, this approach, although biased by the motion planner, leads to good coverage of the range of motion of each individual joint. For experiments that require both the robot's tool tip position and orientation, the setup can be easily extended to sample complete target poses. The final data set D_{map} is shown in Figure 5.6 and contains a total of 102 000 samples, of which three selected ones are visualized in Figure 5.7.

Component Networks

Following the architecture description from Section 5.2, the two modalities contained in D_{map} are processed by two individual component networks C_{vis} and C_{pro} for vision and proprioception, respectively. To account for the fundamentally different nature of visual and proprioceptive data, each network has a different architecture. In the next paragraphs, both component networks are specified and evaluated on a supervised learning task.

Algorithm 2: Generation of the Multimodal Data Set D_{map} **Input:** E_{NRP} : NRP experiment as shown in Figure 5.5**Output:** D_{map} : A multimodal data set $D_{map} = \{(q_0, \mathbf{I}_0), p_0), \dots, ((q_K, \mathbf{I}_K), p_K)\}$ **Parameters:** K : The number of samples to be generated r_{min} : Minimum radius of the sampling space r_{max} : Maximum radius of the sampling space*Generate samples with the robot tip on the half-annulus (r_{min}, r_{max}) in the plane $y = 0$*

```

1  $k \leftarrow 0$ ;  $D_{map} \leftarrow \emptyset$ 
2 while  $k < K$  do
3   Generate uniform random numbers  $r_k, \varphi_k \in [0, 1]$ 
4    $r_k \leftarrow \sqrt{r_k \cdot (r_{max}^2 - r_{min}^2) + r_{min}^2}$ 
5    $\varphi_k \leftarrow \varphi_k \pi$ 
6    $(x_k, z_k) \leftarrow (r_k \cos \varphi_k, r_k \sin \varphi_k + E_{NRP}.robot\_offset)$ 
7   if  $E_{NRP}.move\_robot(x_k, 0, z_k)$  then
8      $q_k \leftarrow E_{NRP}.joint\_angles()$ 
9      $\mathbf{I}_k \leftarrow E_{NRP}.camera\_image()$ 
10     $p_k \leftarrow (x_k, z_k)$ 
11     $D_{map} \leftarrow D_{map} \cup ((q_k, \mathbf{I}_k), p_k)$ 
12     $k \leftarrow k + 1$ 
13     $E_{NRP}.move\_robot("default\_configuration")$ 
14  end
15 end

```

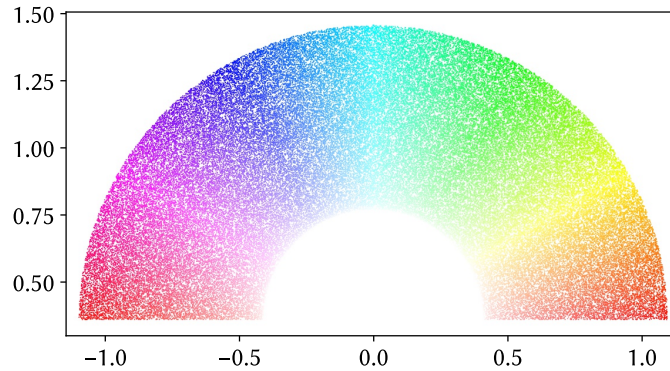


Figure 5.6: Workspace positions of all 102 000 samples in the multimodal data set D_{map} . The coloring serves as a reference to visualize latent space representations in subsequent figures.

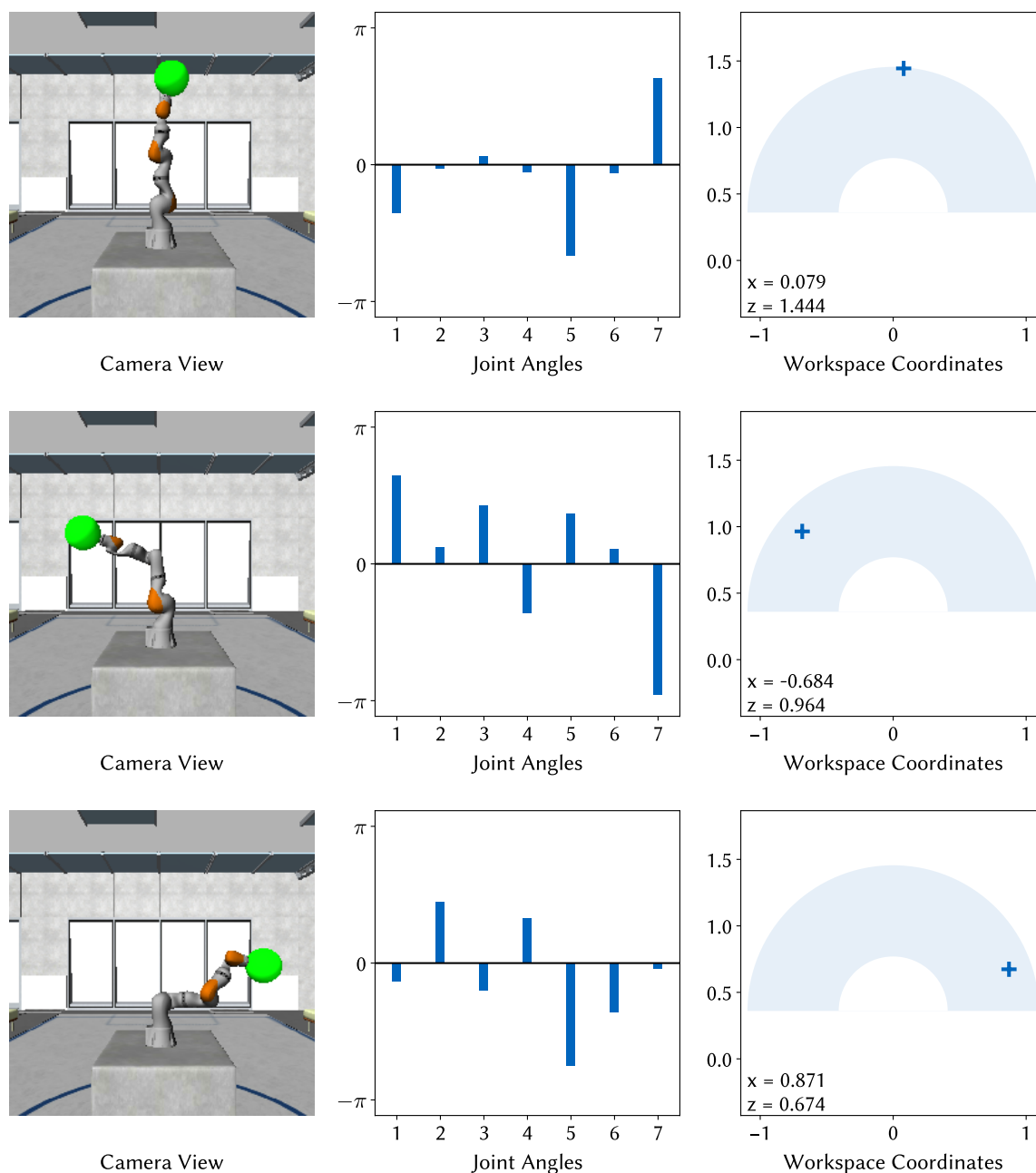


Figure 5.7: Visualization of selected samples from the multimodal data set D_{map} . Each row corresponds to a single sample and each column contains a different representation of the robot's workspace position, which is defined as the center of the green sphere attached to the outermost link. The images in the left column depict a front view of the robot's current configuration, while the corresponding joint angles and workspace coordinates of the center of the sphere at the tip are shown in the middle and right column, respectively. Since the sampling of robot configurations is constrained to a fixed two-dimensional plane in the workspace with $y = 0$, only the x - and z -coordinates of the workspace position are shown. The shaded area in the right column indicates the sampling space.

Network Architecture Figure 5.8 provides an overview of the architectures of the image network C_{vis} and the proprioception network C_{pro} . The latter directly receives raw joint angles as input and is therefore set up as a simple fully connected neural network with four hidden layers and $\text{ReLU}(\cdot)$ activation functions. Each of the inner layers is followed by a batch normalization layer. After an initial expansion of the representation space, the network input is finally compressed to two dimensions in the output layer and thereby mapped to the latent space as shown in Figure 5.3. The output dimension of the vision network C_{vis} is identical to enable the learning of a common latent space representation. Like in the proprioception network, the value range of the output layer is constrained by a $\text{tanh}(\cdot)$ activation function as introduced in Figure 2.6. But since the input is not a low-dimensional robot configuration but a high-dimensional camera image, the network is built from convolution layers with continuously decreasing filter sizes. Unlike in common DNN architectures for image classification, there are no pooling layers. This is because these layers are applied to construct position-invariant feature space representations. Filtering out position-related information, however, would make it impossible to reliably map the location of the sphere to a workspace position. For the intended application, the omission of pooling layers is therefore an essential design choice. It should be noted that the design of C_{vis} and C_{pro} not necessarily maximizes performance or efficiency. In fact, there may be many other architectures that perform better on D_{map} . The main goal in this work is rather to minimize model complexity to reduce side effects and to put the focus on the proposed learning method. Nevertheless, other functionally equivalent types of neural networks are equally applicable to the learning methods presented in this section.

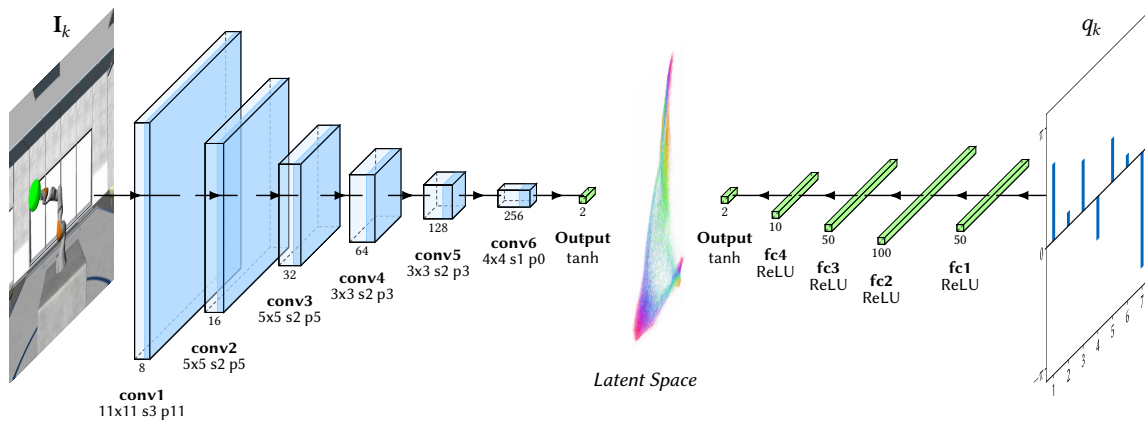


Figure 5.8: Architectures of the component networks C_{vis} and C_{pro} . **conv** and **fc** denote convolution layers and fully connected layers, respectively. Both output layers are fully connected. The convolution layers of C_{vis} (left) have $\text{ReLU}(\cdot)$ activation functions. Stride width s and padding p are annotated below each layer next to the kernel size. C_{pro} (right) is comprised of fully connected layers with $\text{ReLU}(\cdot)$ activation functions. Not shown are the batch normalization layers that follow after every inner layer of both networks. Each of the two output layers applies a $\text{tanh}(\cdot)$ activation function. Both networks project to a two-dimensional space that corresponds to the latent space from Figure 5.3. The network parameters are based on [464].

Supervised Training Both C_{vis} and C_{pro} are implemented with the PyTorch deep learning library and the PyTorch Lightning interface [137, 466]. To evaluate the networks' baseline performance with respect to the extraction of information related to the robot's workspace position, they were trained individually on the ground truth positions p_k stored in D_{map} . In particular, D_{map} can be split into two unimodal data sets for each modality:

$$D_{map}^{vis} = \{(\mathbf{I}_0, p_0), \dots, (\mathbf{I}_K, p_K)\} \quad (5.21)$$

$$D_{map}^{pro} = \{(q_0, p_0), \dots, (q_K, p_K)\} \quad (5.22)$$

As a result, the training of the networks can be formulated as a standard supervised learning task. The prediction error for a set of N samples is computed as the mean squared error (MSE) between the ground truth positions p_1, \dots, p_N and the network prediction $\hat{p}_1, \dots, \hat{p}_N$:

$$\text{MSE}(p_1, \dots, p_N; \hat{p}_1, \dots, \hat{p}_N) = \frac{1}{N} \sum_{n=1}^N \|p_n - \hat{p}_n\|_2^2 \quad (5.23)$$

Minimizing this error function also minimizes the Euclidean distances between the latent space projections of both networks. The $\tanh(\cdot)$ activation functions of the networks' output units were omitted in this particular experiment because the dimensions of the workspace exceed their value range of $(-1, 1)$. The training was performed in PyTorch with the Adam optimizer [467]. All parameters used for the training of the models presented in this section are documented in Appendix A.1.

Results Figure 5.9 provides a visual summary of the results. While the plots and errors are shown for the full data set, only 40 % of the samples were used for training and validation. As one can clearly see, both networks have learned the general topography of the workspace. The overall higher accuracy of the proprioception network can partly be explained by the limited resolution of the camera and the fact that samples in D_{map} with neighboring workspace positions might have been reached with different orientations and are thus represented by considerably different camera images. By contrast, the joint angles processed by the proprioception network provide a complete and unambiguous description of the robot's workspace position. The error shown in Figure 5.9 is computed based on the Euclidean distances between sample workspace positions, which are easier to interpret for the considered problem setup than the MSE. Mean distances above the spatial mapping plots are computed from the pairwise workspace distances between 25 000 samples of D_{map} and capture the overall spatial extent of the latent space mapping. The mean errors for the mapping error plots are computed as the mean of the distances between predicted positions and ground truth. For both C_{vis} and C_{pro} , the mean distances and their standard deviations correspond well to the those of the ground truth positions. In summary, these findings show that both component networks are in principle capable of mapping their input space to a common representation of the robot's workspace, which is an essential prerequisite for the learning method introduced in the next two subsections.

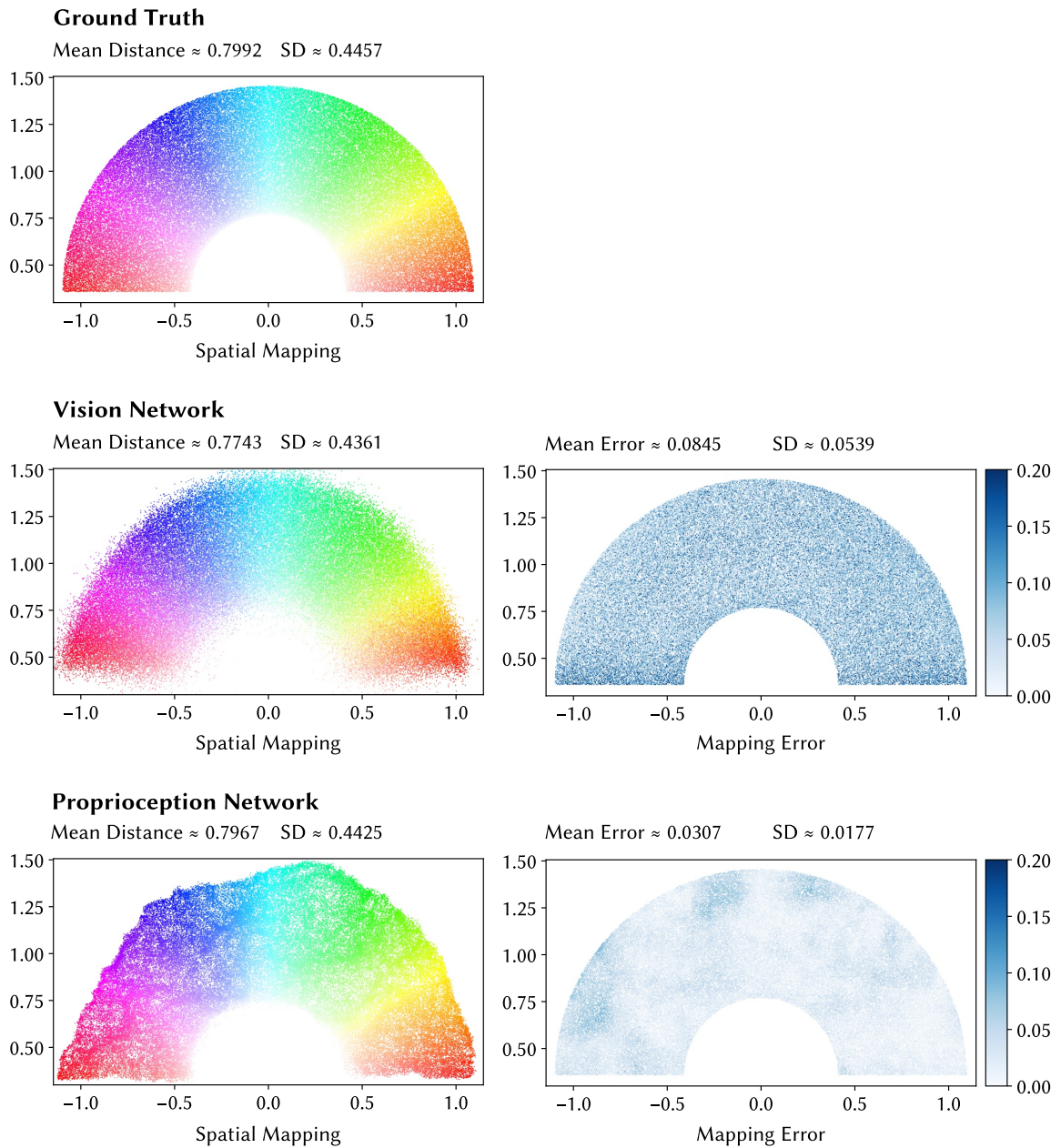


Figure 5.9: Input space mappings learned by the vision network C_{vis} and the proprioception network C_{pro} after supervised training on D_{map}^{vis} and D_{map}^{pro} . *Left:* Distribution of the workspace sample positions inferred by the networks. Sample colors correspond to those in Figure 5.6. Means of the pairwise workspace sample distances computed on a subset of the data set and their standard deviations are noted at the top of each row. *Right:* Visualization of the Euclidean norm of the network prediction error. Means of the Euclidean distance error on the full data set and their standard deviations are noted at the top of each row. The upper limits of the the colorbars are saturation values and do not correspond to the maximum error.

Multimodal Self-Supervised Learning

Training the component networks C_{vis} and C_{pro} with external supervision on workspace positions is unrealistic from the point of view of a self-contained embodied system that has no access to ground truth information. In the following, we therefore develop a *self-supervised* training procedure that requires only data from the two input modalities of D_{map} . The basic idea behind the concept of self-supervision is best explained in the context of the four main machine learning paradigms illustrated in Figure 5.10. While unsupervised learning solely operates on unlabeled samples (\mathbf{I}_k and \mathbf{q}_k in the case of D_{map}) with the goal of identifying latent structure in the data, supervised learning aims to map input samples to pre-defined outputs specified by labels (p_k in the case of D_{map}). Generating labeled data sets is one of the main challenges in supervised learning and a key prerequisite for entering new application domains. Self-supervised learning as defined in this work applies algorithms from supervised learning to unlabeled data by applying a loss function that does not depend on any labels. A prominent example for this class of systems are autoencoder networks that aim to reconstruct their input at the output layer. As the error function used for this type of network measures the reconstruction quality by comparing the system's reconstructed output with the original input, no labels are required. With only one data source being involved that is mapped onto itself, autoencoders are commonly referred to as unsupervised models even though the underlying learning method is self-supervised. In the next paragraph, the learning of a common latent representation from D_{map} is formulated as a self-supervised learning problem with a corresponding loss function. The latter forms the basis of the actual learning algorithm that will be described afterwards.

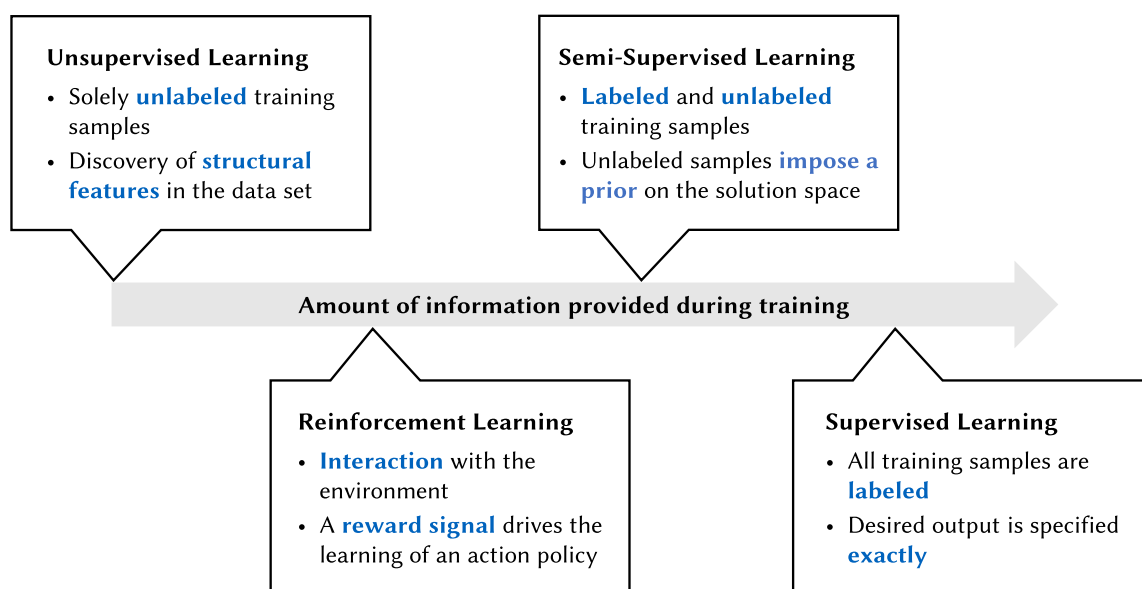


Figure 5.10: Overview of the four main machine learning paradigms. The description of semi-supervised learning is based on the definition by Zhou and Belkin [468].

Problem Setup The overall problem setup is illustrated in Figure 5.8. When considering both component networks as a single model, the resulting architecture is akin to that of a typical autoencoder where data pass through a low-dimensional latent space bottleneck in the middle of the network. The difference lies in the data flow: While autoencoders process input in feedforward direction only, the data flows through the vision network C_{vis} and the proprioception network C_{pro} point in opposite directions to meet at the latent space located at the center. This is essential to make both component networks encoders that map their respective input modality to the latent space. In an autoencoder, the portion of the network following after the latent space becomes a decoder that maps back to the input space, which makes it impossible to use it as a component network.

Basic Self-Supervised Loss With both C_{vis} and C_{pro} processing their inputs independently of each other, their latent output spaces differ by default. Synchronizing them by minimizing the difference between their latent space projections of D_{map} yields a self-supervised learning problem with the following loss functions:

$$\mathcal{L}_{vis}^{self}(C_{vis}\langle\mathbf{I}\rangle, C_{pro}\langle q\rangle) = \text{MSE}(C_{vis}\langle\mathbf{I}\rangle, C_{pro}\langle q\rangle) + \frac{\zeta}{\bar{\sigma}^2(C_{vis}\langle\mathbf{I}\rangle) + \varepsilon} \quad (5.24)$$

$$\mathcal{L}_{pro}^{self}(C_{vis}\langle\mathbf{I}\rangle, C_{pro}\langle q\rangle) = \text{MSE}(C_{vis}\langle\mathbf{I}\rangle, C_{pro}\langle q\rangle) + \frac{\zeta}{\bar{\sigma}^2(C_{pro}\langle q\rangle) + \varepsilon} \quad (5.25)$$

Both losses are defined for batches $\langle\mathbf{I}\rangle, \langle q\rangle$ of samples from D_{map} and compute the MSE between the latent space projections of C_{vis} and C_{pro} . However, the MSE alone would easily lead to trivial solutions where both networks produce identical constant output for all inputs. This is addressed by the second part of the loss functions, which adds the inverse variance of the projection of the current batch scaled by a hyperparameter ζ to promote the spread of data points over the whole latent space. ε is a small positive constant which ensures that the loss does not diverge for small variances. In the literature, the problem of synchronizing latent space representations is also referred to as *manifold alignment* [469].

Learning Algorithm Equations 5.24 and 5.25 require the latent space output of both C_{vis} and C_{pro} to compute the current loss for the weight updates, which implies that the two networks need to be trained at the same time. This is challenging from a technical point of view since the output of the two loss functions for identical inputs changes after each weight update. Both networks are therefore trained on a moving target with direct effects on the convergence and stability of the learning process. To address this issue, losses for one network are computed based on a cached version of the other that is gradually updated over time. The cached networks \tilde{C}_{vis} and \tilde{C}_{pro} are also referred to as *target networks*, a concept originally introduced in the context of deep reinforcement learning [470, 471]. The complete training procedure is described in Algorithm 3 and starts with two randomly initialized component networks C_{vis} and C_{pro} . In every epoch $e \in E$,

Algorithm 3: Multimodal Self-Supervised Learning

Input:

- D_{map} : A multimodal data set $D_{map} = \{(q_0, \mathbf{I}_0), p_0), \dots, (q_K, \mathbf{I}_K), p_K)\}$
- C_{vis} : Component network for vision with randomly initialized weights
- C_{pro} : Component network for proprioception with randomly initialized weights

Parameters:

- \mathcal{L}_{vis} : Loss function for the vision network
- \mathcal{L}_{pro} : Loss function for the proprioception network
- E : Number of training epochs
- b : Batch size
- τ : Target network update rate
- Γ_{vis} : Optimization parameters for C_{vis}
- Γ_{pro} : Optimization parameters for C_{pro}

Initialize the target networks

- 1 $\tilde{C}_{vis} \leftarrow C_{vis}; \tilde{C}_{pro} \leftarrow C_{pro}$
- 2 **foreach** $e \in [1 \dots E]$ **do**
 - Iterate over the full data set in batches of size b*
 - 3 **foreach** $\langle \mathbf{I}, \langle q \rangle \in \langle D_{map}, b \rangle$ **do**
 - Compute loss gradients with respect to synaptic weights*
 - 4 $g_{vis} \leftarrow \nabla_{C_{vis}} \mathcal{L}_{vis}(C_{vis}(\mathbf{I}), \tilde{C}_{pro}(\langle q \rangle))$
 - 5 $g_{pro} \leftarrow \nabla_{C_{pro}} \mathcal{L}_{pro}(\tilde{C}_{vis}(\mathbf{I}), C_{pro}(\langle q \rangle))$
 - Update network weights*
 - 6 $C_{vis} \leftarrow \text{Optimizer}(C_{vis}, \Gamma_{vis}, g_{vis})$
 - 7 $C_{pro} \leftarrow \text{Optimizer}(C_{pro}, \Gamma_{pro}, g_{pro})$
 - Perform soft target network updates*
 - 8 $\tilde{C}_{vis} \leftarrow (1 - \tau)\tilde{C}_{vis} + \tau C_{vis}$
 - 9 $\tilde{C}_{pro} \leftarrow (1 - \tau)\tilde{C}_{pro} + \tau C_{pro}$
- 10 **end**
- 11 **end**

the data set D_{map} is split into batches of size b that form the basis for the computation of losses and gradients. The actual weight updates are applied in Lines 6 and 7 by a gradient-based optimizer. For example, $\text{Optimizer}(\cdot)$ expands as follows for basic stochastic gradient descent and $\Gamma = \{\eta\}$:

$$\text{Optimizer}(C, \Gamma, g) = C - \eta g \quad (5.26)$$

In this work, we instead use the PyTorch implementation of the Adam optimizer that is based on stochastic gradient descent but in addition “computes individual adaptive learning rates for different parameters from estimates of the first and second moments of the gradients” [467]. It is important to note, however, that there is no requirement for using any specific optimization algorithm.

Results The results of two runs of Algorithm 3 over ten epochs each are shown in Figure 5.11. While both of them visualize the full data set D_{map} , training and validation were carried out on 40 % of the samples. Mean distances and latent space errors are computed analogously as described for Figure 5.9. In both runs, the two component networks converge to common latent space representations. The errors are comparable to those from supervised training but are not sufficient to fully describe the quality of the results, which can be seen from the significant differences between the two representations learned in the two runs. We will therefore consider three factors for characterizing the quality of the learned latent space representations:

- *Scale*: The overall spatial extent of the latent space projection of D_{map} as described by the mean pairwise distances between sample positions.
- *Topography*: The degree of correspondence between the layouts of the ground truth workspace and the latent space.
- *Error*: The mean Euclidean distance between corresponding samples in the latent space projections of C_{vis} and C_{pro} .

The scale of the latent space puts the magnitude of the error into context. For example, in degenerate cases where both component networks only produce constant output the error vanishes. But at the same time, the scale as described by the mean pairwise sample distance also converges to zero, resulting in an diverging relative error:

$$\text{Relative Latent Space Error} = \frac{\text{Mean Latent Space Error}}{\text{Mean Distance}} \quad (5.27)$$

In general, small scales do not necessarily correspond to disadvantageous latent space representations but may still be undesirable. This is especially true if the scale decreases during training and numerical precision is lost over time with negative implications on the quality of the latent space representation. Both error and scale are only concerned with the spatial extent and alignment of the representations learned by C_{vis} and C_{pro} . An even more important property is how well the learned latent space representation preserves topographic features of D_{map} in the robot’s workspace. In qualitative terms, a good topographic mapping reproduces the overall layout of the workspace as observed through the camera such that neighborhood relationships are preserved [472]. More formally, the concept of topographic organization can be defined as follows [473]:

“The arrangement of components in a structure, particularly the orderly spatial relationship between the distribution of neural receptors in an area of the body and a related distribution of neurons representing the same functions in cortical sensory regions of the brain. [...]”

In the representation learned during Example Run 1, it is possible to discern the different areas of the workspace. However, the separation is not sharp and there is considerable overlap. Some areas like the purple one seem to be torn apart and are widely distributed

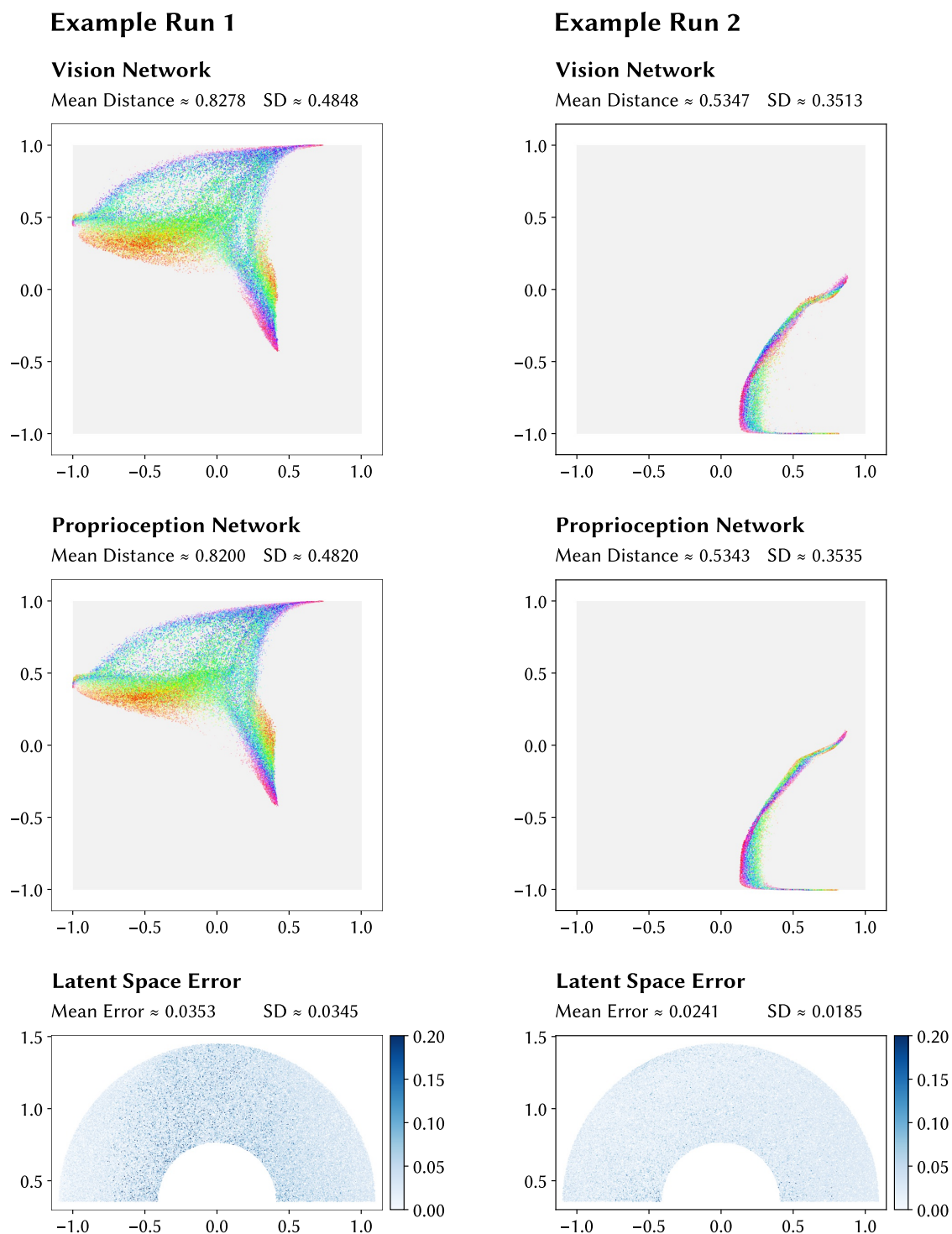


Figure 5.11: Visualization of the results of two selected runs of Algorithm 3 with losses \mathcal{L}_{vis}^{self} and \mathcal{L}_{pro}^{self} from Equations 5.24 and 5.25. Shaded areas indicate the output range of the $\tanh(\cdot)$ activation function. The upper limits of the the colorbars are saturation values and do not correspond to the maximum error.

5 A Brain-Derived Modular-Hierarchical Neural Network Architecture

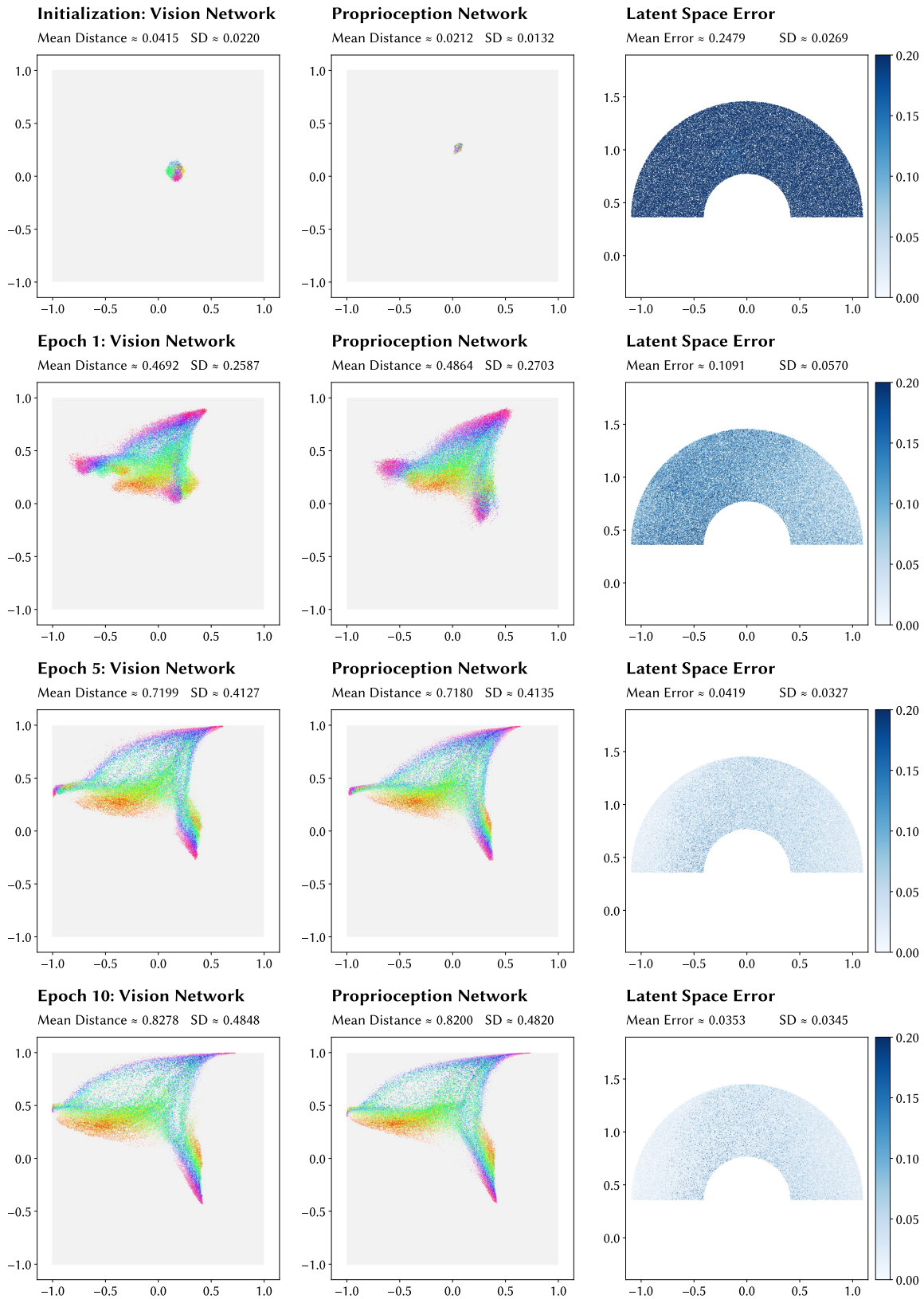


Figure 5.12: Visualization of the learning progress in Example Run 1 from Figure 5.11. Shaded areas indicate the output range of the $\tanh(\cdot)$ activation function. The upper limits of the the colorbars are saturation values and do not correspond to the maximum error.

over the output space. The result from Example Run 2 has a completely different shape with decreased spatial separation at the bottom. This is also manifested in a smaller mean distance with lower standard deviation.

The development of the latent space representation during training in the first run from network initialization to the final epoch is outlined in Figure 5.12. Most evident is the expansion of the projection that can be observed already by the end of the first epoch. Unlike that of the proprioception network, the projection of the vision network already contains discernible colored clusters directly after initialization, which is partly due to the fact that the green sphere in the camera images directly encodes the robot’s workspace position. With the background of the scene remaining unchanged apart from the robot arm, the output of C_{vis} is mainly influenced by the location of the sphere. By contrast, the proprioceptive input processed by C_{pro} resides in the robot’s configuration space, which has a completely different topology. This explains why all colors appear to be mapped completely randomly after initialization and demonstrates the huge impact of the network architecture on the representational power of a model for a specific domain. In conclusion, Algorithm 3 applied with the losses defined in Equations 5.24 and 5.25 can learn common latent space representations for C_{vis} and C_{pro} . However, the topologies of these representations emerge randomly and do not necessarily capture the characteristics of the robot’s workspace well.

Learning Aligned Multisensory Maps with Topographic Loss

The formation of the latent space with the loss functions introduced in the last subsection is mainly driven by the initial network weights and the randomness of stochastic gradient descent-based optimization. It is therefore impossible to control the emerging topography. This is best illustrated in Figure 5.11, where two runs of the algorithm yield two qualitatively completely different results. Even though both exhibit basic structure, none of them reflects the topography of the robot’s workspace well. Providing additional information as a prior that informs the learning process is therefore essential. In the following, we develop an extension of the basic loss functions from Equations 5.24 and 5.25 and introduce a training protocol that balances between individual map formation and cross-modal synchronization. The proposed methodology is motivated by findings about topographic map representations that have been discovered in the brain and which are hypothesized to play a key role in cognitive processing.

Topographic Maps in the Brain Topographic maps have been found to be a key principle for the representation of sensory input in the brain, especially the mammalian neocortex [474]. A prototypic example is the retinotopic mapping from the sensory receptor cells of the eye to the primary visual cortex. Importantly, the density of the cortical representation is not uniform but higher in regions processing foveal input. The concept of topographic organization therefore does not establish a formal but rather a qualitative property of cortical maps [472]. A review by Kaas [474] of experimental findings about cortical maps provides an overview of their structure and function. Especially

early regions “reflect the order of the receptor sheet with the greatest fidelity” [474]. In general, however, there are redundant maps and each map may not be uniform but contain duplications and distortions. Redundancy can serve the representation of different features (e.g. shapes and motion) or a single feature at different scales (i.e. mapping resolutions). While topographic maps are considered essential for perception, there are also maps that represent information related to action such as those encoding body movement in the motor cortex. Furthermore, the neocortex is not the only brain region where topographic maps have been identified. The *superior colliculus* is an important center for multisensory integration. It contains maps for vision, hearing, somatosensation and movement that are aligned to each other. In particular, it was shown that the development of auditory maps is shaped by visual maps [475, 476]. These findings are directly in line with the main features of the model developed in this work.

Self-Supervised Learning with Topographic Loss The latent space loss considered so far is regularized by the inverse variance of the corresponding component network’s latent space projection to encourage spatially diverse mappings and to prevent convergence to trivial solutions. However, the variance does not provide any direct measure of whether the learned mapping is topographic. At the same time, the topography of a mapping is a global property, while gradient based optimization only processes a small batch with randomly drawn of the data set in each step. The challenge therefore lies in breaking down the topographic preservation of neighborhood to a differentiable loss function that enables gradient-based optimization of the global layout of the space solely based on local information. In the following, we realize this goal by developing a *topographic loss* that optimizes the latent space mapping based on two competing losses. They replace the variance-based regularization term and encode two topographic measures:

- *Neighborhood Loss*: Encodes the of the preservation of neighborhood. The loss grows as samples that lie close in the workspace move apart from each other in the latent space.
- *Separation Loss*: Encodes how well different neighborhoods are separated from each other. The loss grows as neighborhoods that lie in separate parts of the workspace begin to overlap in the latent space.

Both loss terms contain notions of neighborhood and closeness for sets of samples. Applying them to batches of randomly selected points is therefore not possible. Instead, we consider a to our knowledge novel *sphere batching* method that is described in Algorithm 4. Rather than picking arbitrary samples from D_{map} in every batch, sphere batches are sampled around a randomly generated set of S midpoints $\langle c \rangle$, each of which corresponds to a sphere with radius r . The sampling procedure ensures that the spheres in the batch do not overlap. As soon as an appropriate set of sphere centers has been generated, the algorithm retrieves all samples from D_{map} that lie inside the spheres based on their workspace positions and their Euclidean distances from the corresponding centers. This can be efficiently implemented with a k -d tree [477] that is defined on the workspace positions of D_{map} .

Algorithm 4: Sphere Batching**Input:**

D_{map} : A multimodal data set $D_{map} = \{(q_0, \mathbf{I}_0), p_0), \dots, (q_K, \mathbf{I}_K), p_K)\}$

Parameters:

S : Number of spheres per batch
 r : Sphere radius in the workspace
 b : Batch size
 B : Number of batches

Output:

$\langle D_{map}, S, r, b, B \rangle_{\mathcal{S}}$: A set of B sphere batches with tuples $(\langle [\mathbf{I}] \rangle, \langle [q] \rangle, \langle [p] \rangle)$. Each batch contains S non-overlapping spheres with camera images $\langle [\mathbf{I}] \rangle^s = [\mathbf{I}^{s_0}, \dots, \mathbf{I}^{s_K}]$, joint angles $\langle [q] \rangle^s = [q^{s_0}, \dots, q^{s_K}]$ and workspace positions $\langle [p] \rangle^s = [p^{s_0}, \dots, p^{s_K}]$. Each of the lists $\langle [\mathbf{I}] \rangle^s$, $\langle [q] \rangle^s$, $\langle [p] \rangle^s$ in a batch corresponds to a sphere with radius r and contains at most $\lfloor b/S \rfloor$ samples. The first elements in the lists of samples are the sphere centers.

Generate B sphere batches

```

1  $\langle D_{map}, S, r, b, B \rangle_{\mathcal{S}} \leftarrow \emptyset$ 
2 foreach  $k \in [0 \dots B - 1]$  do
   Generate random centers for  $S$  non-overlapping spheres with radius  $r$ 
3  $[(q_k^0, \mathbf{I}_k^0), p_k^0), \dots, (q_k^{S-1}, \mathbf{I}_k^{S-1}), p_k^{S-1}] \leftarrow \text{RandomSpheres}(D_{map}, S, r)$ 
   Generate up to  $\lfloor b/S \rfloor$  random samples in each sphere  $((q_s, \mathbf{I}_s), p_s)$ 
4  $\langle [\mathbf{I}] \rangle_k \leftarrow \emptyset$ ;  $\langle [q] \rangle_k \leftarrow \emptyset$ ;  $\langle [p] \rangle_k \leftarrow \emptyset$ 
5 for  $s \in [0 \dots S - 1]$  do
   Get a list of all samples in the sphere excluding the center
6  $\langle [\mathbf{I}] \rangle_k^s, \langle [q] \rangle_k^s, \langle [p] \rangle_k^s \leftarrow \text{KDTree}(D_{map}).\text{query\_sphere}(((q_s, \mathbf{I}_s), p_s), r) \setminus ((q_s, \mathbf{I}_s), p_s)$ 
   Select up to  $\lfloor b/S \rfloor - 1$  random samples from each list
7  $\langle [\mathbf{I}] \rangle_k^s \leftarrow \text{ChooseRandom}(\langle [\mathbf{I}] \rangle_k^s, \lfloor b/S \rfloor - 1)$ 
8  $\langle [q] \rangle_k^s \leftarrow \text{ChooseRandom}(\langle [q] \rangle_k^s, \lfloor b/S \rfloor - 1)$ 
9  $\langle [p] \rangle_k^s \leftarrow \text{ChooseRandom}(\langle [p] \rangle_k^s, \lfloor b/S \rfloor - 1)$ 
   Add sphere center samples to the beginning of the lists
10  $\langle [\mathbf{I}] \rangle_k^s \leftarrow \text{Append}([\mathbf{I}_k^s], \langle [\mathbf{I}] \rangle_k^s)$ 
11  $\langle [q] \rangle_k^s \leftarrow \text{Append}([q_k^s], \langle [q] \rangle_k^s)$ 
12  $\langle [p] \rangle_k^s \leftarrow \text{Append}([p_k^s], \langle [p] \rangle_k^s)$ 
   Add sphere to batch
13  $\langle [\mathbf{I}] \rangle_k \leftarrow \text{Append}(\langle [\mathbf{I}] \rangle_k, \langle [\mathbf{I}] \rangle_k^s)$ 
14  $\langle [q] \rangle_k \leftarrow \text{Append}(\langle [q] \rangle_k, \langle [q] \rangle_k^s)$ 
15  $\langle [p] \rangle_k \leftarrow \text{Append}(\langle [p] \rangle_k, \langle [p] \rangle_k^s)$ 
16 end
17  $\langle D_{map}, S, r, b, B \rangle_{\mathcal{S}} \leftarrow \langle D_{map}, S, r, b, B \rangle_{\mathcal{S}} \cup (\langle [\mathbf{I}] \rangle_k, \langle [q] \rangle_k, \langle [p] \rangle_k)$ 
18 end

```

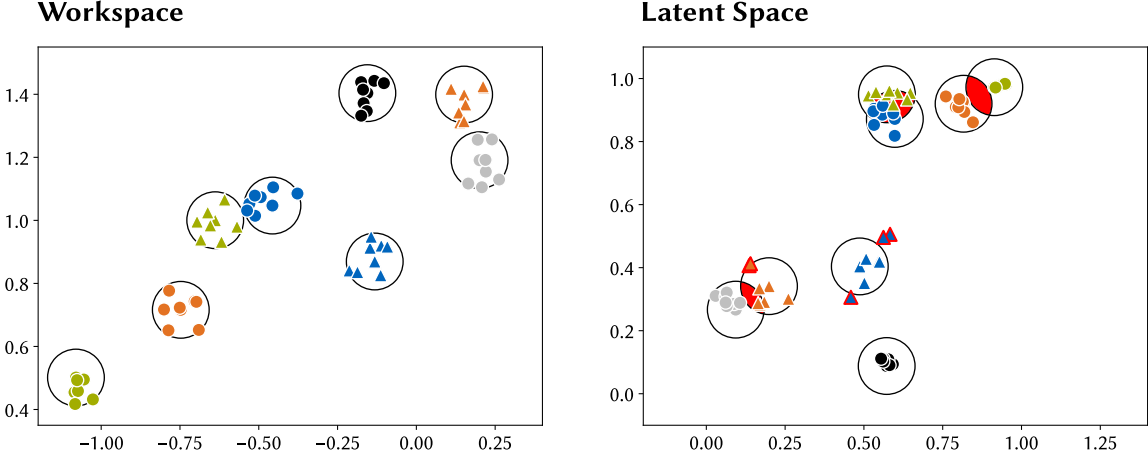


Figure 5.13: Visualization of a sphere batch in the workspace and its latent space projection during an early training epoch. The batch contains $S = 8$ spheres with 8 samples each, resulting in a total batch size of $b = 64$. Neighborhood loss (samples outside their spheres) and separation loss (overlapping spheres) are marked in red. The visualization is based on [464].

Depending on the distribution of D_{map} and the user-defined sphere radius, the number of samples in each sphere will in general be different. In the last step, random subsets of the samples in each of the spheres are drawn to ensure that the total number of samples in the sphere batch does not exceed the maximum user-defined batch size b . An example of a single sphere batch is depicted in Figure 5.13. The clustering of samples around the midpoints of the spheres is clearly visible. In the original workspace, the spheres do not overlap with each other and none of the samples lies outside its corresponding sphere. This is not the case for the latent space projection, where overlapping areas between spheres and samples that are not mapped into their spheres are marked in red color.

Topographic Loss Functions The concepts of neighborhood loss and separation loss defined in the last paragraph can be directly translated into two loss functions \mathcal{L}^{nbh} and \mathcal{L}^{sep} that are defined for latent space projections $\langle [l] \rangle$ of a sphere batch $(\langle [I] \rangle, \langle [q] \rangle, \langle [p] \rangle)$ by C_{vis} or C_{pro} :

$$\mathcal{L}^{nbh}(\langle [l] \rangle) = \frac{1}{S} \sum_{s=0}^{S-1} \frac{1}{|\langle [l] \rangle^s|} \sum_{l \in \langle [l] \rangle^s} \max(0, \|l_{center}^s - l\|_2 - r) \quad (5.28)$$

$$\mathcal{L}^{sep}(\langle [l] \rangle) = \frac{1}{S^2} \sum_{s_1=0}^{S-2} \sum_{s_2=s_1+1}^{S-1} \max(0, 2r - \|l_{center}^{s_1} - l_{center}^{s_2}\|_2) \quad (5.29)$$

The parameters in the equations above are defined as in Algorithm 4. Equation 5.28 sums the distances to the hulls for those samples that are projected to latent space positions outside of their spheres. For each sphere, the sum is normalized by the number of samples it contains. Analogously, the final sum for all spheres is normalized by the total number

of spheres S . Equation 5.29 computes the pairwise distances between all spheres in the batch. Overlap is computed based on the radius r and sphere midpoint distances. The sum of all overlaps is normalized by the squared number of spheres. This is because the maximum number of overlaps for S spheres is equal to $S(S - 1)/2$ and therefore lies in $\mathcal{O}(S^2)$. With the two loss terms from Equations 5.28 and 5.29, the overall topographic loss for a sphere batch can now be computed as follows:

$$\mathcal{L}^{top}(\langle [I] \rangle) = \beta_{nbh} \mathcal{L}^{nbh}(\langle [I] \rangle) + \beta_{sep} \mathcal{L}^{sep}(\langle [I] \rangle) \quad (5.30)$$

The coefficients β_{nbh} and β_{sep} are hyperparameters that make it possible to adjust the influence of the neighborhood loss \mathcal{L}^{nbh} and the separation loss \mathcal{L}^{sep} .

Results of Individual Unsupervised Training The topographic loss from Equation 5.30 can be used to train C_{vis} and C_{pro} completely unsupervised without any target output. The corresponding training procedure is implemented in Algorithm 5. Figure 5.14 shows two examples of latent space representations that were learned over 20 epochs. It

Algorithm 5: Multimodal Unsupervised Learning with Sphere Batching

Input:

- D_{map} : A multimodal data set $D_{map} = \{(q_0, \mathbf{I}_0), p_0), \dots, ((q_K, \mathbf{I}_K), p_K)\}$
- C_{vis} : Component network for vision with randomly initialized weights
- C_{pro} : Component network for proprioception with randomly initialized weights

Parameters:

- E : Number of training epochs
- S : Number of spheres per batch
- r : Sphere radius in the workspace
- b : Batch size
- B : Number of batches
- Γ_{vis} : Optimization parameters for C_{vis}
- Γ_{pro} : Optimization parameters for C_{pro}

```

1 foreach  $e \in [1 \dots E]$  do
    | Iterate over B sphere batches with maximum size b
2   foreach  $\langle [I] \rangle, \langle [q] \rangle \in \langle D_{map}, S, r, b, B \rangle_{\mathcal{S}}$  do
    |   Compute loss gradients with respect to synaptic weights
3      $g_{vis} \leftarrow \nabla_{C_{vis}} \mathcal{L}^{top}(C_{vis}(\langle [I] \rangle))$ 
4      $g_{pro} \leftarrow \nabla_{C_{pro}} \mathcal{L}^{top}(C_{pro}(\langle [q] \rangle))$ 
    |   Update network weights
5      $C_{vis} \leftarrow \text{Optimizer}(C_{vis}, \Gamma_{vis}, g_{vis})$ 
6      $C_{pro} \leftarrow \text{Optimizer}(C_{pro}, \Gamma_{pro}, g_{pro})$ 
7   end
8 end
    
```

is obvious at first glance that both of them do not only reproduce the topography of the workspace very well but that the training is also consistent across the two runs, which is a big difference to the results of purely self-supervised learning in Figure 5.11. Nevertheless, there are still deviations that can be explained by the random initialization of the network weights and the random sampling of sphere batches. Figure 5.15 illustrates the development of the latent space representations of C_{vis} and C_{pro} over time. Remarkably, both networks have learned a good approximation of the workspace topography already by the end of the first epoch. This is also directly reflected in the plots of \mathcal{L}^{nbh} and \mathcal{L}^{sep} in Figure 5.16, where the losses decrease rapidly at the beginning of the training for both component networks. As can be seen in the output of C_{pro} at the end of epoch 20, the perceived quality of the representation may sometimes drop during training. The increased overlap between yellow and red areas in the upper half of the latent space is also expressed by an increase of the separation loss towards the end of the training. These effects can also be explained by the random nature of sphere batching and can be easily mitigated by selecting the best epoch at the end of a training run.

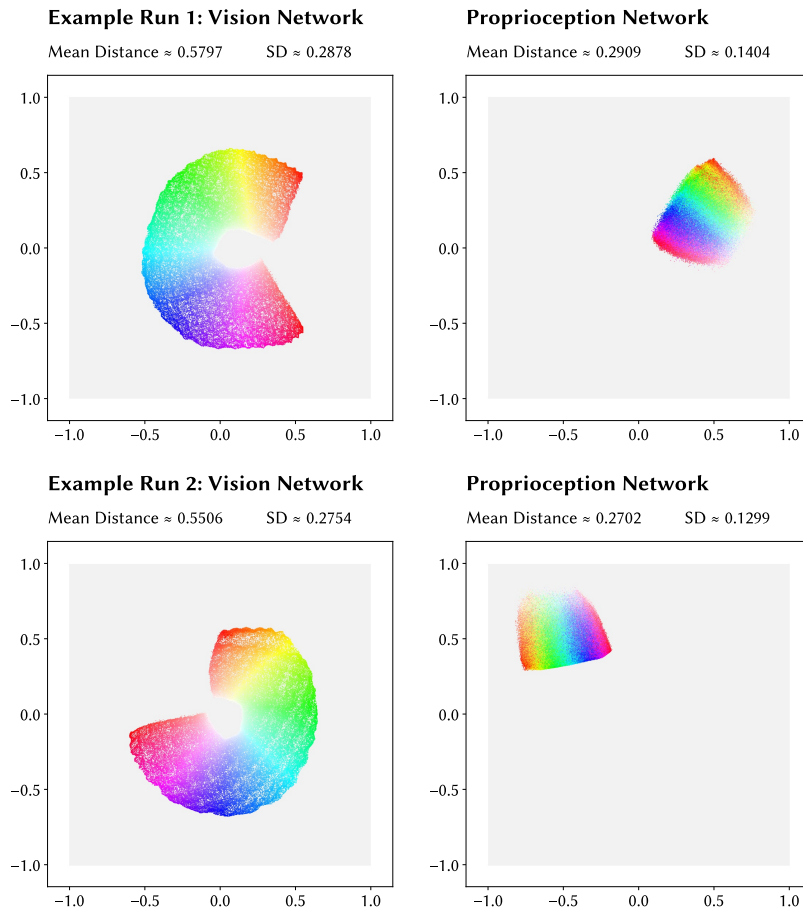


Figure 5.14: Visualization of the results of two selected runs of Algorithm 5 with the topographic loss \mathcal{L}^{top} from Equation 5.30. Shaded areas indicate the output range of the $\tanh(\cdot)$ activation function.

5.3 Self-Supervised Learning of Deep Multisensory Neural Maps

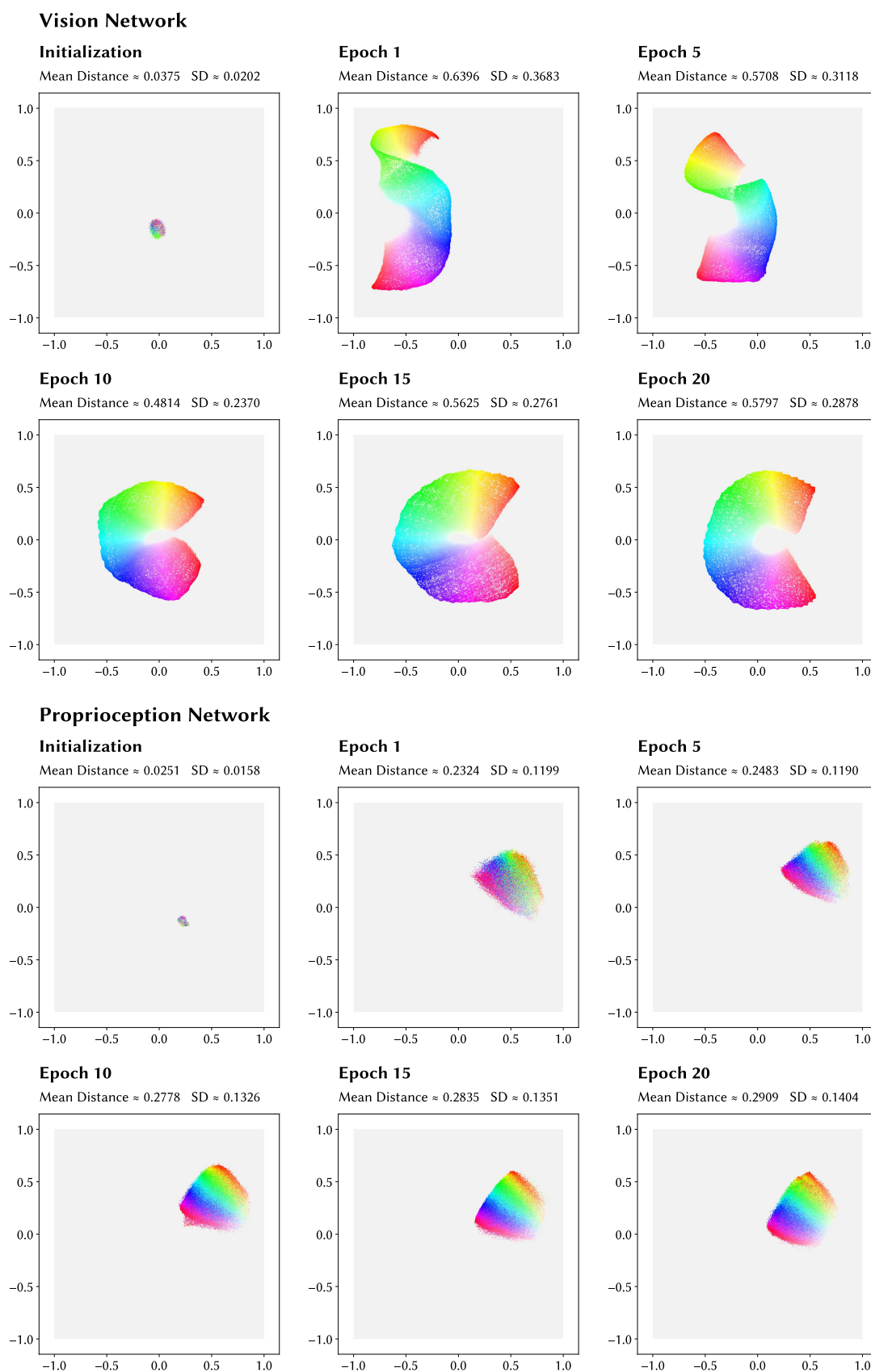


Figure 5.15: Progress of the training over selected epochs in Example Run 1 from Figure 5.14.

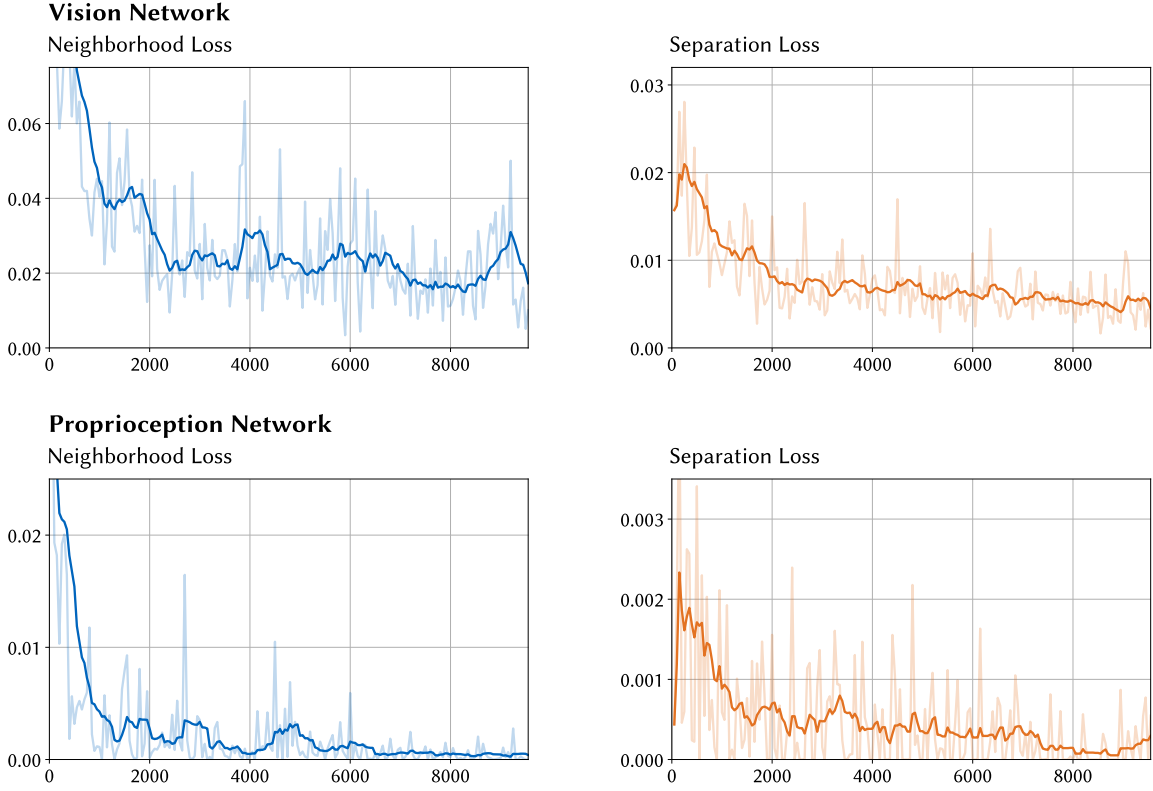


Figure 5.16: Development of the neighborhood losses and separation losses during Example Run 1 from Figure 5.14. Each graph shows both the original data (one data point represents 50 training steps) and a running average with windows size 10.

Results of Joint Self-Supervised Training The topographic loss enables both C_{vis} and C_{pro} to learn topographic maps of the workspace. But unlike the maps learned with the self-supervised training method from Algorithm 3, the maps are not aligned to each other. This directly motivates the definition of the topographic map loss functions \mathcal{L}_{vis}^{map} and \mathcal{L}_{pro}^{map} that combine self-supervised learning and topographic loss:

$$\begin{aligned} \mathcal{L}_{vis}^{map}(C_{vis}\langle[\mathbf{I}]\rangle, C_{pro}\langle[q]\rangle) &= \\ &= \begin{cases} \mathcal{L}^{top}(C_{vis}\langle[\mathbf{I}]\rangle) & \text{if } epoch \leq E_{warm-up} \\ \alpha_{vis} \text{MSE}(C_{vis}\langle[\mathbf{I}]\rangle, C_{pro}\langle[q]\rangle) + \mathcal{L}^{top}(C_{vis}\langle[\mathbf{I}]\rangle) & \text{if } epoch > E_{warm-up} \end{cases} \end{aligned} \quad (5.31)$$

$$\begin{aligned} \mathcal{L}_{pro}^{map}(C_{vis}\langle[\mathbf{I}]\rangle, C_{pro}\langle[q]\rangle) &= \\ &= \begin{cases} \mathcal{L}^{top}(C_{pro}\langle[q]\rangle) & \text{if } epoch \leq E_{warm-up} \\ \alpha_{pro} \text{MSE}(C_{vis}\langle[\mathbf{I}]\rangle, C_{pro}\langle[q]\rangle) + \mathcal{L}^{top}(C_{pro}\langle[q]\rangle) & \text{if } epoch > E_{warm-up} \end{cases} \end{aligned} \quad (5.32)$$

Both loss functions can be adjusted to only apply topographic loss during a warm-up period at the beginning of the training. This ensures that random artifacts that are only caused by the initialization of networks do not slow down the learning process. The alignment of both component networks therefore only begins after they have acquired an early meaningful representation of the workspace. \mathcal{L}_{vis}^{map} and \mathcal{L}_{pro}^{map} can be directly applied in Algorithm 3 when the batch operation in Line 3 is at the same time replaced with the sphere batching method defined in Algorithm 4. Figure 5.17 depicts the results of three runs with different parameters over 50 epochs each. Already basic settings without warm-up and equal weights for the MSE term in both networks yield a common latent space representation. Its layout can be controlled by scaling the MSE. In Example Run 2, the parameter value α_{vis} in \mathcal{L}_{vis}^{map} is very small while α_{pro} in \mathcal{L}_{pro}^{map} is comparatively large. As a result, the representation learned by C_{pro} adapts to that of C_{vis} . This effect reproduces the findings on topographic map alignment in the brain mentioned at the beginning of this section, where auditory maps have been found to be shaped by visual maps. Example Run 3 was executed with equal learning rates but a warm-up period $E_{warm-up}$ of 20 epochs. The overall result resembles that of the first run but the mean error is bigger despite comparable mean distance values. However, this issue is not directly related to the warm-up time, which becomes evident in Figure 5.18, where the progress during training in Example Run 3 is summarized. At the end of the last warm-up period, the results are in line with those from Figure 5.14. Only one epoch later, both representations are closely aligned and the latent space error compares well to Example Run 1. In later epochs, the alignment decreases and the latent space error grows. Possible reasons are the competition between the MSE loss and the topographic loss, as well as the stability issues observed earlier in the purely self-supervised learning setup. This behavior can be prevented by, for example, freezing one network after reaching a defined training epoch and continue to train only the other. The actual development of the loss over the complete training time for both Example Run 1 and Example Run 3 is plotted in Figure 5.19. At the end of the warm-up period, there is an immediate increase in the total loss as the MSE term is activated. The quick recovery to loss values comparable to those in Example Run 1 are in line with the quick alignment of the latent spaces after only a single training epoch.

Conclusion In summary, the proposed training procedure, which is based on a novel topographic map loss function and a corresponding sphere sampling algorithm, can successfully learn common latent space projections from multimodal input. It is therefore an essential component for building MHNNs. Compared to standard batching where each training batch is generated from arbitrary samples in the data set, the introduced sphere batching method encodes information about the spatial structure of the input space. This is because the sampling of points within a sphere and ensuring that spheres do not overlap requires a distance metric between samples. In Algorithm 4, this metric is derived from the Euclidean distances between samples in the workspace. Sphere sampling thereby implicitly leverages ground truth data for training. It is important to

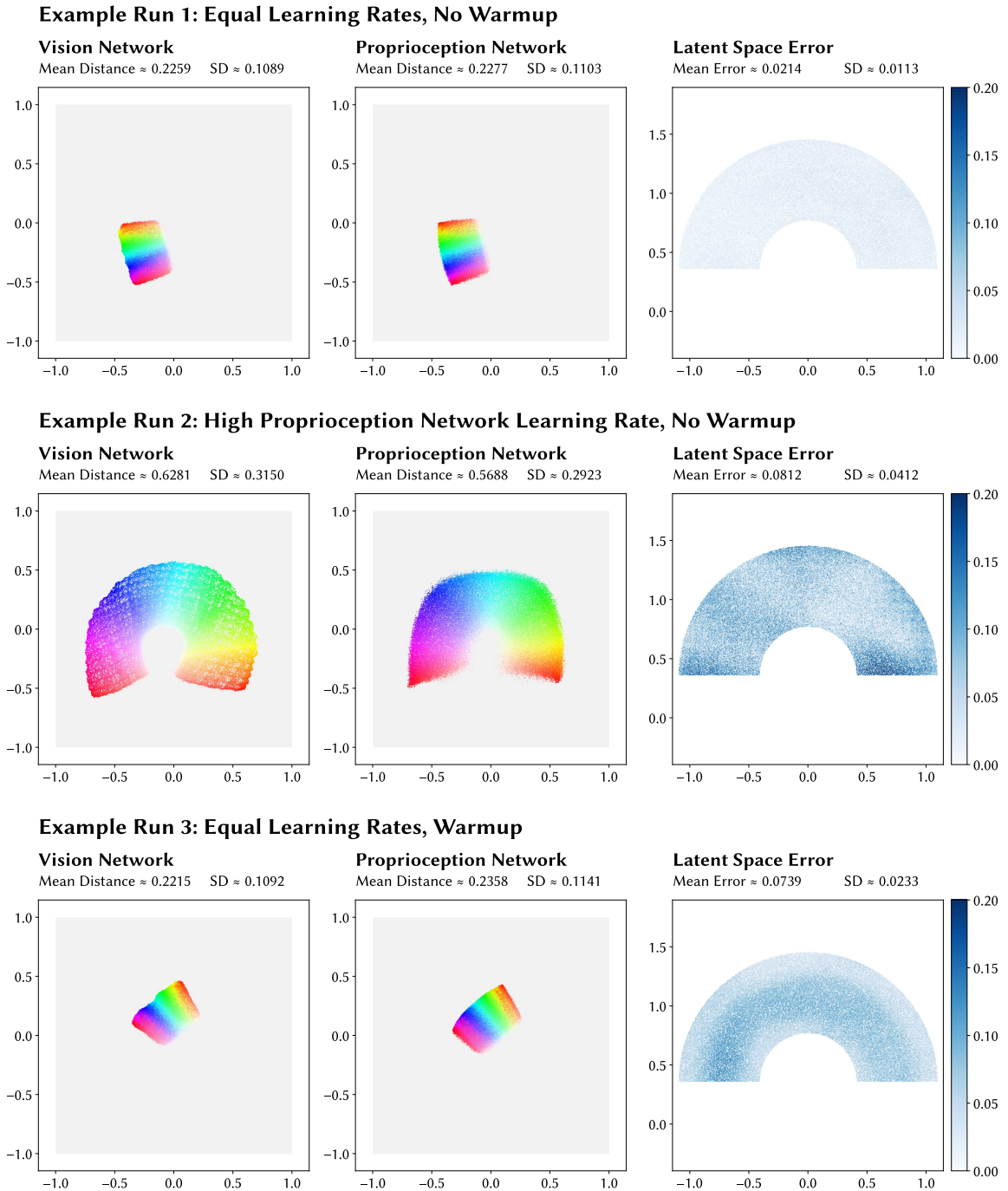


Figure 5.17: Visualization of the results of three selected runs of Algorithm 3 with different parameters for the topographic map losses \mathcal{L}_{vis}^{map} and \mathcal{L}_{pro}^{map} from Equations 5.31 and 5.32. The upper limits of the the colorbars are saturation values and do not correspond to the maximum error.

5.3 Self-Supervised Learning of Deep Multisensory Neural Maps

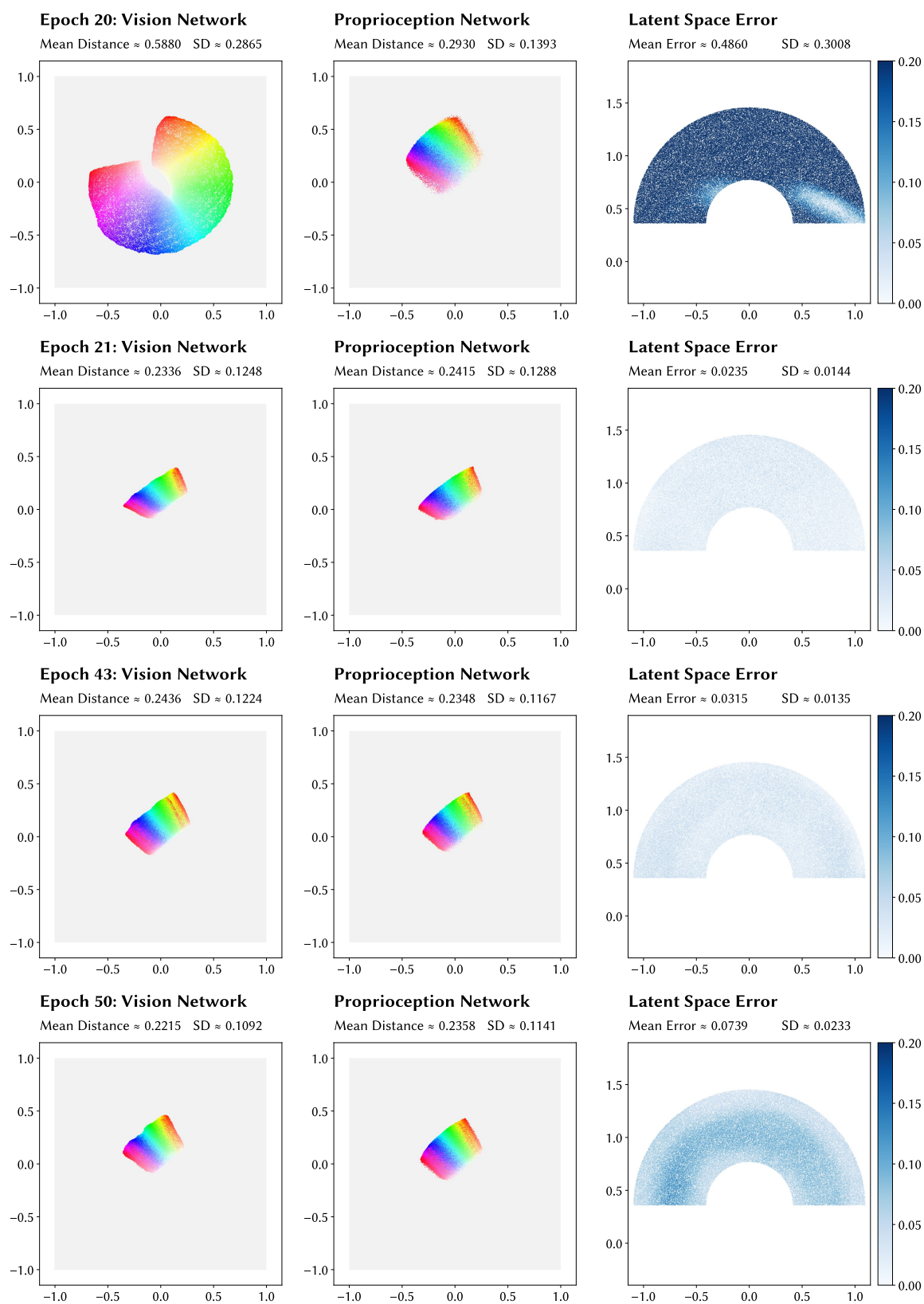


Figure 5.18: Progress of the training over selected epochs in Example Run 3 from Figure 5.17. The upper limits of the the colorbars are saturation values and do not correspond to the maximum error.

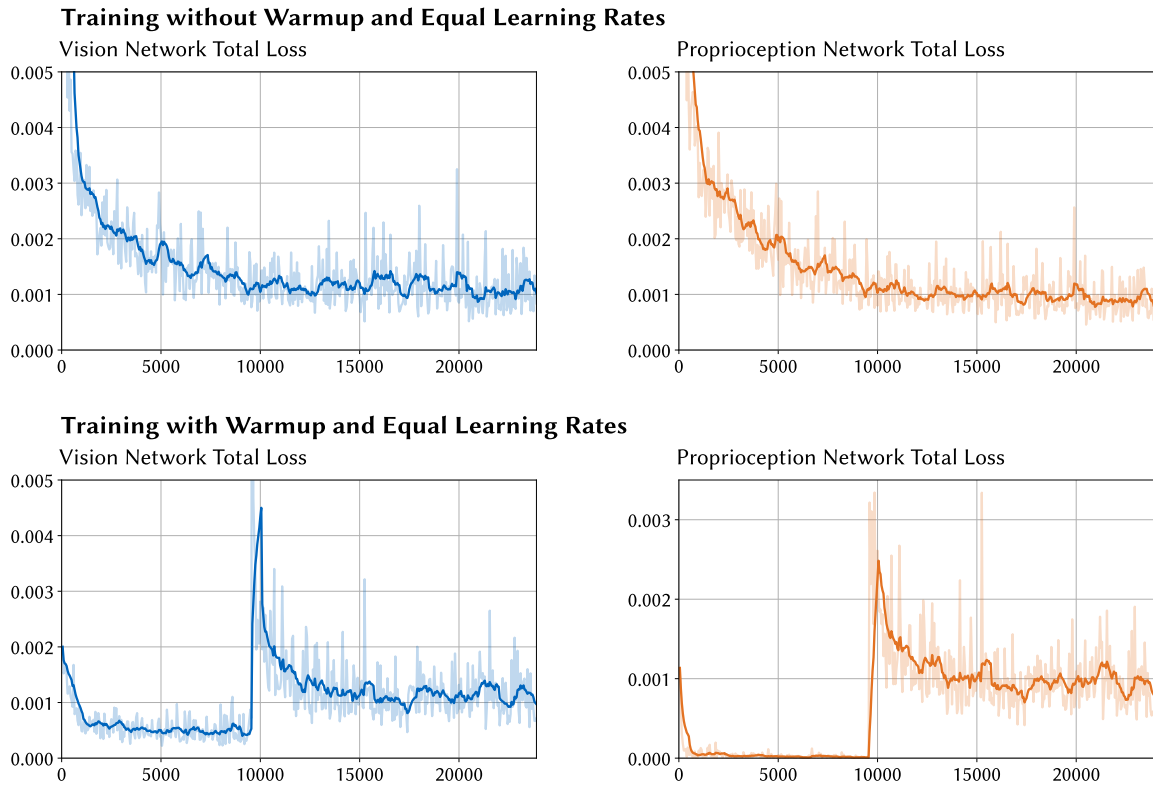


Figure 5.19: Development of the total loss during Example Run 1 and Example Run 3 from Figure 5.17. Each graph shows both the original data (one data point represents 50 training steps) and a running average with window size 10.

note, however, that this is not a shortcoming of the proposed method but rather a key feature because it is a means to encode the desired structure of the latent space. As the results from Figure 5.11 illustrate, the latent space representations that emerge during purely self-supervised learning are very different from the original workspace topography. Moreover, neighborhood relations are also encoded in the brain, where, for example, the retinotopic mapping mentioned earlier directly projects from the retina to the visual cortex. Another important aspect is that the sensory input streams of embodied systems are always continuous. Stimuli that are temporally close to each other are therefore automatically bounded by a small sphere in the input space. In this sense, sphere batching can be thought of as mimicking the perceptual input patterns of embodied systems in a batched representation that is amenable gradient-based optimization. The alignment of the maps shaped by topographic loss is achieved through self-supervised learning. While care must be taken to ensure stability and convergence, the data from the training runs show that both networks converge quickly to a common latent space topography.

The trained networks effectively perform a dimensionality reduction of their input spaces, which is why the proposed method is closely related to SOMs [127]. However, the latter only support unsupervised training. While the learning of common latent spaces is possible as in the already discussed work by Lalle and Dominey [443], the

mode of representation is completely different. The networks trained in this work return continuous output values and therefore can represent data at any resolution. In SOMs, by contrast, the space is discretized as every output value is represented by an individual neuron. It can be argued that this form of representation is more biologically plausible. But in principle, the analog output of the component networks can also be projected onto a neural sheet. What SOMs completely lack, however, is the learning of feature hierarchies. This is naturally possible with method introduced here. On a broader perspective, topographic loss can also be seen as a method for preserving topological properties in the input space. Learning topology-preserving homeomorphic mappings is an active field of research in machine learning and results in this direction have been published only recently [478, 479].

6

Developmental Embodied Learning

A special feature of the MHNN architecture introduced in the last chapter is the independence of the individual network modules of each other. In principle, all component networks can be trained separately as demonstrated in Section 5.3, where two networks for multimodal visual and proprioceptive input learned individual latent space representations solely based on topographic loss. The fusion of data streams from the individual networks only becomes relevant when they jointly provide the information required to execute a task that is implemented by a hub network. At the same time, pre-trained component networks can be aligned by establishing lateral connections or through self-supervised learning as described in Algorithms 1 and 3. These examples indicate that MHNNs not only encode *spatial separation* of information as expressed by the diversification into different network modules but also give rise to a *temporal order* that is determined by the sequence of learning algorithms applied to individual modules and the complete network. In this chapter, we will investigate how the temporal order of learning can be formalized and actively employed to enhance the learning process. In fact, scheduled learning over subsequent phases is a key principle in the development of living creatures. In Section 6.1, we first review the biological background and previous work in robotics before we introduce the concept of *training protocols*. The findings will be applied in the implementation of two different types of hub networks for multisensory integration and inverse kinematics learning in Sections 6.2 and 6.3, respectively. Both networks are developed and evaluated in virtual neurorobotics experiments for the NRP.

6.1 Modeling Biological Development with Training Protocols

The human brain is able to learn and adapt throughout its lifetime. After birth, different skills develop in different phases. For example, babies learn to crawl before they learn to walk. The laws and processes that govern the gradual and ordered acquisition of skills are summarized under the term *cognitive development* [480]. An influential theory in this field was proposed by Piaget [481], who postulated that child development encompasses four subsequent phases: “(1) the sensorimotor stage from birth to 2 years, (2) the preoperational stage from 2 to 7 years, (3) the concrete-operational stage from 7 to 12 years, and (4) the stage of formal operations that characterizes the adolescent and the adult” [482]. Most related to this work is the first stage, which is further sub-divided into different phases that start with the refinement of repetitive reflexive movements. In the last phases, children become capable of pursuing goal-directed actions with trial-and-error learning eventually being replaced by mental imagery. Continuing from there, the subsequent three stages describe the development of language and abstract logical reasoning [482]. Interaction with the environment is essential during the sensorimotor stage, which clearly highlights the importance of embodiment. It can therefore only be fully modeled and investigated with a neurorobotics approach. While the staged model of cognitive development is not the only theory available, it implicitly reflects the fact that the development of the brain has also found to be organized in different phases and provides a guideline for technical implementations.

Brain Development and Body Growth

Developmental processes observed in the human brain have already motivated the formulation of Algorithm 1 in Section 5.2, where unused synapses are pruned based on sensory input correlations after an initialization phase. As already mentioned in the last chapter, neuroscientific evidence for synaptic pruning was published by Huttenlocher [455], who found that the synaptic density in newborns increases during the first two years of live and is up to 50 % higher than in adults. The existence of a time window with increased synaptic density means that postnatal brain development is temporally structured. During *critical periods* “brain circuits that subserve a given function are particularly receptive to acquiring certain kinds of information or even need that instructive signal for their continued normal development” [483]. Anatomical changes in the brain resulting from them are more substantial and lasting than at other points in the life of an individual. In a review, Hensch [483] summarizes evidence for critical periods for all main sensory modalities and identifies important characteristics. In line with the theory of embodiment, a key factor is sensory experience that is gained during interaction with the environment. Importantly, it is not only required to drive development within a critical period but also drives the progression of periods. At the same time, the different pathways and information processing hierarchies in the brain also impose an order on the timing and

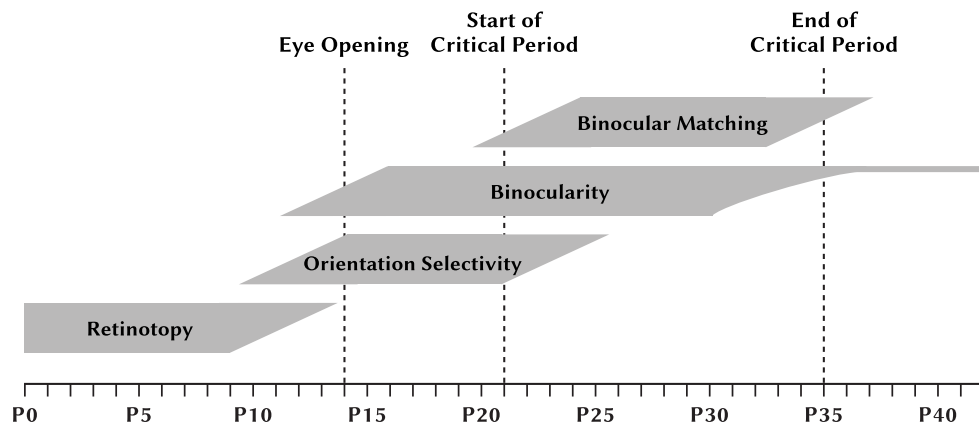


Figure 6.1: Stages in the development of the primary visual cortex of the mouse. During the critical period, the response properties of neurons become aligned between both eyes [484]. Adapted from [484] with permission from Elsevier.

duration of the critical periods in different brain regions. Figure 6.1 depicts a critical period in the postnatal development of the mouse brain. After the formation of the retinotopic map, the neurons of both eyes begin develop their receptive fields and become selective for different orientations. During a critical period, the response properties become aligned between the two eyes [484].

The development of the body proceeds in parallel to that of the brain. Its growth is not only of quantitative but also of qualitative nature. For example, visual acuity drastically increases during the first six months after birth [482]. Visual sensory input thereby undergoes considerable changes. Similarly, the growth of the body and its muscles necessitates adaptation of motor control until adulthood. Starting from the fetal phase and lasting until several months after birth, infants start to perform *general movements* that arise from the exploration of the individual joints [485]. From a robotics perspective, this seems similar to motor babbling where sensory input is generated by random joint movements.

Developmental Robotics

The study of developmental processes in living creatures not only sheds light on development itself but also the cognitive skills it gives rise to. Designing and implementing components of cognitive systems by mimicking developmental progression rather than trying to reproduce their final functions can therefore yield biological insight and at the same time potentially produce better results. As pointed out by Cangelosi and Schlesinger [486], already Turing [487] noted:

“Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child’s? If this were then subjected to an appropriate course of education one would obtain the adult brain. [...] We have thus divided our problem into two parts. The child-programme and the education process.”

The modeling and simulation of biological development are at the core of *developmental robotics*, a branch of robotics that is closely related to neurorobotics and the theory of embodiment. Applying its methods and principles to neurorobotics makes it possible to study closed-loop systems in full biological detail by adding the dimension of time as an experimental parameter. Importantly, the systematic study of many developmental effects such as body growth is only possible in virtual neurorobotics experiments that are not limited by the constraints of physical robots.

The multitude of factors governing development as well as the diversity of theories that have been put forward are reflected by a broad range of different directions of research in developmental robotics. Related work was already partly discussed in Section 3.2 in the context of embodiment, highlighting the close connection between these areas. Lungarella et al. [488] and Asada et al. [261] provide extensive reviews and highlight relevant findings from biology and cognitive science. As outlined in Figure 6.2, research in developmental robotics has in particular been influenced by the phases of Piaget [481]. The focus of this work is on early stage individual development starting from the phase of sensorimotor mapping that was addressed in Section 5.3. An important factor of developmental progression in this stage are constraints imposed by the body. For example, the visual system of infants is not fully developed at birth [489]. However, the decreased sensory resolution may actually be beneficial during early development since it limits the amount information that needs to be processed [488, 490]. Findings from both biology and robotics on visual development furthermore suggest that correlated sensory input acquired in a closed PCA loop from the environment is equally important [258, 491]. According to a theory originally put forward by Bernstein [492], motor control is also affected by a form of self-induced developmental progression. He postulated that to reduce task complexity during the learning of new skills, joints far away from the body are initially *frozen* and the dimensionality of the control problem is thereby decreased. As learning progresses,

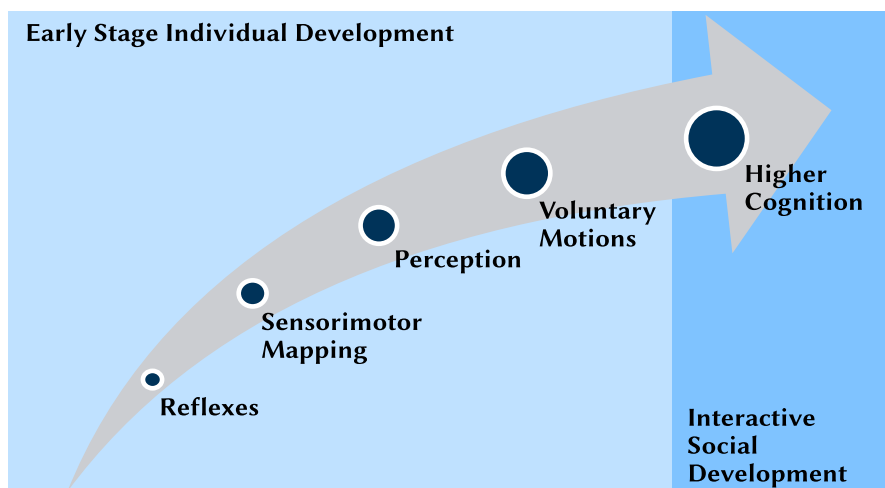


Figure 6.2: Phases of functional development identified by Asada et al. [261] for research in cognitive developmental robotics. The early phases correspond closely to Piaget’s sensorimotor stage.

they are gradually *freed* again [488]. While experimental evidence is mixed and depends on many factors including the type of skill, there are nevertheless results that support the existence of a freezing mechanism [493]. There is also evidence, however, that the number of DoFs can decrease rather than increase over time [488]. But even though the biological significance and implementation of the freezing and freeing of DoFs are not yet fully elucidated, the principle has already been successfully applied in robotics. In a systematic study on robotic developmental progression, for example, Gómez et al. [494] implemented a training procedure that gradually increased the resolution of a robot's sensory inputs, subsequently freed individual robot joints, and incrementally augmented a neural network-based controller. Learning a foveation task with this method was faster compared to a system where all modalities were initialized with full complexity. Ivanchenko and Jacobs [495] showed improved performance in a simple trajectory following task when two of three joints were initially frozen and feedback gains decreased during the training process.

Training Protocols

The findings from biology, neuroscience and robotics clearly highlight the importance of development. It is not only a natural extension of the concept of embodiment but also provides biologically grounded tools for guiding learning processes and managing task complexity. Now, we will formalize this idea with two key concepts that are derived from the principles of developmental progression as discussed in the last two subsections:

- *Schedules*: Development in the brain is shaped by critical periods that follow carefully adjusted schedules. Different brain regions undergo their critical period at different points in time. The order also depends on the hierarchical relationships between regions.
- *Stages*: Development occurs in stages (e.g. walking is preceded by crawling). In particular, learning starts from simple tasks as indicated in Figure 6.2 and proceeds to complex ones. While learning motor tasks, individual joints may be frozen to reduce task complexity.

While schedules are defined intrinsically in terms of brain development, stages are defined extrinsically in terms of task complexity. Both concepts can be directly mapped to corresponding building blocks of the MHNN architecture from Chapter 5. The modality-specific component networks are trained to transform input data into a latent space representation before hub networks are trained on concrete tasks based on their output. This order of training defines a schedule. The learning processes in the hub networks can be further sub-divided into subsequent stages with increasing complexity. Both schedules and stages define sequences of steps for the training of MHNNs that control an embodied system. In this work, we will refer to these sequences as *training protocols*. Figure 6.3 depicts an example of a training protocol for target reaching, based on visual and proprioceptive input. The construction of the network is based on a schedule with four steps. At the beginning, two component networks for proprioception and vision

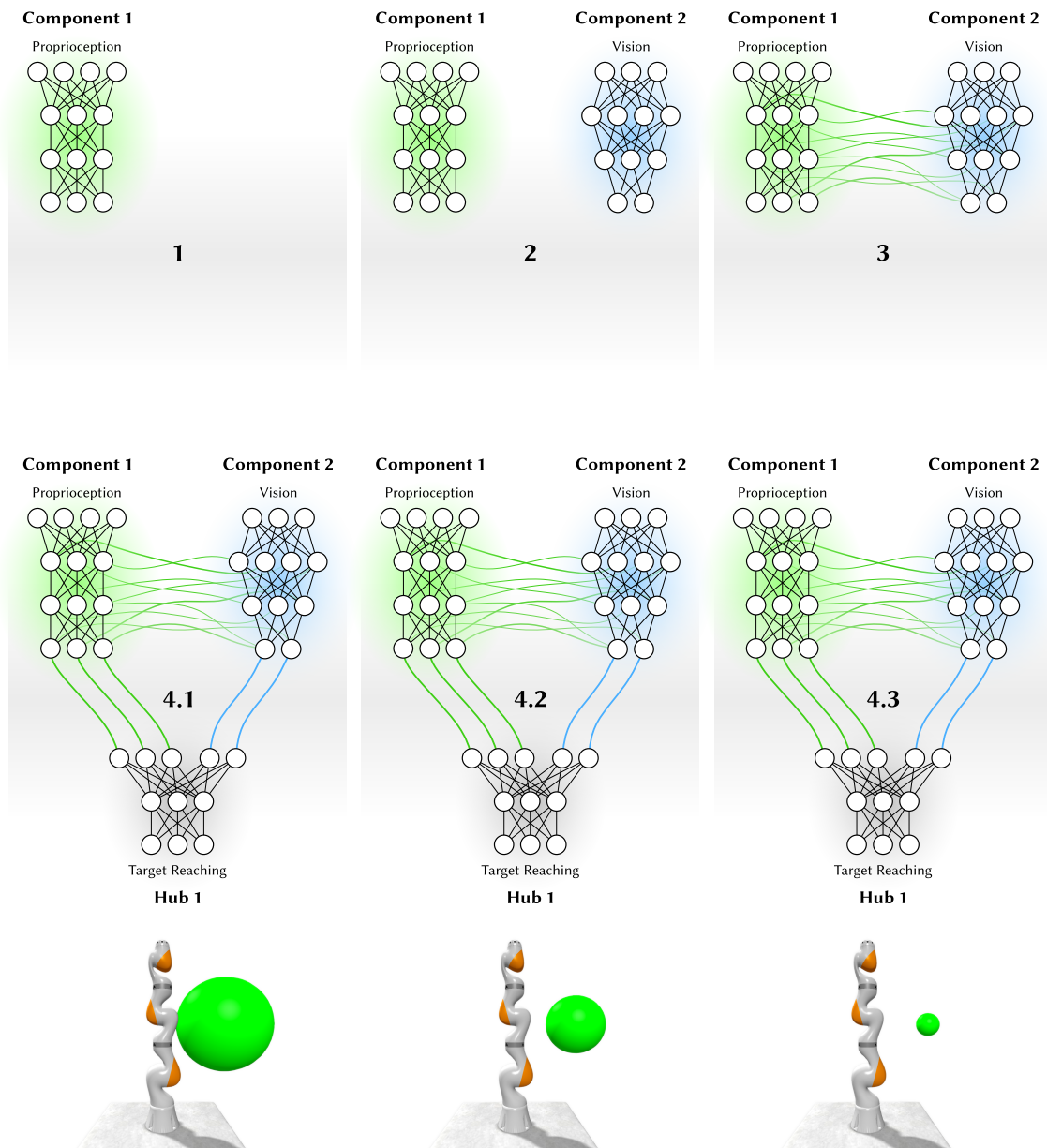


Figure 6.3: Schematic illustration of a training protocol. The protocol is comprised of a schedule with four steps, the last of which encompasses three stages. In steps 1 – 3, two component networks for proprioception and vision are first trained individually before lateral connections are added from the former to the latter. The hub network added in step 4 is trained over three stages to move a robot’s tool center point to a desired workspace position. As indicated by the green spheres, the required positioning accuracy increases from stage to stage.

are trained independently of each other. Alternatively, both networks could also be trained jointly with Algorithm 5 and then be added in a single step. In step 3, lateral connections are computed according to Algorithm 1. The hub network that implements the actual reaching task is instantiated in the last step of the protocol and trained in three stages. As indicated by the decreasing radius of the green spheres, the required positioning accuracy increases with every stage, which means that the complexity of the task gradually increases, too. Depending on the type of the embodied system, its input modalities and the tasks to be learned, the resulting training protocol will be different. However, the general strategy of training component networks and hub networks independently of each other and possibly over several stages remains identical. Next, we will take advantage of training protocols to implement two neurorobotics experiments that highlight how hub networks can fuse multimodal input from component networks, and how training in stages can modulate the learning process, respectively.

6.2 Neuromorphic Data Fusion

The focus in Chapter 5 was on the definition of the architecture schema for MHNNs and on the design of a novel training procedure for latent space representations that are synchronized across component networks. Such representations lend themselves extremely well to data fusion tasks because there is no further alignment of input signals required. As outlined earlier, there is also a huge body of biological evidence for the existence of aligned sensory maps in the superior colliculus, which is an important center of multisensory integration [496]. Neurons in this region have been found to respond particularly well to stimuli that originate from the same location and occur at the same time. In addition to these two principles, which are commonly referred to as the “spatial rule” and the “temporal rule”, there is also an “inverse effectiveness rule” which establishes an inversely proportional relationship between a neuron’s unimodal and multimodal responses [497]. In this section, we will focus on the first two rules and demonstrate how probabilistic population codes (PPCs) enable multisensory integration with aligned topographic maps. PPCs were originally proposed by Ma et al. [498] and are based on spiking neuron models. We will instantiate the model for a prototypical localization task as an experiment in the NRP. The results reported in this section have been partly published in [499].

Multisensory Integration with Maximum Likelihood Estimation

One of the primary motivations for data fusion, which is commonly referred to as multisensory integration in the context of brain research, is to reduce the uncertainty of a state estimate by combining multiple complementary measurements. It is therefore an essential concept for modeling and understanding how the brain integrates data from different sensory modalities such as vision, hearing, proprioception and touch into a unified and coherent perception of the environment. Findings from psychophysical experiments

indicate that the brain integrates sensory information based on maximum likelihood estimation (MLE) [500]. In the following, we will derive an integrated estimate for a stimulus (e.g. the location of an object) from individual estimates provided by two different modalities (e.g. vision and sound). Let s , s_1 and s_2 denote a stimulus and its estimates that are derived from two input modalities. Further assume that both s_1 and s_2 are subject to independent Gaussian noise [501]:

$$s_1 = s + \epsilon_1 \quad \text{with} \quad \epsilon_1 \sim \mathcal{N}(0, \sigma_1^2) \quad \Rightarrow \quad s_1 \sim \mathcal{N}(s, \sigma_1^2) \quad (6.1)$$

$$s_2 = s + \epsilon_2 \quad \text{with} \quad \epsilon_2 \sim \mathcal{N}(0, \sigma_2^2) \quad \Rightarrow \quad s_2 \sim \mathcal{N}(s, \sigma_2^2) \quad (6.2)$$

The two equations above directly yield the probability for a specific pair of observations from both modalities s_1 and s_2 :

$$\begin{aligned} p(s_1, s_2 | s) &= p(s_1 | s) \cdot p(s_2 | s) \\ &= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(s_1 - s)^2}{2\sigma_1^2}\right) \cdot \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{(s_2 - s)^2}{2\sigma_2^2}\right) \end{aligned} \quad (6.3)$$

The MLE for s is determined by computing a stimulus value for s that maximizes the probability $p(s_1, s_2 | s)$. Computing the log-likelihood of $p(s_1, s_2 | s)$ and taking the derivative with respect to s yields:

$$\log p(s_1, s_2 | s) = -\frac{1}{2} \log 2\pi\sigma_1^2 - \frac{(s_1 - s)^2}{2\sigma_1^2} - \frac{1}{2} \log 2\pi\sigma_2^2 - \frac{(s_2 - s)^2}{2\sigma_2^2} \quad (6.4)$$

$$\frac{d}{ds} \log p(s_1, s_2 | s) = \frac{1}{\sigma_1^2} (s_1 - s) + \frac{1}{\sigma_2^2} (s_2 - s) \quad (6.5)$$

Based on Equation 6.5, the MLE for s can now be computed as follows:

$$\frac{d}{ds} \log p(s_1, s_2 | s) \stackrel{!}{=} 0 \quad \Rightarrow \quad s = \frac{\frac{1}{\sigma_1^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} \cdot s_1 + \frac{\frac{1}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} \cdot s_2 \quad (6.6)$$

As it turns out, the MLE for s is simply a weighted sum of the two estimates s_1 and s_2 . Importantly, the variance σ^2 of s is always lower than the variances σ_1^2 and σ_2^2 of the individual estimates:

$$\begin{aligned} \sigma^2 &= \text{var}\left(\frac{\frac{1}{\sigma_1^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} \cdot s_1 + \frac{\frac{1}{\sigma_2^2}}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} \cdot s_2\right) = \frac{1}{\left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}\right)^2} \left(\frac{1}{\sigma_1^4} \text{var}(s_1) + \frac{1}{\sigma_2^4} \text{var}(s_2)\right) \\ &= \frac{1}{\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}} = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2} \leq \sigma_i^2 \quad \text{for } i \in \{1, 2\} \end{aligned} \quad (6.7)$$

It is interesting to note that the derived MLE is actually identical to a maximum a posteriori estimation if the probability distribution $p(s)$ of the stimulus is uniform.

Probabilistic Population Codes

The fact that the brain integrates sensory information by likelihood maximization raises the question how Equations 6.6 and 6.7 can be implemented by neural circuits. As it turns out, even a single multi-compartmental neuron can perform the required computations by representing individual estimates in its dendrites [502]. However, this requires adjusting the neuron's morphology and its dynamic properties specifically for the data fusion task. We will therefore instead consider an alternative model by Ma et al. [498] called probabilistic population codes (PPCs) that can be implemented with simple LIF point neurons. It encodes the MLE in the structure of the network rather than in the morphology and dynamics of individual neurons, which makes it compatible to a broad range of neuromorphic processors. The following description of the PPC model is based on the original publication by Ma et al. [498].

Stimulus Encoding and Reconstruction At the core of PPCs is the representation of estimates about a stimulus with population activity codes as illustrated in Figure 2.3. Figure 6.4 depicts an example of a population code with six neurons. For an input stimulus s , the mean firing rates $r(s, s_n^*)$ of the individual neurons in the population are determined by identically shaped Gaussian tuning curves:

$$r(s, s_n^*) = \frac{1}{\sqrt{2\pi\sigma_r^2}} \exp\left(-\frac{(s - s_n^*)^2}{2\sigma_r^2}\right) \quad (6.8)$$

s_n^* denotes the preferred stimulus of neuron n and σ_r the tuning curve width. In the example from the figure, the firing rates resulting from the stimulus value indicated by the black triangle are plotted at the preferred stimulus location of each neuron. For a modality m that is represented by N neurons with actual firing rates r_m^n , we will in the following denote the population code as $r_m = (r_m^1, \dots, r_m^N)$. We will further assume that all neurons within a population fire independently of each other with Poisson-distributed rates [498]:

$$p(r_m | s) = \prod_{n=1}^N p(r_m^n | s) = \prod_{n=1}^N \frac{r(s, s_n^*)^{r_m^n} \cdot e^{-r(s, s_n^*)}}{r_m^n!} \quad (6.9)$$

By applying Bayes' theorem, the probability of the stimulus $p(s | r_m)$ can be reconstructed from the population code r_m :

$$p(s | r_m) = \frac{p(r_m | s) \cdot p(s)}{p(r_m)} \quad (6.10)$$

$p(r_m)$ can be computed by marginalizing $p(r_m | s)$ over s . In particular, we will assume a uniform prior for s . The reconstructed probability density of an estimate for s is shown in Figure 6.5. In the population code on the left side of the figure, the confidence of the estimate for an input stimulus s is encoded by the intensity of the neurons' activity. More reliable estimates therefore result in higher firing rates and a higher *population gain* g_m .

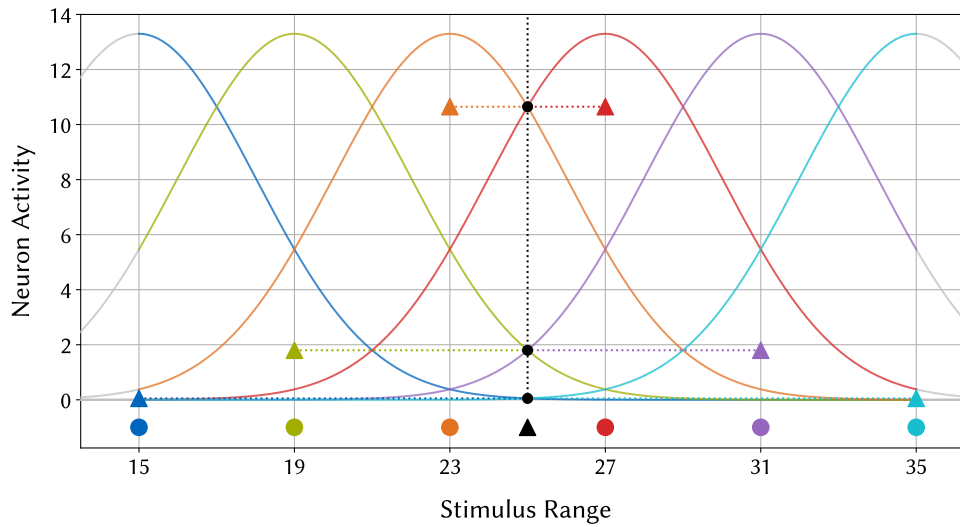


Figure 6.4: Example of neural population coding with tuning curves. Input stimuli lie in the interval $[15, 35]$ and are represented by a population of six neurons as indicated by the colored dots at the bottom of the graph. The position of each neuron on the x -axis corresponds to its preferred stimulus value. All neurons have identically shaped Gaussian tuning curves that are centered around the preferred stimulus values. The triangles correspond to the population activity for one concrete stimulus value $s = 25$.

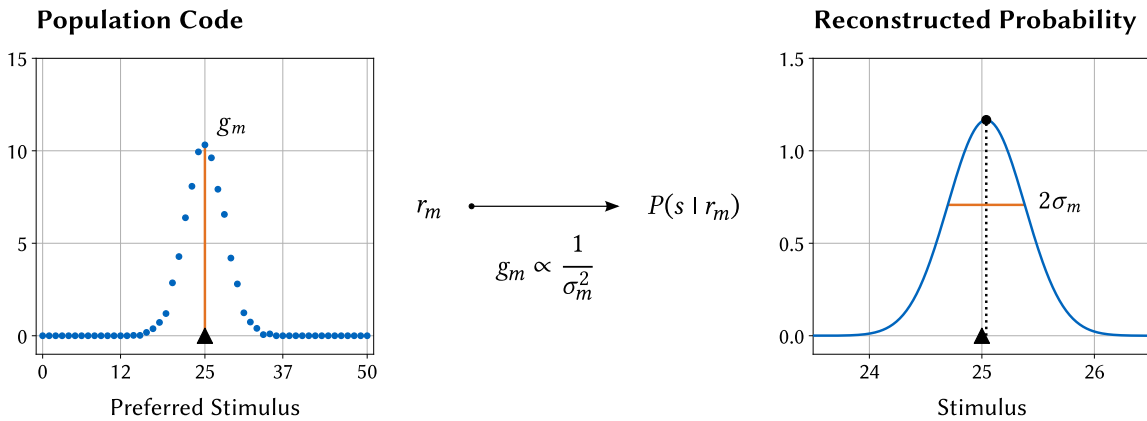


Figure 6.5: Stimulus reconstruction from PPCs. *Left:* Firing rates of the individual neurons in the population. The ground truth input stimulus $s = 25$ is indicated by the black triangle. More reliable estimates are represented by higher firing rates. The confidence is therefore encoded in the gain g_m of the population response. As indicated by the orange line, g_m corresponds its peak height. *Right:* Reconstructed probability density function for $p(s | r_m)$. The variance σ_m^2 is inversely proportional to the population gain. The figure is based on [498].

Note that changes in the gain only affect the peak height of the population activity but not its width. As illustrated in the graph on the right, the reconstructed stimulus is normally distributed with a variance σ_m^2 that is inversely proportional to the population gain g_m :

$$g_m \propto \frac{1}{\sigma_m^2} \Leftrightarrow g_m = \rho \frac{1}{\sigma_m^2} \text{ for } \rho \in \mathbb{R} \quad (6.11)$$

In summary, higher confidence as expressed by increased firing rates corresponds to lower variance and standard deviation of the stimulus estimate [498, Supplementary Materials]. Further examples of stimulus reconstructions for different population gains are shown in Figure 6.8.

Multisensory Integration The main advantage of the population coding scheme introduced in the last paragraph is that it gives way to an extremely simple method of computing the MLE from two stimulus estimates as described in Equations 6.6 and 6.7. Let r_{m_1} and r_{m_2} be the population codes for the estimates of a uniformly distributed stimulus s captured by modalities m_1 and m_2 . Further assume that both populations have the same size and have identical tuning curves. If these conditions hold, the multisensory population code r_{ms} that corresponds to the MLE can be simply computed as the sum of the two unimodal codes r_{m_1} and r_{m_2} [498]:

$$r_{ms} = r_{m_1} + r_{m_2} \quad (6.12)$$

This is because r_{ms} is also Poisson-distributed with gain $g_{ms} = g_{m_1} + g_{m_2}$. With the variance of the decoded stimulus distribution being inversely proportional to the gain, one can further conclude for σ_{ms}^2 :

$$g_{ms} = g_{m_1} + g_{m_2} = \rho \left(\frac{1}{\sigma_{m_1}^2} + \frac{1}{\sigma_{m_2}^2} \right) \stackrel{!}{=} \rho \frac{1}{\sigma_{ms}^2} \Rightarrow \sigma_{ms}^2 = \frac{1}{\frac{1}{\sigma_{m_1}^2} + \frac{1}{\sigma_{m_2}^2}} \quad (6.13)$$

The resulting value for σ_{ms}^2 corresponds exactly to that of the MLE in Equation 6.7. Note that the proportionality coefficient ρ is identical for g_{m_1} and g_{m_2} because the populations share identical tuning curves. For the same reason, the stimulus means encoded by r_{m_1} , r_{m_2} and consequently also by r_{ms} are also identical when neglecting the reconstruction error caused by the stochasticity of the population activity. Equation 6.6 for the mean of the integrated MLE is therefore trivially fulfilled, too. It can be shown that this method can be extended to more general settings. In particular, tuning curves do not need be identical for all input populations when the integrated estimate is not simply the sum but a linear combination of the population activities [498].

A Neurorobotics Experiment for Multisensory Integration

The PPC model for multisensory integration can be directly evaluated in a neurorobotics experiment in the NRP. Figure 6.6 shows a screenshot of the realized setup. It contains two robots that are placed inside the virtual laboratory. An *iCub* humanoid robot [503] is standing in the center of the free space with the head directed towards the other end of room, where a *Clearpath Husky* [504] mobile robot is driving back and forth along a straight line. The *iCub* acts as a passive observer of the Husky robot's current position. A view from one of its head cameras is shown in the right half of the simulation view. In the experiment, the Husky is assumed to be localized by both vision and sound.

Network Architecture The architecture of the network that fuses both estimates is depicted in Figure 6.7. Since the focus of this experiment is to evaluate the PPC model in a neurorobotics context, the activity of the two input populations is computed directly from the simulated ground truth position of the Husky robot. However, adding models for localization by vision and sound is in principle possible and does not require any modifications of the experimental setup or the network model. For example, the location stimulus can be provided by component networks that have been trained with topographic loss. The robot's position along the path is mapped to the interval $[0, 50]$, which is covered by 51 neurons in each of the three populations. As outlined in the figure, the neurons of the populations for vision and hearing are Poisson spike generators whose rates are set based on the tuning curve output and a user-defined gain factor. Both of them project to a third population with LIF neurons. Only units with identical preferred stimulus values are connected. The network was simulated with NEST [505] both inside and outside of the NRP experiment. Parameters were adjusted to ensure that the response of the population encoding the multisensory integration estimate corresponds to the sum of the input rates received from the two unimodal populations. All model parameters are documented in Appendix A.2.

Results Figure 6.8 summarizes the simulation results for a fixed input stimulus $s = 25$. The gain of the sound estimate was set to half the value of that for the vision estimate to reflect the higher accuracy of localizing the Husky robot visually along its trajectory. As one can clearly see, both gains add up in the response of the population encoding of the integrated estimate. In the bottom row, the stimulus probability is reconstructed for every population response. Distribution means and standard deviations were determined by fitting Gaussians to the resulting density functions. The retrieved values for σ_V , σ_S and σ_M are consistent with Equation 6.7. Due to the stochasticity of the encoding scheme, the computed means are slightly different from the original stimulus. However, the error of the vision estimate is significantly reduced after fusion with the sound estimate. This clearly highlights the relevance of mechanisms for robust multisensory integration in brain-derived systems.

The network was also evaluated directly in the NRP experiment, where the current ground truth stimulus was fed into the network in every time step of the simulation.

6.2 Neuromorphic Data Fusion

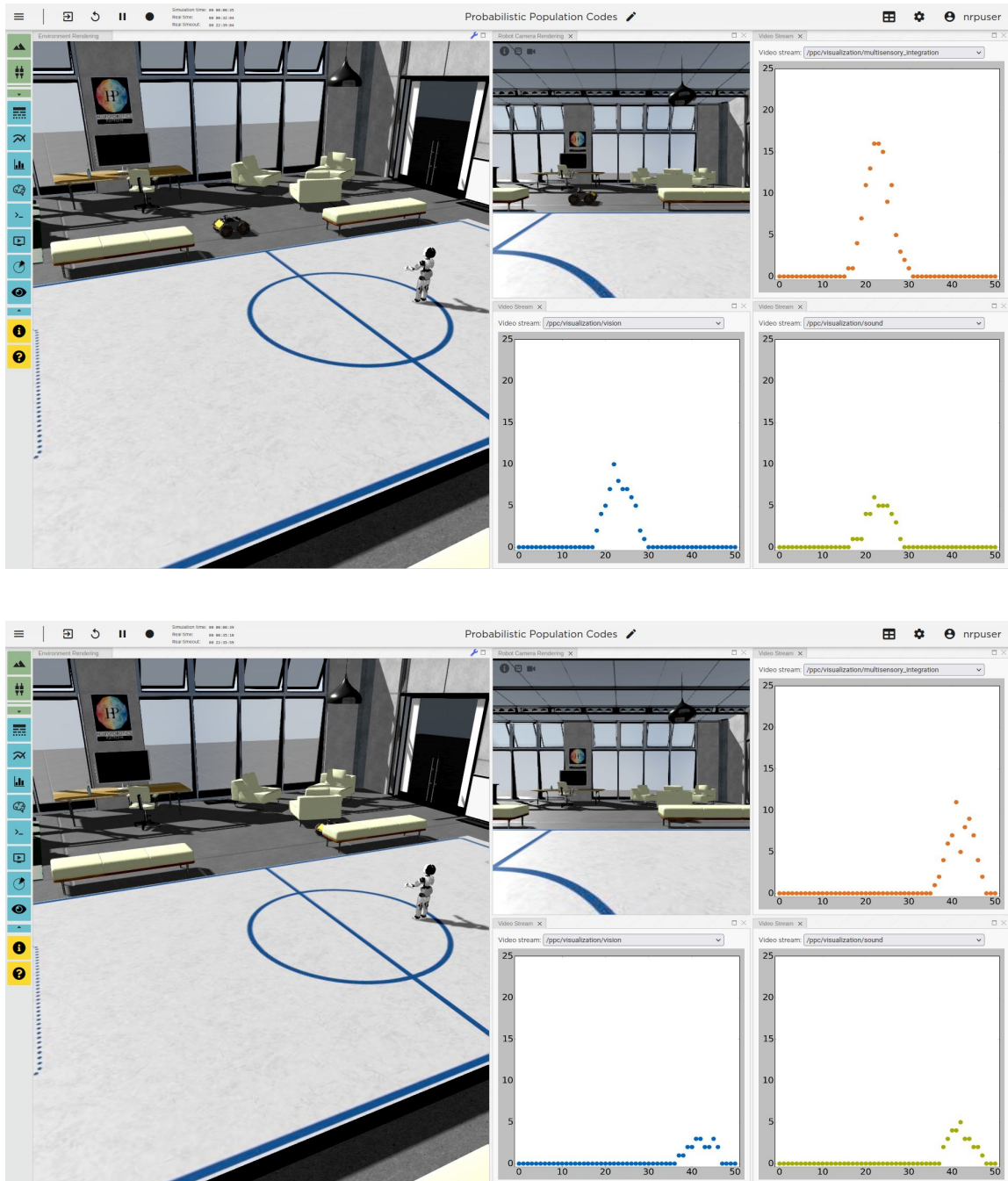


Figure 6.6: Simulation of the PPC model in the NRP. The experiment is comprised of an *iCub* robot [503] that “observes” a mobile *Clearpath Husky* robot [504] as it drives on a straight line back and forth through the room. The insets on the right side depict the camera view and the population codes for the multisensory position estimate (orange), the visual position estimate (blue) and the auditory estimate (green). *Top*: As long as the Husky robot is visible, the gain of the visual position estimate is higher than that of the auditory one. *Bottom*: When the robot drives behind one of the two benches, the visual gain decreases.

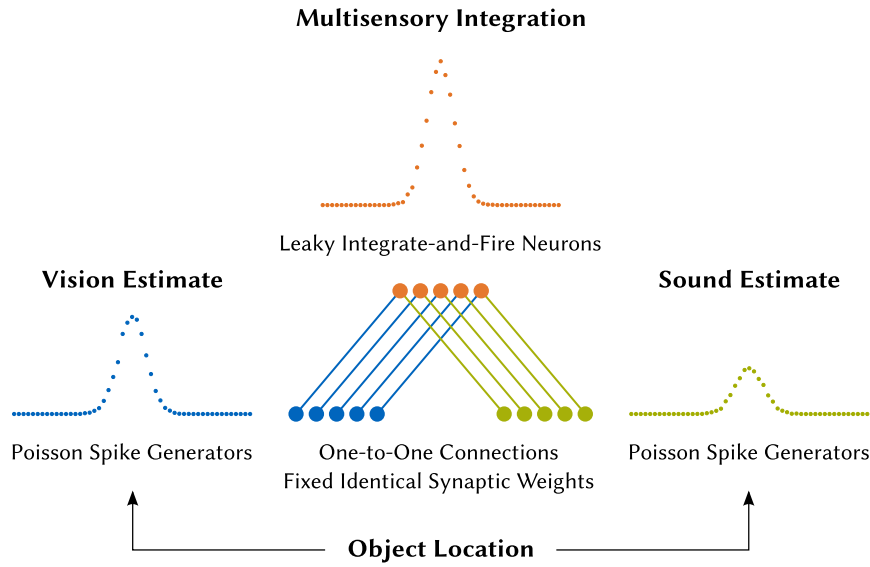


Figure 6.7: Neural network architecture for a virtual neurorobotics experiment on audio-visual localization with PPCs. The populations for vision and sound have identical tuning curves. All three populations contain 51 neurons each. The actual object location is directly retrieved from the NRP experiment. Adapted from [499].

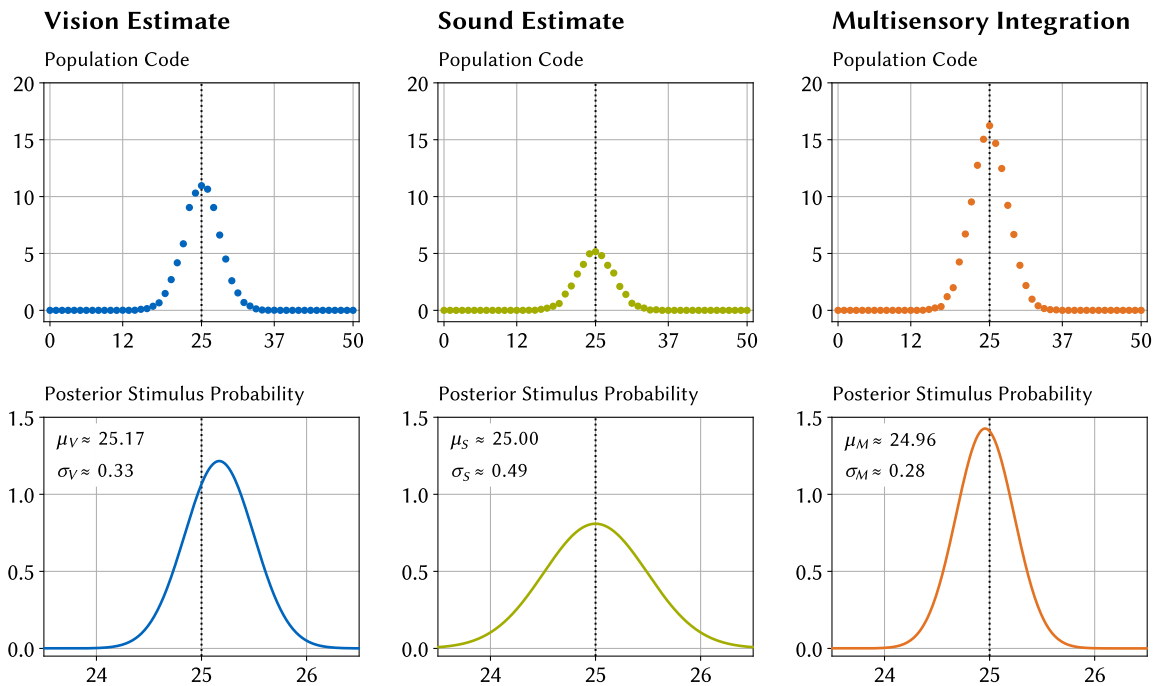


Figure 6.8: Simulation results for population codes and reconstructed stimulus estimates for vision, sound and multisensory integration. The ground truth input stimulus $s = 25$ provided to the network model from Figure 6.7 is indicated by the dashed line. Spike rates are averaged over 100 s to reduce the influence of stochastic fluctuations on the population activity and the stimulus reconstructions. Means and standard deviations were determined by fitting a Gaussian to the reconstructed probability density.

Additionally, as shown at the bottom of Figure 6.6, the gain of the vision stimulus decreased as soon as the Husky robot was occluded by one of the two benches. A special challenge of implementing the PPC model in neurorobotics is to retrieve a reliable readout of the population code within a simulation time step. In Figure 6.8, neuron firing rates were averaged over a duration of 100 s to reduce stochastic fluctuations. This is not possible in a neurorobotics experiment where the input constantly changes and the current estimate needs to be available at any point time with low latency. To address this issue, the simulation time step was increased to 50 ms and every position was represented by 100 neurons instead of only a single one. The latter modification makes it possible to get reliable estimates even within a relatively short time span. An alternative approach that requires fewer neurons but introduces additional latency beyond the simulation time step is to compute moving average of the population activities.

6.3 Staged Reinforcement Learning

PPCs fit well with the concept of training protocols because every sensory modality is represented by an own population of neurons. The population activities can be computed from raw sensory input data by component networks that have been trained individually based on a schedule. What is still missing yet is a method for training hub networks in stages during a schedule step, as shown in Figure 6.3. In this section, we develop a framework for implementing staged training in the NRP. In the first part, we introduce a formal definition of training stages and present a prototypical neurorobotics experiment to which it can be applied. It is based on a reaching task that requires learning the inverse kinematics of a robot arm with seven DoFs and supports increasing the task complexity over time. In the second part of this section, we use our novel framework to extend the *Deep Deterministic Policy Gradient (DDPG)* reinforcement learning algorithm with support for staged training. The resulting staged DDPG algorithm is then evaluated against its baseline version in the NRP on the described reaching task. The models and algorithms developed in this section are implemented based on an extended version of the training environment presented in [506].

Implementation of Training Protocols in the NRP

Developmental progression over several stages within a training protocol step can be modeled along three different but interdependent dimensions: the body, the brain and the environment. As the body becomes more capable (e.g. freeing of joints), the brain can learn more complex behaviors (e.g. coordinated movements) that in turn make it possible to solve more complex tasks in the environment (e.g. target reaching). Virtual neurorobotics is a key technology for the systematic investigation of these effects, because it allows the modeling of developmental processes without the constraints imposed by real-world robotics experiments. In particular, this makes it possible to investigate phenomena that cannot be replicated with physical robots, such as body growth. Studies on developmental

processes form a new class of neurorobotics experiments that is not based on a static description of the robot and its environment but instead on specification of how they evolve over time. In the following, we define both an experiment for the NRP with support for developmental progression and a control interface for modeling its subsequent stages.

Experiment Setup Figure 6.9 depicts an overview of the complete NRP experiment. It is based on a simplified version of the KUKA LBR iiwa robot model from Section 4.1 that is placed in the virtual laboratory. There is no tool attached and only joint space control is available. The goal of the experiment is to control the robot to reach a randomly sampled target position with the outermost link’s tip. In Section 4.1, the target is marked by the center of the green sphere and the required positioning accuracy is indicated by the sphere’s radius. In order for the robot to reach the target, it is effectively required to learn a controller that computes its inverse kinematics, which is the main purpose of the experiment. All observations from the environment, including the target position, are directly provided as numerical values. The setup can be easily extended for learning from visual input by adding cameras.

A key feature of the experimental setup is that it supports variation along two main dimensions. First, the task can be changed by randomizing the position of the sphere within the robot’s workspace. A visualization of the sampling space is depicted in Figure 6.10. At the same time, the positioning accuracy for a reaching movement to be considered

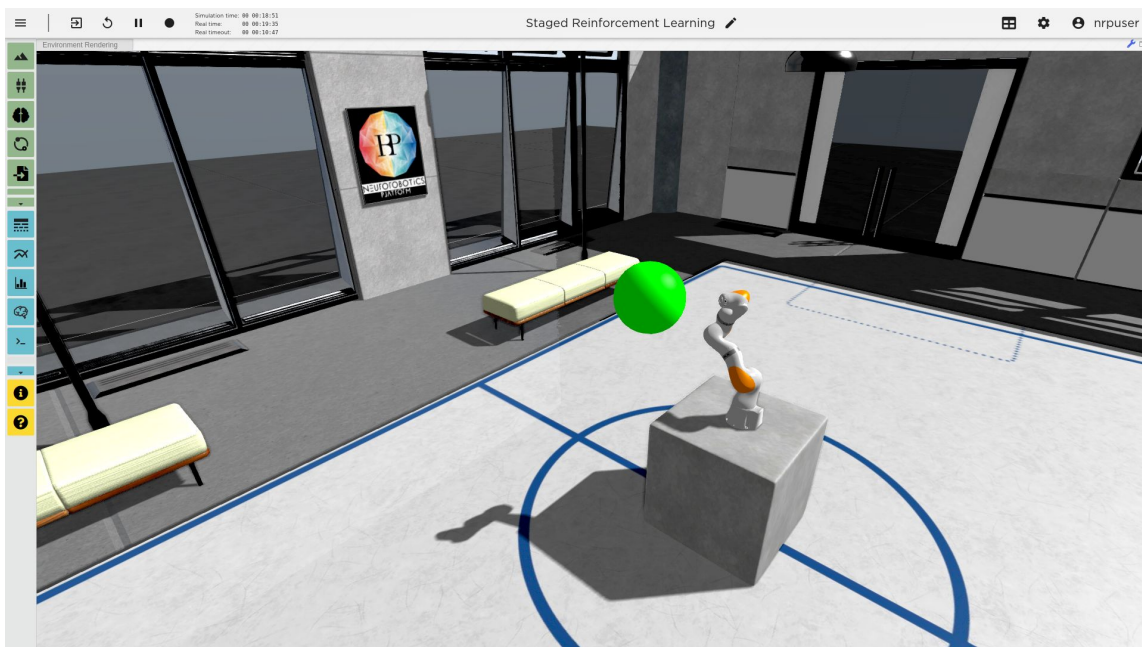


Figure 6.9: NRP experiment for staged reinforcement learning with training protocols. The setup is based on the KUKA LBR iiwa robot model introduced in Section 4.1, but only contains a joint space controller. The green sphere can be generated dynamically and marks a target position in the workspace. To reach this position with the tip of the last joint, the robot needs to learn an inverse kinematics model.

successful can be freely adjusted as indicated by the different sphere sizes in the example from Figure 6.11. This makes it possible to gradually increase the complexity of the task by decreasing the target sphere volume over subsequent training stages. The second main dimension of variation lies in the robot’s kinematic configuration. Its seven DoFs not only make control complex despite the simplicity of the task, but are also redundant and thereby enable the investigation of the effect of freezing and freeing individual joints according to the theory of Bernstein [492].

Definition of Training Stages In the scope of this work, we will investigate developmental progression along the two lines of variation implemented by the experimental setup described above. This requires a specification of how the set of active robot joints and the required positioning accuracy evolve throughout the learning process. In its most general form, the progression of parameters can be modeled as follows:

$$\text{Stage}(d) = (\mathbf{F}_d, o_d, \mathbf{H}_d) \in \mathbb{R}^{7 \times 7} \times \mathbb{R}^7 \times \mathbf{R}^{M \times N} \quad (6.14)$$

The function $\text{Stage}(\cdot)$ maps every development stage d to a linear transformation from joint control commands $c_J \in \mathbb{R}^7$ to a constrained control space:

$$c_J \mapsto \mathbf{F}_d c_J + o_d \quad (6.15)$$

Additional hyperparameters for the training in the current stage, such as the required positioning accuracy, can be provided in the optional matrix \mathbf{H}_d . The encoding of joint constraints in a linear mapping enables not only the freezing and freeing of joints but also makes it possible to induce fixed correlations between their movements. This is in line with experimental studies, where motion data from individual joints is often analyzed in terms of its cross-correlations [493]. In a sense, a frozen joint is not necessarily fixed to a constant angle but rather tightly correlated to the motions of other joints. From a robotics point of view, Equation 6.15 can be interpreted as the definition of an *underactuated* system for an appropriate choice of the constraint matrix \mathbf{F}_d . When $\text{Stage}(\cdot)$ is modeled with a function approximator such as a neural network, it can, for example, be trained in an outer optimization loop to automatically determine appropriate training protocols for different types of tasks. This can be interpreted as a form of *meta-learning* [507].

Staged DDPG Algorithm

Both the neurorobotics experiment from Figure 6.9 and the definition of developmental stages from Equation 6.14 are completely agnostic of the actual training procedure. The proposed developmental model is therefore not a new learning algorithm but a method for guiding and shaping the training process. Candidate solutions for the reaching task in the experiment can be easily evaluated by checking if the robot’s tip is located in the target sphere. However, without any prior knowledge, it is not obvious how to produce good candidate solutions in the first place. Following the schema from Figure 5.10, this problem setup lends itself well to a reinforcement learning approach. In particular, we

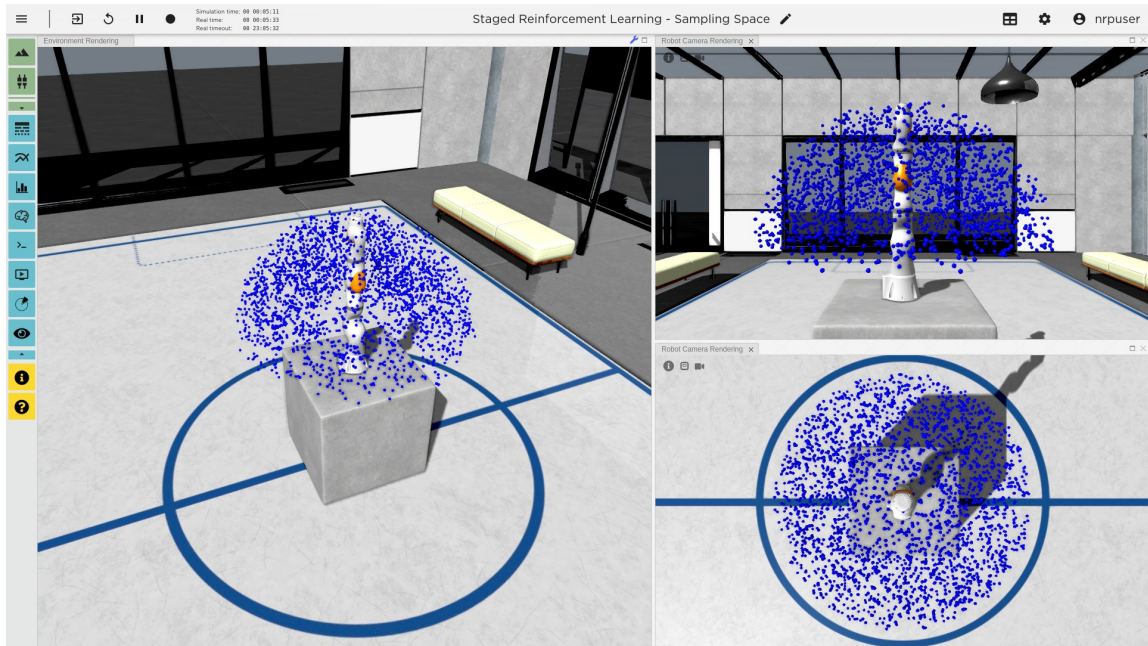


Figure 6.10: Visualization of the sampling space for target points. Samples are drawn from a hemisphere that is centered around the robot arm. Unreachable positions close the robot's base are left out. The depicted samples are part of a data set that is used for evaluating the robot's performance.

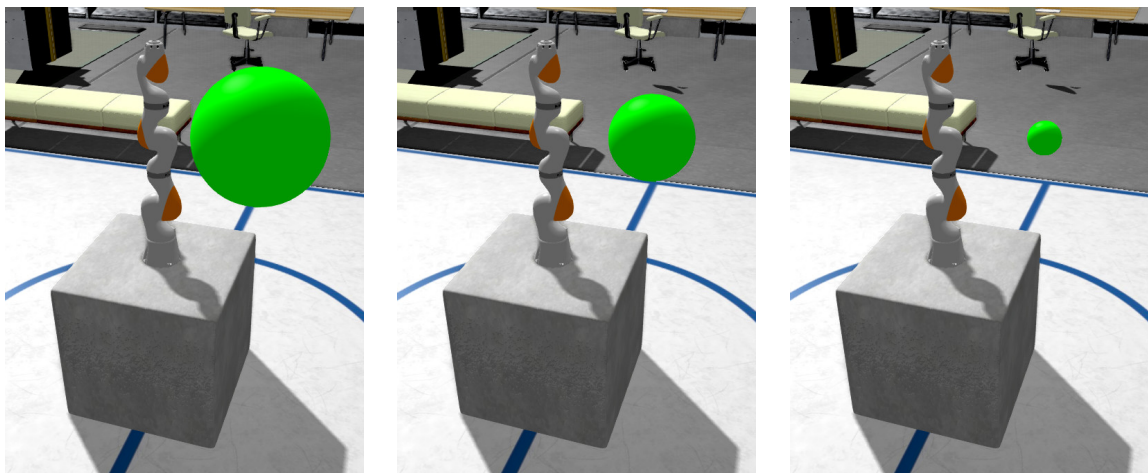


Figure 6.11: Increasing accuracy requirements in the target reaching experiment from Figure 6.9 over subsequent training stages. The three spheres are centered at the same target position. Increased positioning accuracy requirements are indicated by decreasing sphere volumes.

will investigate how the DDPG algorithm for deep reinforcement learning algorithm can be extended to support the stages of training protocols. The application of reinforcement learning to solve a task is also fully in line with MHNNs, where the hubs perform task-based sensor fusion. As the reward signal in reinforcement learning is directly related to task performance, the hubs automatically learn to integrate and fuse their component network input to accomplish the assigned task.

Related Work The concept of training protocol stages developed in this section is similar to what is called *curriculum learning* in traditional machine learning [508]. Similar to stages, a curriculum imposes a temporal order on the training process by setting constraints on the data processed that evolve as learning progresses. In a general non-embodied setting, however, the definition of a curriculum is less obvious than for a concrete physical task. Possible choices include an increase of the amount of noise in the data or the growth of the vocabulary in a language processing task with the goal of faster training and better performing models [508]. Defining curricula is much easier in reinforcement learning applications, where the task to be solved often can be naturally made easier by simplifying the goal or by introducing subtasks such as proposed in a study by Riedmiller et al. [509]. In competitive multi-agent settings, curricula can also emerge automatically as opposing agents create increasingly complex tasks for their opponents. Such *autocurricula* have, for example, been investigated by Baker et al. [510], who reported that the trained agents automatically learned to use tools during an emergent sequence of phases. Closely related to the method developed in this section is the pioneering work by Lungarella and Berthouze [511, 512]. They investigated the gradual freeing of DoFs in a humanoid robot, where the knee and hip joints were controlled by neural oscillators. The system was freely suspended with the goal of producing swinging motions through leg movements. Freeing the knee joints after the stabilization of the hip joints resulted in robust convergence to stable motor patterns, which was not the case when both joints were unlocked from the beginning on. When the experimental setup was later extended to include non-linear perturbations from the environment, freeing the second joint neither produced consistent results nor led to improved performance. However, it was observed that alternating freeing and freezing of the joint could re-stabilize the system [511, 513]. In an experiment on developmental learning of robot grasping by Savastano and Nolfi [514], an incremental training process was defined in terms of basic reflexes and constraints in the neural system. The achieved performance was superior to a non-developmental baseline and the system turned out to freeze the outer joints of the robot arm at the beginning before they were freed in later phases. Stulp and Oudeyer [515] applied a direct reinforcement learning algorithm to learn a reaching task. Based on their findings on the amount of exploratory movements of each joint, they concluded that the algorithm automatically froze outer degrees of freedom at the beginning of the learning process. However, it should be noted reaching motions were only trained for two fixed goals. A formal framework for developmental learning was proposed by Lee et al. [516]. It is called “*Lift-Constraint, Act, Saturate*” and describes the subsequent lifting of constraints

as experiences gathered by acting under the current constraint set saturate. Differently from the method developed in this work, the framework is not defined formally but rather outlines a general strategy for developmental learning. Moreover, to our knowledge, we also for the first time combine developmental learning based on the freezing and freeing of joints with deep reinforcement learning.

Reinforcement Learning As a preparation for the development of the staged DDPG algorithm, this paragraph briefly introduces the theoretical foundations of reinforcement learning based on the work by Sutton and Barto [517]. At the core of all reinforcement learning problems is the modeling of the environment with Markov decision processes (MDPs) as illustrated in Figure 6.12. As one can easily see, the overall structure is very similar to that of an embodied system which is coupled to its environment through a closed PCA loop. The *agent*, however, does not need to be embodied and the interaction with the environment is abstracted as a sequence of states $s_t \in S$, actions $a_t \in A$ and rewards $r_t \in R \subset \mathbb{R}$. In this introduction, we will only consider finite discrete-time MDPs with $t \in \mathbb{N}$ and a finite number of states and actions. As rewards are deterministic and depend on both actions and states, R is finite, too. The environment dynamics are modeled by a state transition probability function $p(s', r | s, a)$ [517, p. 48]:

$$p(s', r | s, a) = p(s_{t+1} = s', r_{t+1} = r | s_t = s, a_t = a) \quad \text{for } s, s' \in S, a \in A \text{ and } r \in R \quad (6.16)$$

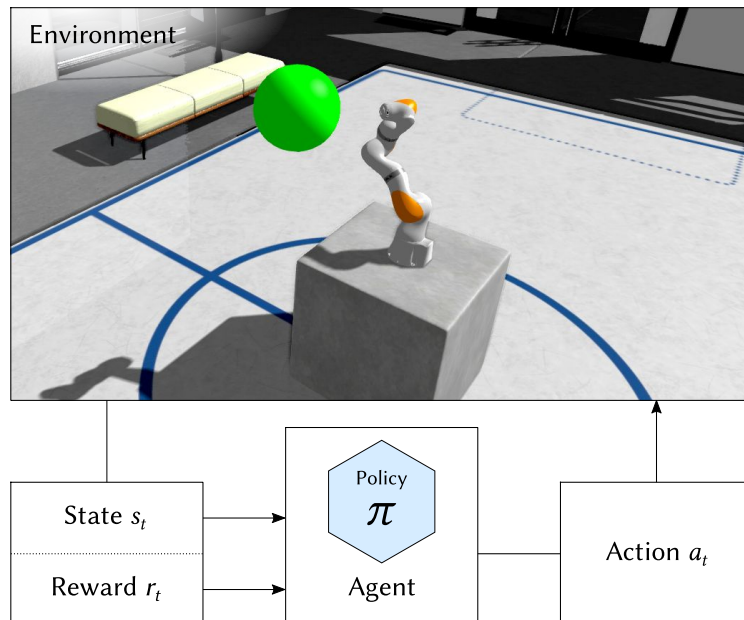


Figure 6.12: Schematic illustration of the PCA-loop in an MDP for reinforcement learning. The individual components of the system can be directly mapped to those from Figure 3.2. In particular, the cognitive model M is instantiated by a policy π that is executed by a reinforcement learning agent. The perception is comprised of a state description s_t of the environment and a reward r_t for the last action a_{t-1} . The figure is based on a schema by Sutton and Barto [517, p. 48].

While the set of possible actions in a time step t may depend on the state s_t , we will assume that all actions are eligible in every state. This makes sense in a robot control task where any command can be issued to the robot independently of the robot's configuration. Commands that cannot be executed (e.g. due to joint limits) are not excluded from the action set but simply result in the same state. In summary, an MDP can be specified as a tuple (S, A, R, p) of states, actions, rewards and a state transition probability function.

The interactions of the agent with its environment result in a sequence of states, actions and rewards. At time step t , the future reward for a specific trajectory in the state space is defined as the *return* g_t [517, p. 55]:

$$g_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma g_{t+1} \quad (6.17)$$

In the equation above, γ is a *discount rate* that determines the weight given to future rewards. The actual value of g_t not only depends on the states and rewards of the environment but in particular also on the *policy* π that an agent employs to select its next action a in state s :

$$\pi(a | s) = p(a_t = a | s_t = s) \quad (6.18)$$

The *expected return* resulting from following policy π in state $s \in S$ and time step t is formalized as the *state value function* $v_\pi(s)$ [517, p. 58]:

$$v_\pi(s) = \mathbb{E}_\pi [g_t | s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] \quad (6.19)$$

Analogously, the *action value function* $q_\pi(s, a)$ computes the expected return resulting from following policy π after taking action $a \in A$ at state $s \in S$ in time step t [517, p. 58]:

$$q_\pi(s, a) = \mathbb{E}_\pi [g_t | s_t = s, a_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right] \quad (6.20)$$

Both the state value function and the action value function can be computed recursively as defined by the *Bellman equations* [517, p. 59]:

$$v_\pi(s) = \sum_{a \in A} \pi(a | s) \sum_{s' \in S, r \in R} p(s', r | s, a) (r + \gamma v_\pi(s')) \quad (6.21)$$

$$q_\pi(s, a) = \sum_{s' \in S, r \in R} p(s', r | s, a) \left(r + \gamma \sum_{a' \in A} \pi(a' | s') q_\pi(s', a') \right) \quad (6.22)$$

The main goal of reinforcement learning is to determine an optimal policy π_* that yields the maximum attainable expected return [517, pp. 62 – 63]:

$$v_*(s) = \max_{\pi} v_\pi(s) \quad (6.23)$$

$$q_*(s, a) = \max_{\pi} q_\pi(s, a) \quad (6.24)$$

v_* and q_* are the optimal state and action value functions of the MDP. Based on $q_*(s, a)$, a candidate solution for π_* can be directly derived by taking only those actions that maximize $q_*(s, a)$ in every state s and every time step t . The challenge of reinforcement learning problems lies in the fact that both the model parameters of the MDP and the optimal value function $q_*(s, a)$ are typically unknown. They therefore need to be approximated based on experiences $(s_t, a_t, r_{t+1}, s_{t+1})$ gained from interaction with the environment. In the *Q-learning* algorithm, this is achieved by applying the following update rule to compute estimates for $q(s_t, a_t)$ with update step size α [517, p. 131]:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a \in A} q(s_{t+1}, a) - q(s_t, a_t) \right) \quad (6.25)$$

The update rule is completely independent from the policy that generates the experiences $(s_t, a_t, r_{t+1}, s_{t+1})$, which is why Q-learning is called an off-policy algorithm. It has been shown to converge to the optimal action value function as long as the policy provides sufficient coverage of the state space [517, p. 131]. Therefore, it is important that the agent not only *exploits* the knowledge it has already gained, but also *explores* the environment at the cost of possibly less immediate reward.

DDPG Algorithm The optimal value function $q_*(s, a)$ in the formal reinforcement learning framework introduced so far is only defined for discrete state and action spaces. Essentially, it corresponds to a table that maps states and actions to their corresponding optimal action values. Moreover, if the action space A is continuous, choosing an action that maximizes $q_*(s, a)$ would entail running a computationally expensive optimization procedure in every time step and is therefore impractical [471]. Policy gradient methods offer a solution to this issue by training a parameterized policy $\pi(a | s, \theta_\pi)$ directly with gradient-based optimization instead of deriving it from $q_*(s, a)$. The performance of the policy $\pi(a | s, \theta_\pi)$ (the actor) can then be assessed by a value function approximator $q(s, a | \theta_q)$ (the critic) that is trained in parallel and supports a continuous action space [517, p. 331]. In the DDPG algorithm by Lillicrap et al. [471], both the actor and the critic are modeled as DNNs N_π and N_q , which makes the method directly applicable to the training of hubs in MHNNs. As the actor is no longer a probabilistic policy but instead implements a direct mapping from states to actions, it will be denoted as a function $\mu(s | N_\mu)$.

Both the actor $\mu(s | N_\mu)$ and the critic $q(s, a | N_q)$ can be trained with standard gradient-based optimization algorithms. As μ is deterministic, experience needs to be collected off-policy with respect to μ from a stochastic policy β to ensure sufficient exploration, which yields a discounted state visitation distribution ρ^β [518]. Assuming that μ always chooses actions that maximize $q(s, a | N_q)$, the loss function for the critic can now be defined as follows [471]:

$$\mathcal{L}_q(N_q, \gamma_Q(\cdot | N_q, N_\mu)) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_{t+1} \sim p(r|s_t, a_t)} \left[\left(q(s_t, a_t | N_q) - \gamma_Q(r_{t+1}, s_{t+1} | N_q, N_\mu) \right)^2 \right] \quad (6.26)$$

Its second argument is the function stated below:

$$\gamma_Q(r_{t+1}, s_{t+1} \mid N_q, N_\mu) = r_{t+1} + \gamma q(s_{t+1}, \mu(s_{t+1} \mid N_\mu) \mid N_q) \quad (6.27)$$

Note that the loss function in Equation 6.26 has the same form as the update term of the original Q-learning rule in Equation 6.25. Minimizing the loss therefore corresponds to vanishing updates in Equation 6.25 and thus convergence to the optimal action value function. The actor is not trained on a loss but on a *performance objective* $J_\beta(N_\mu)$ [471, 518]:

$$\begin{aligned} J_\beta(N_\mu) &= \int_S \rho^\beta(s_t) q(s_t, \mu(s_t \mid N_\mu) \mid N_q) ds_t \\ &= \mathbb{E}_{s_t \sim \rho^\beta} \left[q(s, a \mid N_q) \Big|_{s=s_t, a=\mu(s_t \mid N_\mu)} \right] \end{aligned} \quad (6.28)$$

Based on $J_\beta(N_\mu)$, the deterministic policy gradient can now be computed as follows [471]:

$$\begin{aligned} \nabla_{N_\mu} J_\beta(N_\mu) &\approx \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_{N_\mu} q(s, a \mid N_q) \Big|_{s=s_t, a=\mu(s_t \mid N_\mu)} \right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_a q(s, a \mid N_q) \Big|_{s=s_t, a=\mu(s_t \mid N_\mu)} \nabla_{N_\mu} \mu(s \mid N_\mu) \Big|_{s=s_t} \right] \end{aligned} \quad (6.29)$$

Equations 6.26 and 6.29 form the basis of the DDPG algorithm.

Staged DDPG Algorithm We have extended the original version of the DDPG algorithm by Lillicrap et al. [471] with support for training protocol stages in Algorithm 6. An earlier version of this algorithm was presented in [506]. The developmental progression defined in the training protocol is reflected by an additional outer loop that iterates over all stages of the scheduling step. At the beginning of each development stage d , the current parameters are retrieved from the stage definition function $\text{Stage}(\cdot)$ introduced in Equation 6.14. Within every stage, the default training loop of the DDPG algorithm is executed for the number of episodes and time steps defined by the stage's hyperparameters \mathbf{H}_d . The target networks \tilde{N}_q and \tilde{N}_μ store delayed copies of N_q and N_μ , respectively, to stabilize the learning process. In addition, experiences collected during the interaction with the environment are added to a replay buffer \mathbf{B}_d . Depending on the training protocol, it is retained or cleared at the beginning of a new stage. By learning from batches that are drawn randomly from this buffer, correlations between subsequent experiences can be resolved, which is essential for stochastic-gradient-based optimization. As suggested by Fujimoto et al. [519], the action noise required for exploratory behavior is drawn from a normal distribution instead of an Ornstein-Uhlenbeck process. The most important difference from the original DDPG algorithm is the selection of the next action a_t in Line 8. Rather than directly executing the action returned by the policy, the joint constraints imposed by the current stage are applied according to Equation 6.15. As a result, Algorithm 6 implements the stages defined by the training protocol.

Algorithm 6: Staged DDPG Algorithm**Input:**

- E : An environment defined as a Markov decision process (S, A, R, p)
- N_μ : Actor network with randomly initialized weights
- N_q : Critic network with randomly initialized weights

Parameters:

- D : Number of developmental stages in the training protocol
- $\text{Stage}(\cdot)$: Training stage function as defined in Equation 6.14
- b : Batch size
- τ : Target network update rate
- Γ : Optimization parameters

Set up replay buffer and initialize target networks

1 $\mathbf{B}_0 \leftarrow \emptyset; \tilde{N}_\mu \leftarrow N_\mu; \tilde{N}_q \leftarrow N_q$

Iterate over D stages of developmental progression

2 **foreach** $d \in [1 \dots D]$ **do**

Retrieve parameters for the current stage d and initialize the replay buffer

3 $\mathbf{F}_d, o_d, \mathbf{H}_d \leftarrow \text{Stage}(d)$

4 $\mathbf{B}_d \leftarrow \text{InitializeBuffer}(\mathbf{H}_d[\text{retain_replay_buffer}], \mathbf{B}_{d-1})$

5 **foreach** $e \in [1 \dots \mathbf{H}_d[\text{epochs}]]$ **do**

6 $s_1 \leftarrow$ Initial state of E with respect to \mathbf{H}_d

7 **foreach** $t \in [1 \dots \mathbf{H}_d[\text{time_steps}]]$ **do**

Execute an action based on the constrained policy and exploration noise

8 $a_t \leftarrow \mathbf{F}_d(\mu(s_t | N_\mu) + \mathcal{N}(0, \mathbf{H}_d[\sigma_{exp}]^2)) + o_d$

9 $s_{t+1}, r_{t+1} \leftarrow$ Execute action a_t in $E(s_t, \mathbf{H}_d[E])$

Add the retrieved observation to the replay buffer

10 $\mathbf{B}_d \leftarrow \text{Append}(\mathbf{B}_d, (s_t, a_t, r_{t+1}, s_{t+1}))$

Sample a random batch of size b from the replay buffer

11 $\langle (s_i, a_i, r_{i+1}, s_{i+1}) \rangle \leftarrow \text{GetSamples}(\mathbf{B}_d, b)$

Update the critic network N_q

12 $g_q \leftarrow \nabla_{N_q} \mathcal{L}_q(N_q \langle (s_i, a_i, r_{i+1}, s_{i+1}) \rangle, y_Q(\cdot | \tilde{N}_q, \tilde{N}_\mu) \langle (s_i, a_i, r_{i+1}, s_{i+1}) \rangle)$

13 $N_q \leftarrow \text{Optimizer}(N_q, \Gamma, g_q)$

Update the actor network N_μ

14 $g_\mu \leftarrow \nabla_{N_\mu} J_\beta(N_\mu \langle (s_i, a_i, r_{i+1}, s_{i+1}) \rangle)$

15 $N_\mu \leftarrow \text{Optimizer}(N_\mu, \Gamma, g_\mu)$

Perform soft target network updates

16 $\tilde{N}_\mu \leftarrow (1 - \tau)\tilde{N}_\mu + \tau N_\mu$

17 $\tilde{N}_q \leftarrow (1 - \tau)\tilde{N}_q + \tau N_q$

18 **end**

19 **end**

20 **end**

Experimental Evaluation

Algorithm 6 and the NRP experiment from Figure 6.9 provide all components required for training a hub network in stages according to a training protocol. In the next paragraph, we instantiate them with concrete models and parameters. Based on the target reaching task defined by the experiment, we then investigate two different types of stages. We first consider a training protocol in which the difficulty of the reaching task is gradually increased by decreasing the radius of the target as learning progresses. In the second type of protocol, the target radius remains constant, but the robot's DoFs are gradually freed. Finally, in the last paragraph, we provide a summary of our main results.

Reinforcement Learning Setup As usual in reinforcement learning, the training process is organized in *episodes*. Every episode is comprised of a number of steps, each of which corresponds to a single iteration of the PCA loop from Figure 6.12. At the beginning of an episode, the robot moves to a random start configuration and a random target position p_{target} is drawn from the sampling space $\mathbb{P}_{target} \subset \mathbb{R}^3$ depicted in Figure 6.10. The sampling procedure ensures that the generated configurations and target positions comply to the constraints imposed by the currently active training stage. An episode ends after at most 20 steps or when the robot reaches the target with an accuracy defined by the radius r_{target} . Based on the robot's joint limits from Equation 5.19, the experiment's state space S and action space A can be defined as follows:

$$S : \mathbb{C}_{KUKA} \times \mathbb{P}_{target} \quad (6.30)$$

$$A : (-1, 1)^7 \quad (6.31)$$

Analogously, the signatures of the actor and the critic are specified as stated below:

$$\mu(s \mid N_\mu) : \mathbb{C}_{KUKA} \times \mathbb{P}_{target} \rightarrow (-1, 1)^7 \quad (6.32)$$

$$q(s, a \mid N_q) : \mathbb{C}_{KUKA} \times \mathbb{P}_{target} \times (-1, 1)^7 \rightarrow \mathbb{R} \quad (6.33)$$

The state space S not only includes the current configuration of the robot but also the position of the target. The actions are relative motion commands in the robot's joint space. Before they can be executed, they must be mapped from the input action space A to the robot's actual action space \mathbb{C}_{KUKA} . To retrieve the robot's new target configuration, the resulting motion command is added to its current configuration. In the last step, invalid target joint angles are clipped with respect to \mathbb{C}_{KUKA} . The reward for an action is computed based on the negative workspace distance between the position p_{tool} of the robot's tool tip center the target position p_{target} . To encourage the actor to reach the target in as few steps as possible, every step further adds a fixed negative reward of -1 . This is motivated by the fact that oscillatory movements close to the target result in many additional steps but add only little negative distance-based reward. The complete reward function is defined as follows:

$$r(p_{tool}) = \begin{cases} -\|p_{target} - p_{tool}\| - 1 & \text{if } \|p_{target} - p_{tool}\| > r_{target} \\ 15 & \text{if } \|p_{target} - p_{tool}\| \leq r_{target} \end{cases} \quad (6.34)$$

Network Architecture and Training The architectures of the actor network N_μ and the critic network N_q are shown in Figure 6.13. Both networks are comprised of two fully connected hidden layers with $\text{ReLU}(\cdot)$ activation functions. To ensure that the actor only generates valid actions $a \in A$, its output units have $\text{tanh}(\cdot)$ activation functions. The output unit of the critic has a linear activation function, as the reward function from Equation 6.34 can yield both positive and negative value function predictions. Both networks were implemented in TensorFlow [135] and Keras [520] using the reinforcement learning framework *keras-rl* [521], which supports OpenAI Gym-compatible environments and provides an implementation of the DDPG algorithm. The extension of the framework required for training protocols as well as an OpenAI Gym interface for the NRP were presented in [506]. The interface also contains a forward kinematics model of the robot for training without the NRP. This enables considerably faster training compared to the full-featured NRP experiment, which also simulates the robot’s dynamics and the complete environment. The forward kinematics are computed with the tool *ikpy* [522] based on the robot’s original URDF model that is also used in the NRP experiment. As the experimental setup considered in this section only requires the simulation of the robot’s

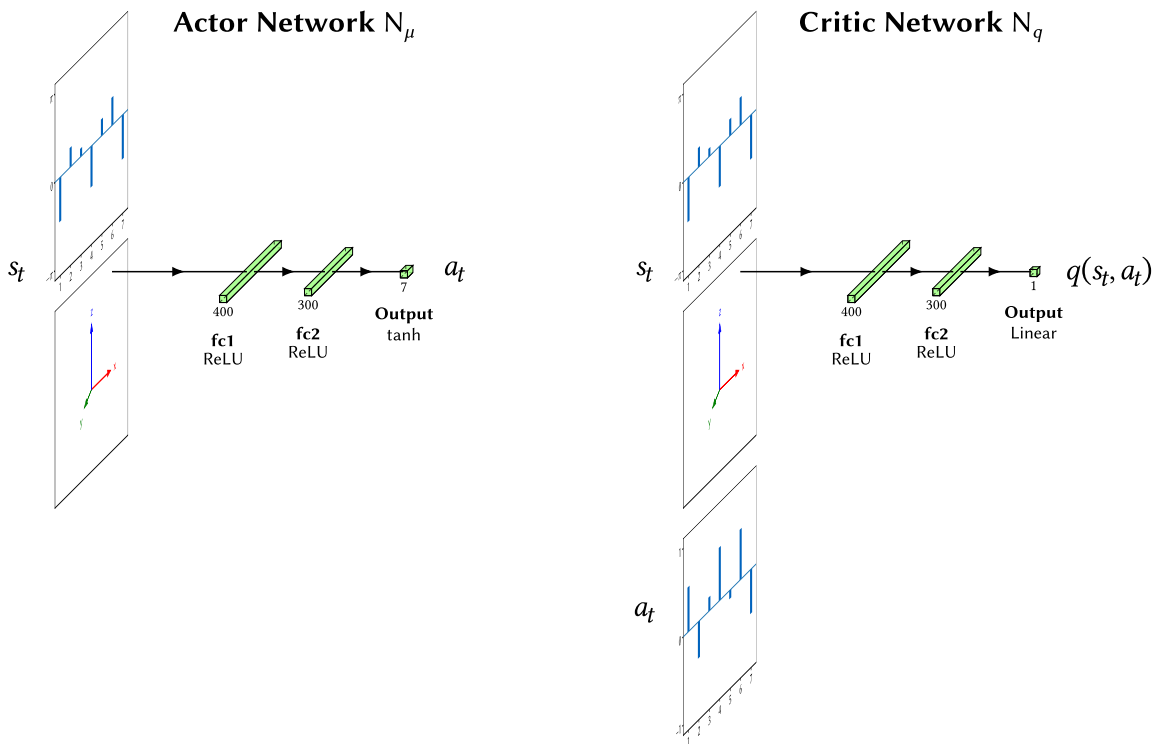


Figure 6.13: Architectures of the actor network N_μ and the critic network N_q . Both networks are comprised of two fully connected hidden layers with $\text{ReLU}(\cdot)$ activation functions. The output of the actor is limited to the robot’s action space A by a $\text{tanh}(\cdot)$ activation function. Note that the generated actions also need to be post-processed by adding exploration noise, transforming them into the robot’s physical action space \mathcal{C}_{KUKA} , and applying the constraint mapping of the currently active stage.

forward kinematics, all results presented in the next paragraphs were obtained with the *ikpy*-based robot model to speed up training. Apart from small numerical deviations, tests of the trained models in the NRP yielded identical results, proving the validity of this approach. Of course, more complex experiments involving visual input, robot dynamics and interaction with the environment require training in the NRP. The hyperparameters of the networks and the training algorithm were adjusted based on proven values for similar types of tasks available as part of *Stable Baselines3*, a collection of implementations of reinforcement learning algorithms [523, 524]. Synaptic weight updates were computed using the Adam optimizer [467]. In order to obtain unbiased results, all experiments described in the following were executed 100 times each with different random seeds. All hyperparameters used for the training of the models presented in this section are documented in Appendix A.3.

Results of Training with Environment-Based Stages In a first experiment, we evaluated Algorithm 6 on a training protocol with environment-based stages. It is listed in Table 6.1. All joints of the robot are unlocked throughout all stages, except for the outermost one. This is because there is no tool attached and therefore the angle of the outermost joint does not affect the workspace position of the robot’s tip. The main parameter of the protocol is the target radius r_{target} , which decreases over time. As a result, the required positioning accuracy increases and the complexity of the reaching task grows. Correspondingly, the standard deviation σ_{exp} of the exploration noise decreases in each stage, and the last two stages are considerably longer in terms of the number of steps. Before the actual training of the actor and critic networks starts, there is a *warm-up phase* with $\mathbf{H}_d[\text{warm-up}]$ steps, during the which the system collects experience from the environment to fill the replay buffer of Algorithm 6. The results of 100 training runs with this protocol are summarized in Figure 6.14. The left column shows the baseline performance resulting from training without the progression of task complexity. Compared to the protocol from Table 6.1, both the target radius r_{target} and the standard deviation σ_{exp} of the

Table 6.1: A training protocol with environment-based stages. $\text{Diag}(\cdot)$ defines a diagonal matrix, with the first entry in the argument corresponding to the robot’s base joint. $\mathbf{H}_d[r_{target}]$ specifies the target radius for the reward function from Equation 6.34 and $\mathbf{H}_d[\sigma_{exp}]$ the standard deviation for the exploration noise applied in Line 8 of Algorithm 6. $\mathbf{H}_d[\text{steps}]$ and $\mathbf{H}_d[\text{warm-up}]$ determine the total number of PCA steps performed in the corresponding stage and the length of the warm-up period, respectively.

Stage d	\mathbf{F}_d	o_d	$\mathbf{H}_d[r_{target}]$	$\mathbf{H}_d[\sigma_{exp}]$	$\mathbf{H}_d[\text{steps}]$	$\mathbf{H}_d[\text{warm-up}]$
1	$\text{Diag}(1, 1, 1, 1, 1, 1, 0)$	0	0.40	0.070	40 000	1000
2	$\text{Diag}(1, 1, 1, 1, 1, 1, 0)$	0	0.30	0.050	30 000	500
3	$\text{Diag}(1, 1, 1, 1, 1, 1, 0)$	0	0.25	0.042	30 000	500
4	$\text{Diag}(1, 1, 1, 1, 1, 1, 0)$	0	0.20	0.035	30 000	500
5	$\text{Diag}(1, 1, 1, 1, 1, 1, 0)$	0	0.15	0.025	150 000	500
6	$\text{Diag}(1, 1, 1, 1, 1, 1, 0)$	0	0.10	0.017	200 000	500

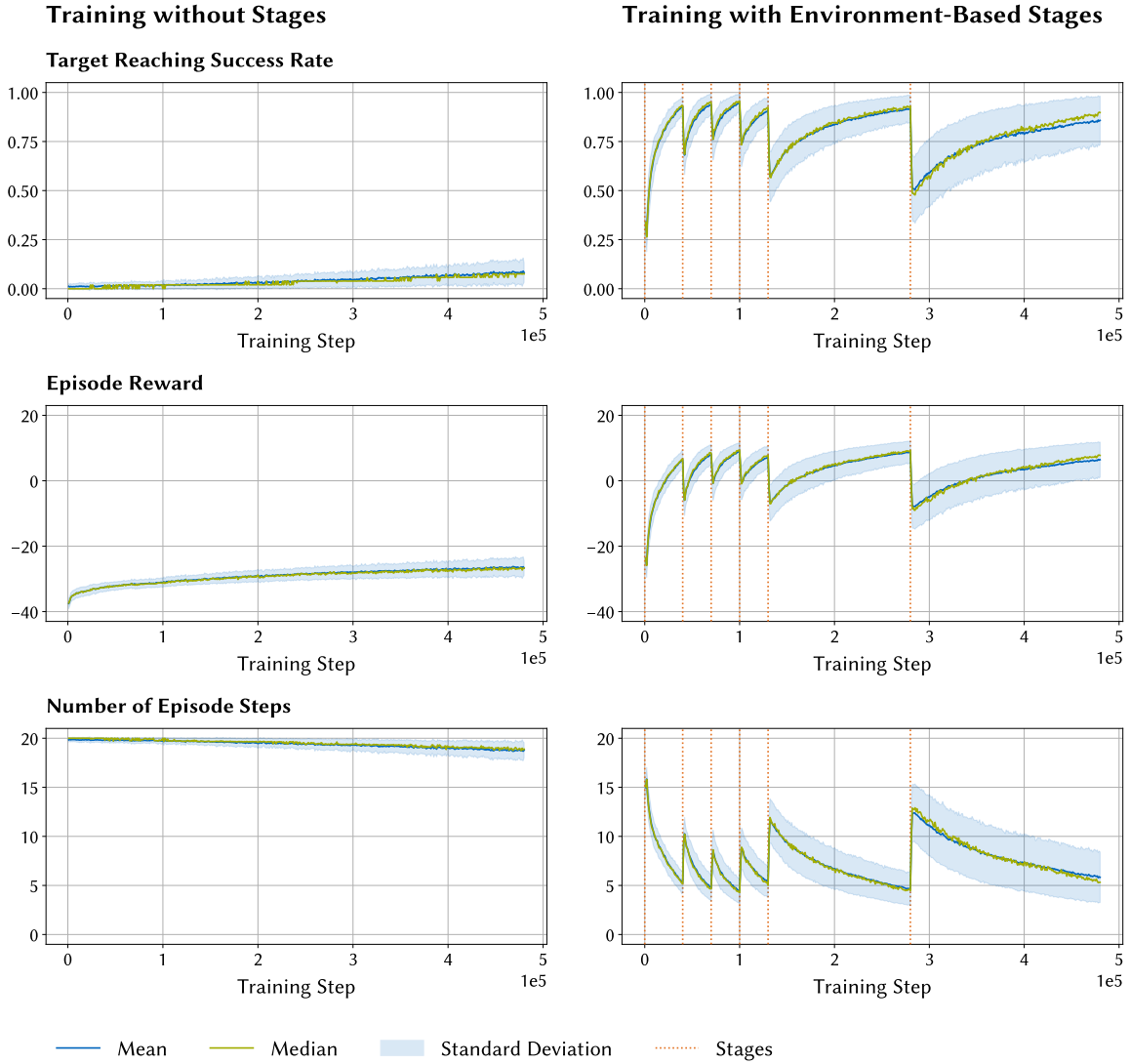


Figure 6.14: Results from 100 training runs of Algorithm 6 with environment-based stages. The data points from each run are averages from 1000 PCA cycles. *Left:* Training with fixed target radius $r_{target} = 0.1$ and exploration noise as described in Table 6.1. *Right:* Training with the full protocol from Table 6.1.

Table 6.2: Evaluation of the models trained in Figure 6.14. The figures were computed from the average performance metrics of the individual models, the relative change refers to the mean values.

Metric	No Stages				Environment-Based Stages				Change
	Mean	SD	Min	Max	Mean	SD	Min	Max	
<i>Success Rate</i>	0.072	0.044	0.012	0.329	0.794	0.144	0.414	0.997	1003 %
<i>Reward</i>	-26.745	2.326	-30.382	-14.267	4.838	5.945	-10.295	13.716	118 %
<i>Episode Steps</i>	18.838	0.709	14.658	19.803	6.348	2.794	1.924	13.350	-66 %
<i>Target Distance</i>	0.444	0.053	0.239	0.552	0.101	0.033	0.053	0.206	-77 %

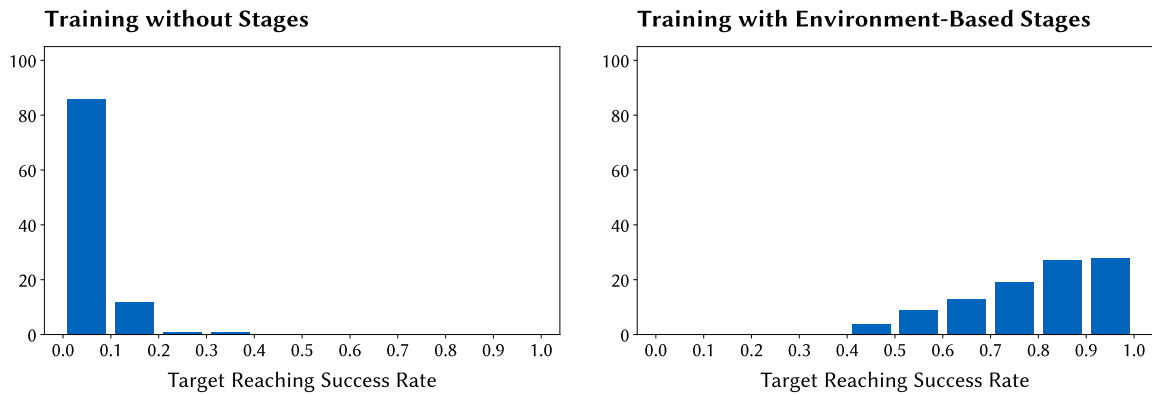


Figure 6.15: Histogram of the target reaching success rates of the models trained in Figure 6.14. The success rates were determined with the evaluation data set described in the text. *Left:* Training with fixed target radius $r_{target} = 0.1$ and exploration noise as described in Table 6.1. *Right:* Training with the full protocol from Table 6.1.

exploration noise were set to their final values from the first stage on. Another difference is that the replay buffer is not cleared at the end of each stage. This is only required when r_{target} changes in a new stage and stored observations become invalid. In essence, this corresponds to standard training without a protocol, but with a discrete exploration noise decay.

It is obvious at first glance that training with environment-based stages clearly outperforms the baseline version without training protocol. At the beginning of each stage, the performance drops, but always remains above the value from the start of the training, which shows that knowledge is retained across the stages. Within each stage, learning progresses at a higher rate compared to the baseline. This in particular also applies to the final stage, where the values of the target radius r_{target} are identical for both versions. The final models from all runs were evaluated on a test data set with 50 000 samples, each consisting of a random target position and a start configuration for the robot. This ensures that the evaluation results are consistent and reproducible. A subset of the sample target positions is depicted in Figure 6.10. Table 6.2 provides a summary of the results that was computed based on the average performance metrics of the individual runs on the test data set. The target reaching success rate is orders of magnitude higher when training with environment-based stages, with the best model achieving close to 100%. Even the model with the worst test results still outperforms the best one from the baseline training runs. The increase of the overall model performance can also be seen in Figure 6.15, which shows histograms of the target reaching success rates for both training without stages and training with environment-based stages. Tests with more steps in the last stage of the protocol from Table 6.1 indicate that the number of models with a success rate of close to 100% can be further increased through longer training.

Results of Training with Robot-Based Stages In the following, we will consider a different type of training protocol that implements robot-based stages and thus corresponds to the second main dimension of variation of the experiment introduced at the beginning of this section. This makes it possible to study the freezing and freeing of DoFs, which is hypothesized to guide the learning of motor skills in humans. Table 6.3 defines a training protocol that frees the robot’s DoFs in a sequential order from its base to the tip. Like before, the robot’s outermost joint always remains locked. Since some joints share the same rotation axis in specific constraint configurations, they are not freed one after another, but in a sequence which ensures that the reachable workspace grows from stage to stage. Compared to the environment-based training protocol from Table 6.1, the target radius r_{target} and the standard deviation σ_{exp} of the exploration noise remain constant over all stages. Moreover, the replay buffer is retained, which is why only the first stage includes a warm-up phase. An important difference from environment-based stages is that the space reachable by the robot changes in every stage. Therefore, when sampling a new target at the beginning of an episode, it is not possible to select an arbitrary point in the robot’s workspace, as it may not be reachable with the current set of constraints. For this reason, sampling is performed in the configuration space for stages in which one or more of the robot’s first six joints are frozen. The resulting target point is then computed with the *ikpy*-based robot model mentioned earlier. In general, this sampling method does not yield a uniform coverage of the portion of the workspace that is reachable in a certain protocol stage. However, this is compensated for by switching to uniform workspace sampling as soon as all six joints are freed in the final stage.

Figure 6.16 provides a summary of 100 training runs. While not as high as for environment-based stages, there is still a considerable performance gain compared to training without stages. The performance losses at the beginnings of the stages are larger, which seems natural as the addition of further joints can be interpreted as the definition of a new learning task. Within the individual stages, the systems learn faster than the baseline version. This is in particular also true for the last stage, where all joints are unlocked, indicating that knowledge from the previous stages is retained. The results of the evaluation of the trained models on the test data set are listed in Table 6.6 and histograms of the target reaching success rates are shown in Figure 6.19. As in the case of training with environment-based stages, not only does the average success rate improve with the training protocol, but also the distribution of target reaching success rates.

To further investigate the effectiveness of training with robot-based stages, we evaluated another training protocol with an identical sequence of robot joint constraints but a larger target radius $r_{target} = 0.2$. Its definition is listed in Table 6.4, the corresponding evaluation results are available in Figure 6.17, Table 6.7 and Figure 6.20. The increased target radius turns out to simplify the task significantly. Already the baseline models achieve good results and the performance gain from staged training is lower. However, as can be seen from the distribution of success rates in Figure 6.20, almost all staged training runs yield high performance, while the baseline runs have a much higher variance. This clearly shows that training protocols can be beneficial even for simple tasks where they are not strictly necessary to achieve good results.

The sequential freeing of DoFs is only one possibility for the definition of the training protocol stages. This raises the question of how the specific sequence of robot joint constraints that is applied during training influences the learning process. The training protocol defined in Table 6.5 encompasses the same total number of steps as the that from Table 6.3. In particular, the lengths of the final stages are identical. The difference is that the DoFs are not unlocked sequentially. Instead, the protocol iterates over different subsets of joints that increase in size as learning progresses. Note that not all possible combinations are considered to limit the length of the protocol. For example, the combination of the fifth and the sixth joint is omitted because it covers only a very small portion of the robot’s workspace. The training progress and the evaluation results are provided in Figure 6.18, Table 6.8 and Figure 6.21. In contrast to all other training protocols implemented so far, the performance of the models trained in stages lies below the baseline. During the first stages, the systems learns quickly, similar to the training with the schedule from Table 6.3 in Figure 6.16. However, the performance starts to decrease by the end of the third stage and the learning speed becomes very slow in all subsequent stages. These results indicate that the specific order in which the robot’s DoFs are freed and frozen has a substantial impact on the effectiveness of training protocols. A possible explanation for the observed behavior is that an unfavorable training protocol can push the system towards suboptimal regions of the solution space, similar to a poor choice of hyperparameters.

Conclusion Our results clearly show that both environment-based stages and robot-based stages can improve the learning process significantly. This makes the training protocol framework introduced in this thesis a promising new tool for learning not only in neurorobotics, but robotics in general. Based on the results presented in this section, a wide range of future directions of research opens up. An important next step is the combination of environment-based stages and robot-based stages in a single protocol. Since the application of training protocols is not limited to Algorithm 6, it will also be interesting to evaluate them on other algorithms such as *TD3*, which is an improved version of the original DDPG algorithm [519]. Another research topic is the definition of training protocol stages. In a first step, future versions of the framework could include support for dynamic stage lengths, which will make it possible to automatically proceed to the next stage when the performance saturates or, like in the third stage of the training runs shown in Figure 6.18, starts to decrease. A second step is then to not only automatically adapt stage lengths but to learn the definition of the complete training protocol. The identification of meaningful stages is very challenging, which is particularly reflected in the finding that the specific sequence in which the robot’s DoFs are freed and frozen directly impacts the performance of training with robot-based stages. The automatic generation of task-optimized training protocols will not only make them more easily usable, but also enable comparative studies with data from biology.

Table 6.3: A training protocol with robot-based stages and sequential freeing of DoFs. The columns are defined as in Table 6.1.

Stage d	\mathbf{F}_d	o_d	$\mathbf{H}_d[r_{target}]$	$\mathbf{H}_d[\sigma_{exp}]$	$\mathbf{H}_d[steps]$	$\mathbf{H}_d[warm-up]$
1	Diag(1, 1, 0, 0, 0, 0, 0)	0	0.10	0.042	80 000	1000
2	Diag(1, 1, 0, 1, 0, 0, 0)	0	0.10	0.042	80 000	0
3	Diag(1, 1, 1, 1, 0, 0, 0)	0	0.10	0.042	80 000	0
4	Diag(1, 1, 1, 1, 0, 1, 0)	0	0.10	0.042	80 000	0
5	Diag(1, 1, 1, 1, 1, 1, 0)	0	0.10	0.042	250 000	0

Table 6.4: A training protocol with robot-based stages and sequential freeing of DoFs with lower target reaching accuracy compared to Table 6.3. The columns are defined as in Table 6.1.

Stage d	\mathbf{F}_d	o_d	$\mathbf{H}_d[r_{target}]$	$\mathbf{H}_d[\sigma_{exp}]$	$\mathbf{H}_d[steps]$	$\mathbf{H}_d[warm-up]$
1	Diag(1, 1, 0, 0, 0, 0, 0)	0	0.20	0.042	20 000	1000
2	Diag(1, 1, 0, 1, 0, 0, 0)	0	0.20	0.042	20 000	0
3	Diag(1, 1, 1, 1, 0, 0, 0)	0	0.20	0.042	20 000	0
4	Diag(1, 1, 1, 1, 0, 1, 0)	0	0.20	0.042	20 000	0
5	Diag(1, 1, 1, 1, 1, 1, 0)	0	0.20	0.042	270 000	0

Table 6.5: A training protocol with robot-based stages and non-sequential freeing of DoFs. The columns are defined as in Table 6.1.

Stage d	\mathbf{F}_d	o_d	$\mathbf{H}_d[r_{target}]$	$\mathbf{H}_d[\sigma_{exp}]$	$\mathbf{H}_d[steps]$	$\mathbf{H}_d[warm-up]$
1	Diag(1, 1, 0, 0, 0, 0, 0)	0	0.10	0.042	50 000	1000
2	Diag(0, 0, 1, 1, 0, 0, 0)	0	0.10	0.042	50 000	0
3	Diag(1, 0, 0, 1, 0, 0, 0)	0	0.10	0.042	50 000	0
4	Diag(0, 1, 1, 1, 0, 0, 0)	0	0.10	0.042	80 000	0
5	Diag(1, 1, 1, 1, 0, 0, 0)	0	0.10	0.042	90 000	0
6	Diag(1, 1, 1, 1, 1, 1, 0)	0	0.10	0.042	250 000	0

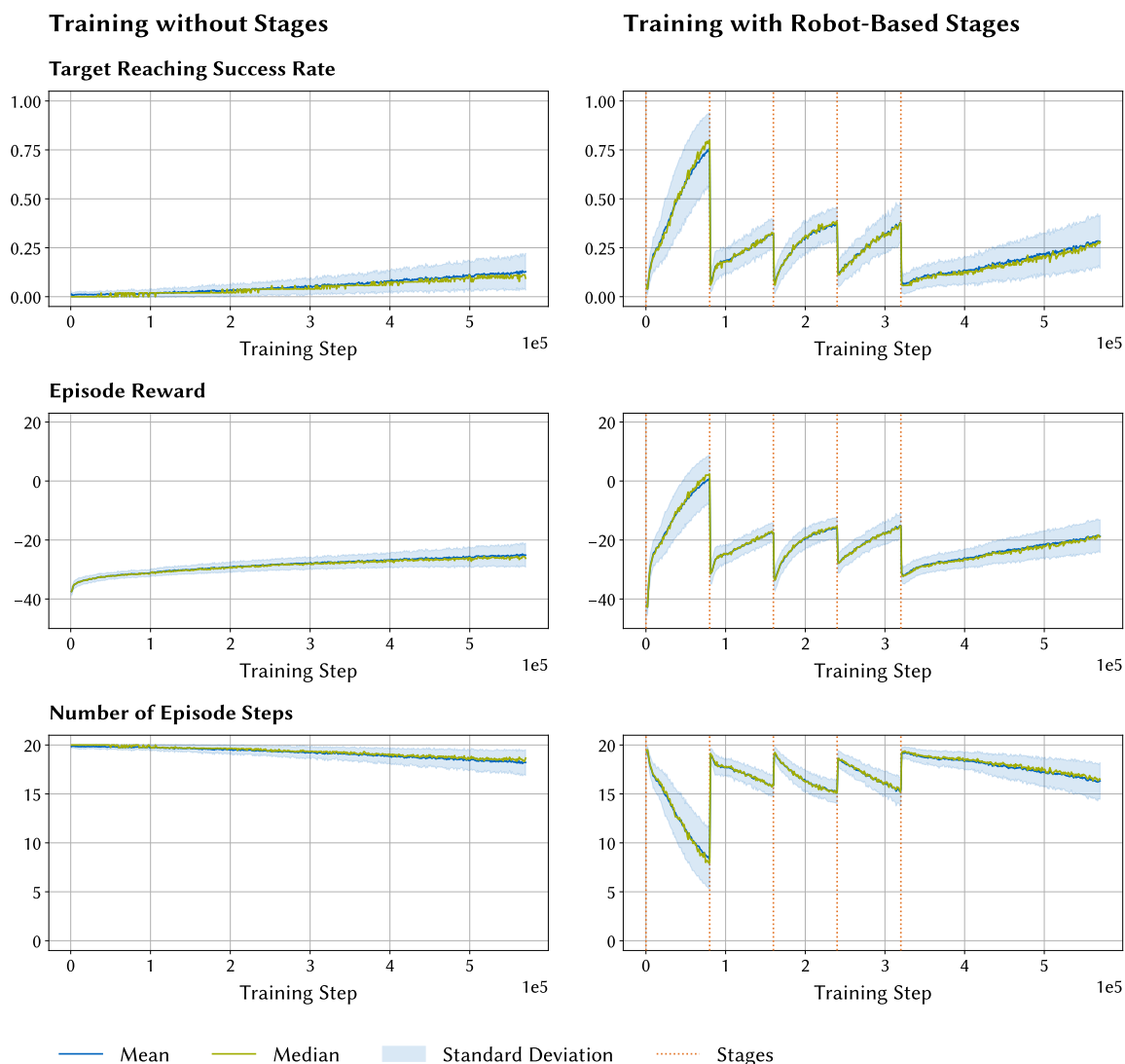


Figure 6.16: Results from 100 training runs of Algorithm 6 with robot-based stages. The data points from each run are averages from 1000 PCA cycles. *Left:* Training without stages. *Right:* Training with the protocol from Table 6.3.

Table 6.6: Evaluation of the models trained in Figure 6.16. The figures were computed from the average performance metrics of the individual models, the relative change refers to the mean values.

Metric	No Stages				Robot-Based Stages				Change
	Mean	SD	Min	Max	Mean	SD	Min	Max	
<i>Success Rate</i>	0.093	0.056	0.019	0.289	0.199	0.087	0.022	0.617	114 %
<i>Reward</i>	-25.913	2.948	-30.365	-16.486	-20.560	4.307	-29.989	-1.722	21 %
<i>Episode Steps</i>	18.497	0.900	15.286	19.690	16.891	1.437	9.564	19.647	-9 %
<i>Target Distance</i>	0.436	0.061	0.281	0.542	0.335	0.072	0.122	0.527	-23 %

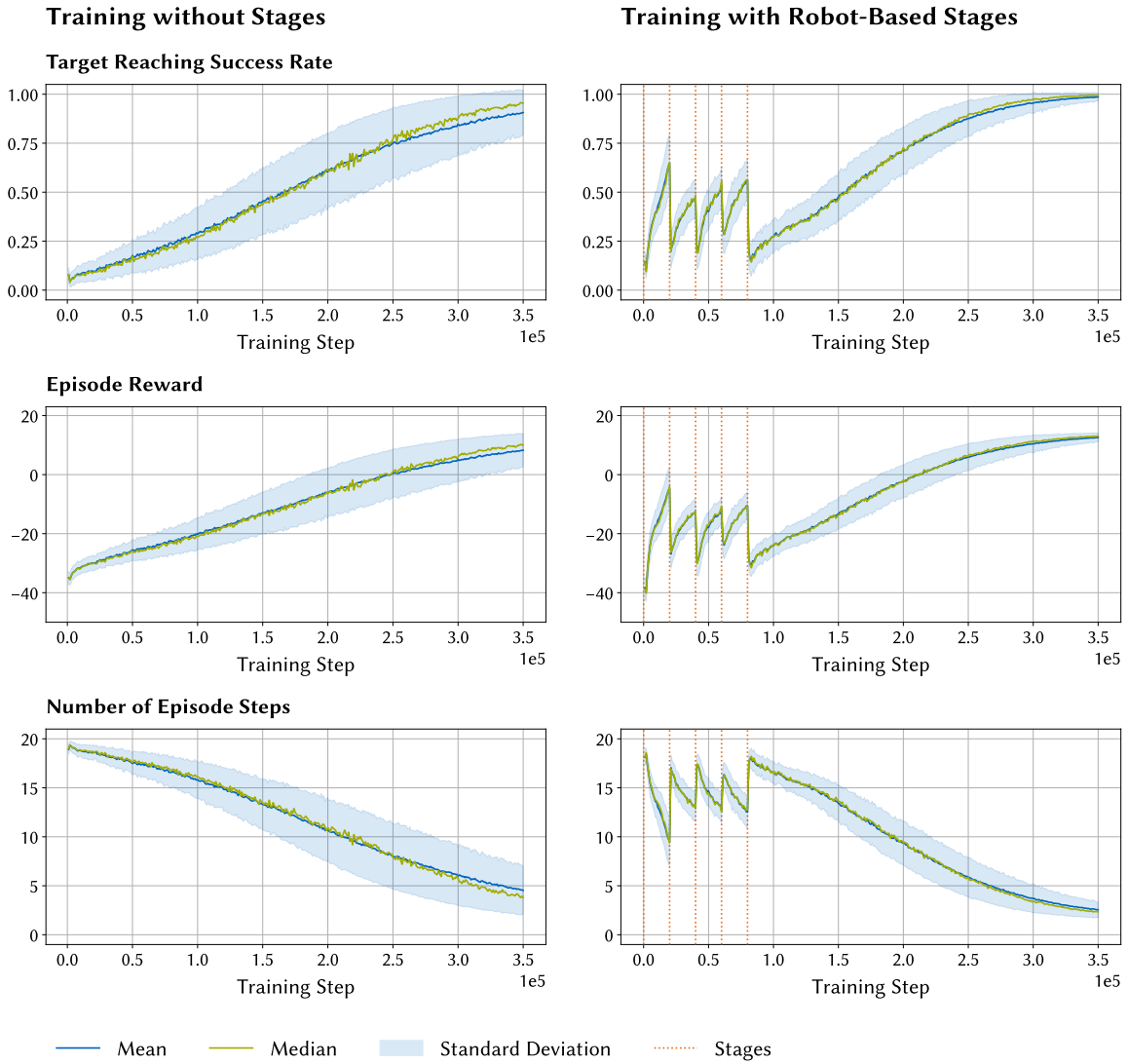


Figure 6.17: Results from 100 training runs of Algorithm 6 with robot-based stages and low positioning accuracy. The data points from each run are averages from 1000 PCA cycles. *Left:* Training without stages. *Right:* Training with the protocol from Table 6.4.

Table 6.7: Evaluation of the models trained in Figure 6.17. The figures were computed from the average performance metrics of the individual models, the relative change refers to the mean values.

Metric	No Stages				Robot-Based Stages				Change
	Mean	SD	Min	Max	Mean	SD	Min	Max	
<i>Success Rate</i>	0.867	0.129	0.436	0.999	0.967	0.036	0.830	1.000	12 %
<i>Reward</i>	7.367	6.098	-12.119	14.246	12.307	1.923	5.431	14.516	67 %
<i>Episode Steps</i>	4.756	2.670	1.512	12.832	2.543	0.965	1.344	5.843	-47 %
<i>Target Distance</i>	0.170	0.051	0.107	0.339	0.131	0.016	0.104	0.180	-23 %

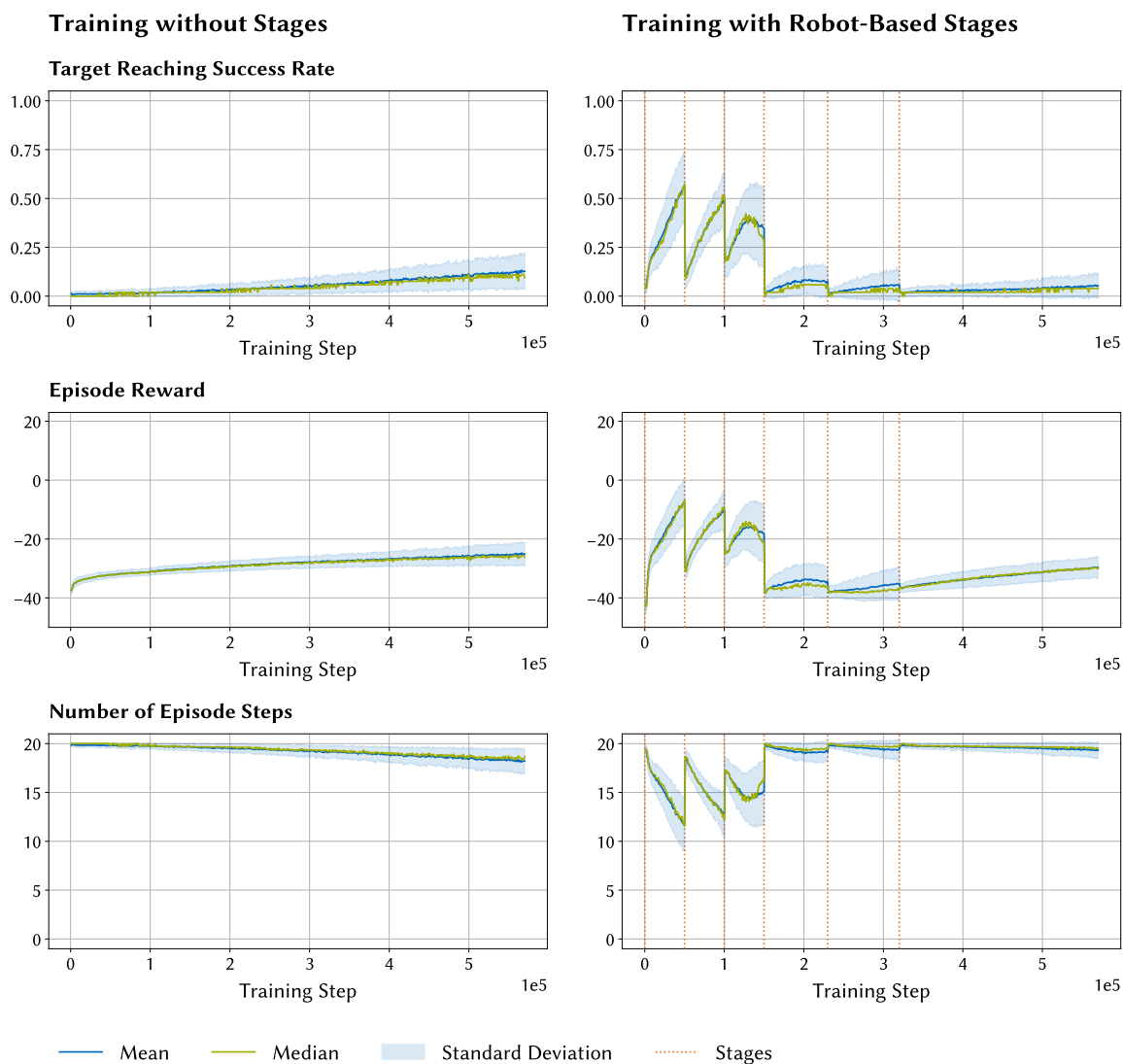


Figure 6.18: Results from 100 training runs of Algorithm 6 with robot-based stages and both freeing and freezing of DoFs. The data points from each run are averages from 1000 PCA cycles. *Left:* Training without stages. *Right:* Training with the protocol from Table 6.5.

Table 6.8: Evaluation of the models trained in Figure 6.18. The figures were computed from the average performance metrics of the individual models, the relative change refers to the mean values.

Metric	No Stages				Robot-Based Stages				Change
	Mean	SD	Min	Max	Mean	SD	Min	Max	
Success Rate	0.093	0.056	0.019	0.289	0.041	0.039	0.013	0.286	-56 %
Reward	-25.913	2.948	-30.365	-16.486	-29.949	3.024	-37.040	-17.019	-16 %
Episode Steps	18.497	0.900	15.286	19.690	19.413	0.610	15.533	19.829	5 %
Target Distance	0.436	0.061	0.281	0.542	0.539	0.103	0.282	0.869	24 %

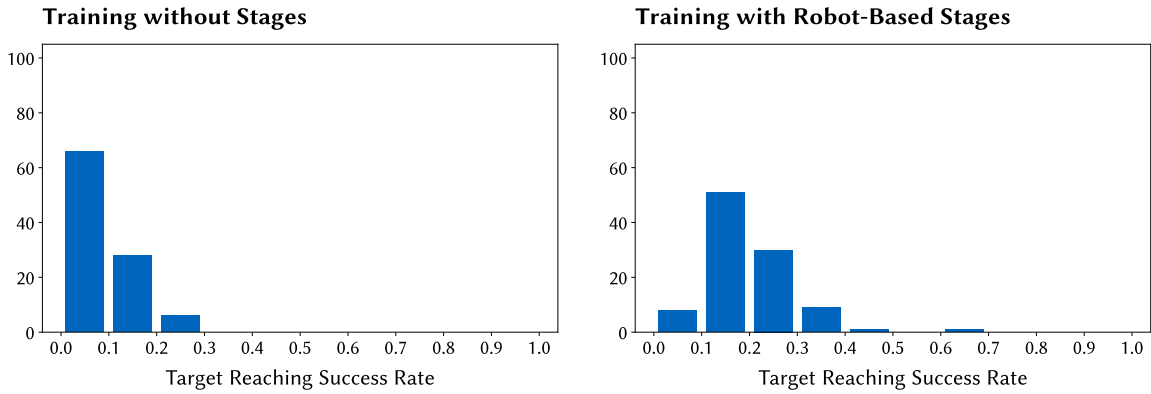


Figure 6.19: Histogram of the target reaching success rates of the models trained in Figure 6.16. The success rates were determined with the evaluation data set described in the text. *Left:* Training without stages. *Right:* Training with the protocol from Table 6.3.

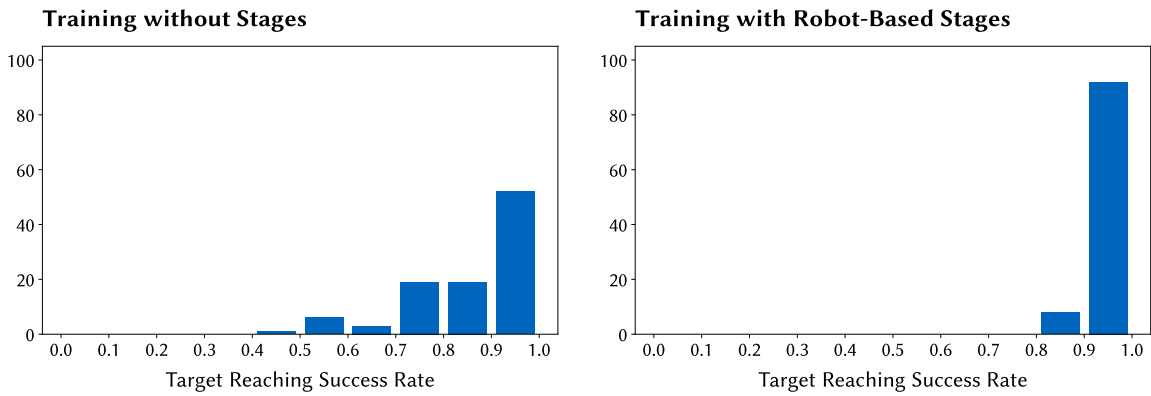


Figure 6.20: Histogram of the target reaching success rates of the models trained in Figure 6.17. The success rates were determined with the evaluation data set described in the text. *Left:* Training without stages. *Right:* Training with the protocol from Table 6.4.

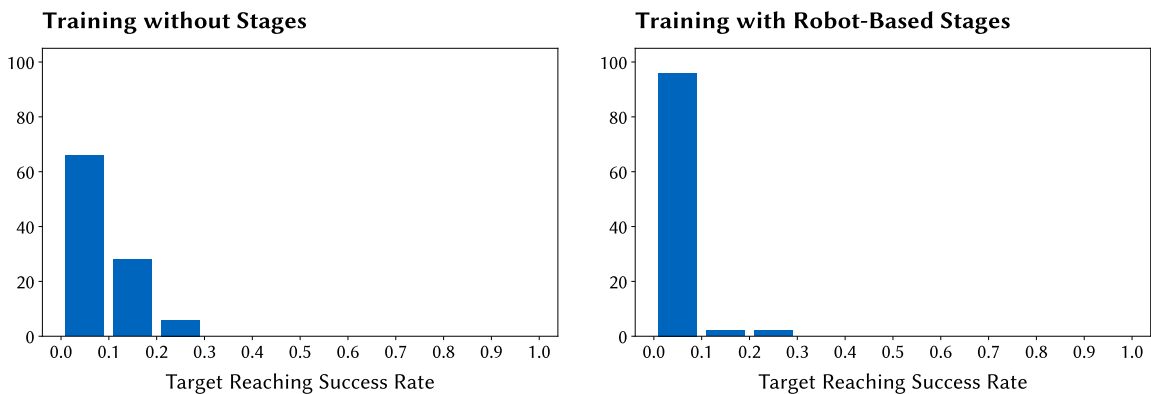


Figure 6.21: Histogram of the target reaching success rates of the models trained in Figure 6.18. The success rates were determined with the evaluation data set described in the text. *Left:* Training without stages. *Right:* Training with the protocol from Table 6.5.

7

Conclusion and Outlook

This work has brought together ideas, concepts, findings and models from a highly diverse range of different fields of research centered around robotics, AI and neuroscience. The common thread that links these disciplines is embodiment, which is sometimes made the focus of study, as in developmental robotics, and sometimes completely neglected, as in data-driven AI. Currently, there is still a huge gap between research on embodied systems and traditional approaches. While the former are typically argued to have a huge potential of answering central questions in brain research and overcoming longstanding challenges in robotics and AI, the latter deliver better results in practice and are easier to handle both analytically and technically. The core idea that we have developed and realized over the past chapters is to bridge this gap by designing a new framework for brain-derived learning that draws from insights in brain research to combine the conceptual elegance of embodiment with the practical advantages of data-driven AI. In the following three sections, we will summarize our main contributions, highlight prospective practical applications, and identify future directions of research.

7.1 Main Contributions of this Work

Our main contributions focus on the individual steps required to design and train a learning embodied system as outlined in Figure 1.7. Chapters 2 and 3 introduce the theoretical background that is required for a principled implementation of the concepts developed in the subsequent chapters. Our first main contribution is the formulation of a modern definition of neurorobotics.

A Modern Definition of Neurorobotics We have developed a new definition of *neurorobotics* that takes into account the most recent results in the field. It also gives way to the introduction of a formal concept for *neurorobotics experiments*. The specification of *experiment classes* makes it possible to abstract from concrete experimental setups and thereby cover a wide range of variations. Having a multitude of models is essential in order to capture the diversity of the real world and can only be reproduced efficiently in large scale in simulations. For this reason, all of the ideas and concepts developed in this work have been implemented in the HBP Neurorobotics Platform.

Large-Scale Virtual Neurorobotics The new tool set developed in Chapter 4 based on the NRP, makes it possible to run parallel distributed experiments with an arbitrary number of simulation instances. This makes high-performance computing accessible for the first time to robotics and thereby introduces a substantial paradigmatic change. Physical robotics experiments are very constrained due to high costs and complex setup procedures. Virtual experiments, by contrast, are only limited by computing power and storage, both of which have become highly scalable with the widespread adoption of cloud computing. An indicator for the capability of the new tool can be seen in the fact that none of the neurorobotics experiments implemented in this work has ever been executed on a physical robot.

Neuromorphic Computing for Neurorobotics The second extension for the NRP, a network-based interface to Intel’s neuromorphic chip Loihi, makes it possible to accelerate neurorobotics experiments with spiking neuron models at high speed and efficiency. In fact, the NRP can be seen as a virtual development environment for embedded neuromorphic algorithms with applications ranging from highly-energy efficient wearables to adaptive systems with on-chip learning for automotive applications.

Biomimetic Neurorobotics The TUM Robot Mouse enables the transfer of knowledge from virtual neurorobotics experiments with highly realistic musculoskeletal body models to physical robots. Its compliant design with an actuated spine takes advantage of advances in rapid manufacturing, which makes it possible to produce the robot body at low cost with 3D printing. Design changes can thus be implemented quickly with little effort. This makes the robot a unique tool that complements virtual neurorobotics experiments. While the initial versions of the robot were built with a focus on locomotion, a first prototype of the head with cameras is already available. Its small size makes the TUM Robot Mouse also an interesting tool for industrial applications such as inspection tasks.

Brain-Derived Modular Neural Networks The definition of a novel brain-derived modular-hierarchical neural network (MHNN) architecture is one of our main contributions. It is based on findings about the modular organization of the brain and employs state-of-the-art DNNs as individual building blocks. Unlike previous models, this makes it possible to take advantage of automatically learned deep feature hierarchies, and at the

same time enables individual functions to be localized in sub-networks. The resulting networks are therefore more transparent than traditional monolithic neural networks. While other modular neural network architectures have been proposed earlier, our architecture is unique in two important aspects. First, the support for standard DNNs and gradient-based optimization enables compatibility to a huge body of existing network models and software tools. Second, there are two types of network modules: modality-specific component networks that process sensory input and task-specific hub networks that produce behavioral output based on data received from the component networks. Data fusion is performed individually in every hub network and therefore task-based.

Self-Supervised Learning of Deep Multisensory Maps The individual component networks of MHNNs project raw input data into an abstract latent space. In the brain, such projections have been found to be topographic representations of the input spaces that are aligned across modalities in some brain regions. To date, SOMs are one of the most common neural network architectures for modeling cortical maps. However, they do not support hierarchical feature extraction like DNNs. This work has introduced a novel topographic map loss that enables gradient-based learning of topographic maps. We have shown that two neural networks can be trained in parallel based on this loss in a self-supervised setup to enable the learning of a common aligned topographic map. By changing the hyperparameters of the loss function, it is possible to select one dominant network that determines the shape of the final map. Interestingly, similar phenomena have been observed in the brain, where the formation of auditory maps in the superior colliculus is guided by visual maps.

Training Protocols Our final contribution is a concept for training protocols that controls the temporal order of learning processes in component networks and hub networks through schedules and stages. While curriculum learning has long been applied in machine learning, our work is the first to explore the application of a bio-inspired curriculum that is defined in terms of the DoFs of an embodied system to state-of-the-art deep reinforcement learning algorithms.

7.2 Practical Relevance

MHNNs are directly motivated by the findings about functional specialization in biological brains discussed at the beginning of Chapter 5. While it is possible to argue that they capture information processing in the brain more closely than standard ANNs, this does not necessarily imply that they are also better suited to practical applications in AI. But as it turns out, functional specialization across different component networks bears very promising potential for a number of implementation challenges that have not yet been addressed by DNNs.

Modular Neural Architectures An obvious feature of MHNNs is the possibility of specifying system architectures. Monolithic DNNs are thereby replaced by structured descriptions with discrete components that are closely intertwined with each other. This is an approach that has been pursued for decades, primarily in symbolic AI as part of the research on cognitive architectures [86]. The advantages of being able to explicitly state the actual structure of the system compared to defining it implicitly as part of a purely data-driven learning process are twofold. Firstly, a transparent system architecture makes it easier to better understand the resulting computations and to identify possible failure modes. Unexpected behavior can be traced to specific components and the interactions between them. Secondly, the individual building blocks of the architecture become reusable. This is very different from DNNs that must be re-trained for new domains even when only a small part of the input is different. Even if one cannot expect the components of a MHNN to be as clearly separated as in traditional software engineering, just the mere possibility of localizing sub-systems and their interactions goes considerably beyond the common practice in current AI systems.

Artificial Lesions for Neuromorphic Chips As outlined in Section 5.1, lesions have played a central role in the anatomical localization of brain functions. More than that, they also provide implicit information about the brain's resistance to physical damage. Based on a functional brain atlas, it is, for example, possible to predict cognitive impairments from brain injuries at different locations. Even though the quality of these predictions can vary greatly due to inter-individual differences and the general complexity of structural changes caused by brain injuries, nothing comparable has been achieved yet for ANNs. This does not mean that artificial lesions have not been considered. In fact, *dropout* [525], a common technique to avoid the overfitting of training data, is based on the random deactivation of neural units. But these "artificial lesions" provide no additional insight into how the structure of a neural network is related to its function. Such information is highly relevant in many practical applications, where the microcircuits of the processor that executes a neural network model can suffer from local defects due to imperfect manufacturing or due to aging during operation. Being able to predict the functional consequences of these structural defects can help to reduce manufacturing costs (as long as performance degradation caused by a defect remains below a certain threshold) and to ensure fault-tolerant operation (by guaranteeing that common defects do not severely degrade performance).

Interpretability An implicit assumption behind the idea of quantifying the effects of structural damage to a neural network on its functional performance is that different sub-networks or neurons serve different purposes that are of different criticality. This becomes most obvious with the example of a DNN for classification, where damage to one of the neurons in the output layer directly results in the loss of recognizing the class this neuron represents. Effects are likely to be less drastic for a neuron that represents the value of an input pixel. Predictions for the effect of damage to the input and output layer are relatively easy because each neuron inside them has a concrete semantic meaning.

However, it is extremely challenging to draw similar conclusions for other layers of the network, where information is no longer clearly separable as a direct consequence of the currently prevalent monolithic deep architectures. By contrast, a modular network design with clear spatial separation of distinct functions across the network architecture allows for a better judgment of damage. In particular, this works not only for simple cases like classes or pixels but also for complete functional sub-systems. A striking example in this context is the *mushroom body* in the brain of the fly *Drosophila melanogaster*. Lesion studies have revealed that the loss of this part of the brain, even though it comprises only about 2500 neurons [526], impairs associative learning of odor cues [527]. MHNNs with clearly localized functions therefore have a great potential for safety-critical applications, where a continuous real-time assessment of system performance is indispensable.

Self-Repairing Neuromorphic Systems Parts of an integrated circuit might fail during operation. As long as the overall integrity of the chip is preserved, it may continue to operate with the defective components. A common source of defects are imperfections in the manufacturing process. Issues arising at this early stage can be mitigated by redundancies in the design of the chip that make it possible to simply disable malfunctioning components on the die [526]. In general, better techniques for handling manufacturing defects can increase yield and lower production costs. However, improving yield solely based on hardware is limited to measures taken during the design phase of the chip. Once the functional degradation of a specific hardware defect becomes predictable, even units that fail testing can still be viable for some applications. In a sense, this is comparable to earlier approaches where CPUs with malfunctioning cores were sold for a lower price, since they still could be used for computationally less demanding tasks [528].

As already described, MHNNs fit perfectly into this concept since they allow for a direct mapping between possible hardware defects and performance degradation. An important prerequisite is, however, that the network is executed on a neuromorphic system where the execution units on the chip actually correspond to neurons and synapses. In traditional von Neumann processors, storage and computation are highly centralized, which implies that defects usually affect the system globally. Defects that are already known directly after manufacturing can be handled offline in a safe environment. By contrast, chip failure during runtime in applications such as automated driving is potentially highly safety-critical. With standard monolithic DNNs, there is usually no way to assess the system's remaining reliability and the corresponding system must consequently be shut down completely. In MHNNs, defects can be directly mapped to the corresponding functional components of the network. Operation can be continued if the overall system can still function reliably enough (e.g. when radar data can no longer be processed and camera images are sufficient for the current environmental conditions). It may even be possible to relocate network components to undamaged parts of the chip, just like biological brains can recover from injury through gradual reorganization of the cortical network. The idea of self-repairing control systems is, in fact, already several decades old and has been studied extensively in the field of aviation [529].

7.3 Future Directions of Research

The results of this work mark a starting point for many new directions of research. They are motivated by both open questions that can be addressed by directly extending the models and algorithms presented in the previous chapters and by completely new challenges that can be solved with the framework of virtual neurorobotics and advanced embodied learning.

Neural Modeling As outlined in Chapter 2, there is a huge diversity of different neural coding schemes. In the scope of this thesis, we have only considered rate codes implemented by simple analog neurons or, in the case of the PPC model from Section 6.2, by spiking LIF neurons. The simulation of this model in the NRP revealed a major disadvantage of population codes. The latency for transmitting new stimuli is limited by the time that is required to average the population activity. Shorter time windows for averaging can decrease the latency, but will result in lower accuracy. Encoding information with precisely timed spikes therefore promises to be faster and also more efficient, because fewer spikes must be generated in order to transmit a stimulus. The low latencies observed in the human visual system suggest that even the brain employs advanced spike coding to realize fast response times [101]. In general, the encoding and representation of information in the brain is still not fully understood. Neurorobotics offers a promising perspective by enabling the study of emerging neural activity during closed-loop interaction with the environment. Neural codes that develop as a result of this interaction can be directly interpreted in terms of the behavior that is related to them.

Integrating them into an NRP experiment will enable the simulation of body growth, which is another unique feature of virtual experiments that cannot be realized with physical robots on the same scale and level of detail. Finally, in the future, training protocols should be evaluated on online learning algorithms that do not require separate phases for training and inference.

Advanced Training Protocols Training protocols have a high potential for guiding and shaping the learning of complex systems. This is also reflected by the fact that curriculum learning is widely applied in many machine learning tasks. The training protocol mechanism developed in Chapter 6 can be extended along many different dimensions. Most importantly, the stages are currently fixed, which makes them, in essence, a complex hyperparameter. By following a meta-learning paradigm training protocols can become part of the learning process in the future. This will also allow for better assessment of the impact of different scheduling variables on the system's performance. Rather than only considering the freezing and freeing of physical joints, an interesting avenue for future studies would be to analyze and modulate the dimension of the control space as discussed by Newell and Vaillancourt [531]. In particular, this may include the developmental progression of the complexity of the neural system. Another promising addition is the automatic definition of tasks through autotricula. Recent work in this direction was published by Plappert et al. [532], who trained a system with asymmetric



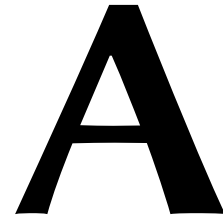
Figure 7.1: Modeling of body growth. The 3D body models cover all developmental stages from Carnegie stage 13 (an embryo about one month old) until adulthood. They are based on [20] and [530].

self-play between two agents. While one of them learns to create increasingly challenging tasks, the other learns to solve them. The second line of extending training protocols consists in enhancing the simulation of the body and the environment. As illustrated in Figure 7.1, we have already started working on a series of human body models that cover all developmental stages.

Natural Language Processing This work has mainly focused on sensorimotor learning. From a developmental point of view, this makes sense because basic perception and motor control are acquired at the beginning of the developmental process outlined in Figure 6.2. A longstanding research question is how models for low-level sensorimotor interaction can be combined with symbolic models for logical reasoning and natural language processing. The modular neural network architecture for question answering developed by Andreas et al. [432] partly achieves this task by mapping natural language questions to building blocks of neural networks. However, these networks need to be generated on the fly for every single question. The development of a unified model that integrates neural networks with formal grammars is not only a scientifically interesting research topic but also has huge practical relevance. Especially the programming of robots is still quite cumbersome and error-prone, and can greatly benefit from a language model that is formally well-defined and at the same time grounded in the environment. Work in this direction has already been published by Giuliani and Knoll [533].

Virtual Synthesis of Robotic Systems – A Vision A long-term research vision that is rooted in the contributions of this work and which builds on the research topics identified above is the virtual synthesis of robotics systems. Knoll and Walter [534] outlined this vision as an approach to generate a factory layout from scratch, solely based on the description of the product to be manufactured. The interface between the human and the robot is thus radically simplified and reduced to the formulation of the actual goal without any intermediate steps of abstraction. Realizing this vision is only possible through a complete virtualization of the robot and its environment. As described in this thesis, this will enable highly accelerated training in massively parallel simulations and allow huge solution spaces to be covered within realistic time frames and without physical constraints. Only then it becomes possible to completely automate manufacturing tasks without lengthy planning and programming. The results of this thesis and the research topics presented in the last paragraphs, such as advanced training protocols or language processing that is grounded in the environment, mark important milestones towards this vision.

In this thesis, we have developed tools, models and methods for advanced embodied learning solely based on virtual neurorobotics experiments. The transition from physical experiments to large scale simulations of virtual environments marks a disruptive paradigm shift in robotics. We hope that our findings will contribute significantly to this shift and will advance the state of the art in brain-derived cognitive models for robotics.



Parameters of Models and Algorithms

A.1 Deep Multisensory Neural Maps

Parameters for the Generation of the Multimodal Data Set

Table A.1 lists the parameters that were used to generate the multimodal data set D_{map} with Algorithm 2. The environment parameter $E_{NRP}.robot_offset$ is 0.36, `default_configuration` refers to the robot's upright home position with all joint angles set to 0. The simulations were executed with an extended version of the NRP release 3.0.0.

Table A.1: Parameters for the generation the multimodal data set D_{map} with Algorithm 2.

Parameter	Value	Description
K	102 000	The number of samples to be generated
r_{min}	0.410	Minimum radius of the sampling space
r_{max}	1.096	Maximum radius of the sampling space

Hyperparameters for the Training Algorithms and Loss Functions

The following tables list the hyperparameters that were used to train the different variants of deep multisensory maps on the multimodal data set D_{map} , including the baseline from Figure 5.9, which was generated with supervised learning. Unless otherwise indicated, the parameter values are identical for the vision network C_{vis} and the proprioception network C_{pro} . The neural network models were implemented with PyTorch 1.8.0 and PyTorch Lightning 1.0.8. All models were trained with the Adam optimizer [467] and all training sessions were started with different random seeds.

Table A.2: Hyperparameters for Figure 5.9. Supervised training of the vision network C_{vis} and the proprioception network C_{pro} on D_{map}^{vis} and D_{map}^{pro} with the MSE loss function from Equation 5.23.

Parameter	Value	Description
b	64	Batch size
η_{vis}	2.42×10^{-5}	Learning rate for C_{vis}
η_{pro}	7.54×10^{-4}	Learning rate for C_{pro}
β_1	9.00×10^{-1}	First moment exponential decay rate in the Adam optimizer
β_2	9.99×10^{-1}	Second raw moment exponential decay rate in the Adam optimizer
ϵ_{Adam}	1.00×10^{-8}	Denominator offset in the Adam optimizer
λ_{vis}	5.30×10^{-3}	L2 weight decay factor for C_{vis}
λ_{pro}	1.14×10^{-5}	L2 weight decay factor for C_{pro}

Table A.3: Hyperparameters for Figures 5.11 and 5.12. Self-supervised training of the vision network C_{vis} and the proprioception network C_{pro} on D_{map} with Algorithm 3 and the loss functions \mathcal{L}_{vis}^{self} and \mathcal{L}_{pro}^{self} from Equations 5.24 and 5.25.

Parameter	Value	Description
b	64	Batch size
η_{vis}	1.00×10^{-3}	Learning rate for C_{vis}
η_{pro}	1.00×10^{-3}	Learning rate for C_{pro}
β_1	9.00×10^{-1}	First moment exponential decay rate in the Adam optimizer
β_2	9.99×10^{-1}	Second raw moment exponential decay rate in the Adam optimizer
ϵ_{Adam}	1.00×10^{-8}	Denominator offset in the Adam optimizer
λ_{vis}	1.00×10^{-5}	L2 weight decay factor for C_{vis}
λ_{pro}	1.00×10^{-5}	L2 weight decay factor for C_{pro}
ζ	1.00×10^{-3}	Inverse variance factor in \mathcal{L}_{vis}^{self} and \mathcal{L}_{pro}^{self}
ϵ	1.00×10^{-7}	Denominator offset in \mathcal{L}_{vis}^{self} and \mathcal{L}_{pro}^{self}
τ	1.00×10^{-2}	Target network update rate in Algorithm 3

Table A.4: Hyperparameters for Figures 5.14 and 5.15. Unsupervised training of the vision network C_{vis} and the proprioception network C_{pro} on D_{map} with the topographic loss function \mathcal{L}^{top} from Equation 5.30 based on Algorithms 4 and 5.

Parameter	Value	Description
b	64	Batch size
S	8	Number of spheres per batch
r	9.00×10^{-2}	Sphere radius in the workspace
η_{vis}	1.00×10^{-3}	Learning rate for C_{vis}
η_{pro}	1.00×10^{-3}	Learning rate for C_{pro}
β_1	9.00×10^{-1}	First moment exponential decay rate in the Adam optimizer
β_2	9.99×10^{-1}	Second raw moment exponential decay rate in the Adam optimizer
ϵ_{Adam}	1.00×10^{-8}	Denominator offset in the Adam optimizer
λ_{vis}	1.00×10^{-5}	L2 weight decay factor for C_{vis}
λ_{pro}	1.00×10^{-5}	L2 weight decay factor for C_{pro}
β_{nbh}	1.00×10^{-2}	Neighborhood loss factor in \mathcal{L}^{top}
β_{sep}	5.00×10^{-2}	Separation loss factor in \mathcal{L}^{top}

Table A.5: Hyperparameters for Example Run 1 from Figure 5.17. Self-supervised training of the vision network C_{vis} and the proprioception network C_{pro} on D_{map} with Algorithm 3 and the topographic map loss functions \mathcal{L}_{vis}^{map} and \mathcal{L}_{pro}^{map} from Equations 5.31 and 5.32.

Parameter	Value	Description
b	64	Batch size
S	8	Number of spheres per batch
r	9.00×10^{-2}	Sphere radius in the workspace
η_{vis}	1.00×10^{-3}	Learning rate for C_{vis}
η_{pro}	1.00×10^{-3}	Learning rate for C_{pro}
β_1	9.00×10^{-1}	First moment exponential decay rate in the Adam optimizer
β_2	9.99×10^{-1}	Second raw moment exponential decay rate in the Adam optimizer
ϵ_{Adam}	1.00×10^{-8}	Denominator offset in the Adam optimizer
τ	1.00×10^{-2}	Target network update rate in Algorithm 3
λ_{vis}	1.00×10^{-5}	L2 weight decay factor for C_{vis}
λ_{pro}	1.00×10^{-5}	L2 weight decay factor for C_{pro}
α_{vis}	1.00×10^0	MSE loss factor in \mathcal{L}_{vis}^{map}
α_{pro}	1.00×10^0	MSE loss factor in \mathcal{L}_{pro}^{map}
β_{nbh}	1.00×10^{-2}	Neighborhood loss factor in \mathcal{L}^{top}
β_{sep}	5.00×10^{-2}	Separation loss factor in \mathcal{L}^{top}

A Parameters of Models and Algorithms

Table A.6: Hyperparameters for Example Run 2 from Figure 5.17. Self-supervised training of the vision network C_{vis} and the proprioception network C_{pro} on D_{map} with Algorithm 3 and the topographic map loss functions \mathcal{L}_{vis}^{map} and \mathcal{L}_{pro}^{map} from Equations 5.31 and 5.32. The MSE loss factor is higher for \mathcal{L}_{pro}^{map} than for \mathcal{L}_{vis}^{map} .

Parameter	Value	Description
b	64	Batch size
S	8	Number of spheres per batch
r	9.00×10^{-2}	Sphere radius in the workspace
η_{vis}	1.00×10^{-3}	Learning rate for C_{vis}
η_{pro}	1.00×10^{-3}	Learning rate for C_{pro}
β_1	9.00×10^{-1}	First moment exponential decay rate in the Adam optimizer
β_2	9.99×10^{-1}	Second raw moment exponential decay rate in the Adam optimizer
ϵ_{Adam}	1.00×10^{-8}	Denominator offset in the Adam optimizer
τ	7.00×10^{-1}	Target network update rate in Algorithm 3
λ_{vis}	1.00×10^{-5}	L2 weight decay factor for C_{vis}
λ_{pro}	1.00×10^{-5}	L2 weight decay factor for C_{pro}
α_{vis}	1.00×10^{-3}	MSE loss factor in \mathcal{L}_{vis}^{map}
α_{pro}	2.00×10^0	MSE loss factor in \mathcal{L}_{pro}^{map}
β_{nbh}	1.00×10^{-2}	Neighborhood loss factor in \mathcal{L}^{top}
β_{sep}	5.00×10^{-2}	Separation loss factor in \mathcal{L}^{top}

Table A.7: Hyperparameters for Example Run 3 from Figure 5.17. Self-supervised training with warm-up of the vision network C_{vis} and the proprioception network C_{pro} on D_{map} with Algorithm 3 and the topographic map loss functions \mathcal{L}_{vis}^{map} and \mathcal{L}_{pro}^{map} from Equations 5.31 and 5.32.

Parameter	Value	Description
$E_{warm-up}$	20	Number of warm-up training epochs
b	64	Batch size
S	8	Number of spheres per batch
r	9.00×10^{-2}	Sphere radius in the workspace
η_{vis}	1.00×10^{-3}	Learning rate for C_{vis}
η_{pro}	1.00×10^{-3}	Learning rate for C_{pro}
β_1	9.00×10^{-1}	First moment exponential decay rate in the Adam optimizer
β_2	9.99×10^{-1}	Second raw moment exponential decay rate in the Adam optimizer
ϵ_{Adam}	1.00×10^{-8}	Denominator offset in the Adam optimizer
τ	1.00×10^{-2}	Target network update rate in Algorithm 3
λ_{vis}	1.00×10^{-5}	L2 weight decay factor for C_{vis}
λ_{pro}	1.00×10^{-5}	L2 weight decay factor for C_{pro}
α_{vis}	1.00×10^0	MSE loss factor in \mathcal{L}_{vis}^{map}
α_{pro}	1.00×10^0	MSE loss factor in \mathcal{L}_{pro}^{map}
β_{nbh}	1.00×10^{-2}	Neighborhood loss factor in \mathcal{L}^{top}
β_{sep}	5.00×10^{-2}	Separation loss factor in \mathcal{L}^{top}

A.2 Neuromorphic Data Fusion

All simulations of the network model outside the NRP were performed with NEST 2.20.1. The simulation of the full experiment was executed on an extended version of the NRP release 3.0.0 using the included NEST 2.12.0.

The neural populations in the network model from Figure 6.7 are comprised of 51 neurons that represent a value range of $[0, 50]$. In the NRP experiment, they were replaced by 51 populations of 100 neurons each to compensate for the shorter simulation time step. The neurons in the sensory input populations were NEST poisson generators of type `poisson_generator`. The population activities for an input stimulus were determined by Gaussian tuning curves with receptive field size $\sigma_r = 3$ as described in Equation 6.8. The actual spike rate was computed by multiplying the tuning curve output with a gain factor that encoded the confidence of the estimate. In Figure 6.8, the gains of the vision estimate and the sound estimate were set to $g_V = 80.0$ and $g_S = 40.0$, respectively. In the NRP experiment from Figure 6.6, the gain g_V for the vision estimate was dynamically varied in the interval $[25.0, 80.0]$ depending on the current degree of occlusion of the Husky robot, which was computed based on the robot's current position. The population representing the integrated multisensory estimate was comprised of LIF neurons of type `iaf_psc_alpha`, whose parameters were set as listed in Table A.8.

Table A.8: Parameters of the LIF neurons in the neural population for multisensory integration from the network model shown in Figure 6.7. The parameter descriptions are adapted from [535].

Parameter	Value	Description
<code>E_L</code>	-70.0 mV	Resting membrane potential
<code>C_m</code>	80.0 pF	Membrane capacitance
<code>tau_m</code>	10.0 ms	Membrane time constant
<code>t_ref</code>	1.0 ms	Duration of the refractory period
<code>V_th</code>	-55.0 mV	Spike threshold
<code>V_reset</code>	-70.0 mV	Reset membrane potential
<code>tau_syn_ex</code>	2.0 ms	Rise time of the excitatory synaptic alpha function
<code>tau_syn_in</code>	2.0 ms	Rise time of the inhibitory synaptic alpha function
<code>I_e</code>	0.0 pA	Constant input current
<code>V_min</code>	$-\infty$ mV	Minimum membrane potential

All neurons were connected with the `one_to_one` connection pattern of NEST. This means that all neurons in the two populations encoding the sensory input were connected to the corresponding neurons in the population encoding the integrated multisensory estimate. All synaptic connections are based on NEST's `static_synapse` model with a fixed weight of 500 and a transmission delay of 1.0 ms.

A.3 Staged Reinforcement Learning

The models were implemented with TensorFlow 1.14.0, Keras 2.3.1 and keras-rl 0.4.2. The kinematics model of the robot for accelerated training was generated with ikpy 2.3.3 and the results were validated with an extended version of the NRP release 3.0.0. The hyperparameters listed in Table A.9 apply to both the actor network N_μ and the critic network N_q .

Table A.9: Hyperparameters for staged reinforcement learning with Algorithm 6.

Parameter	Value	Description
b	128	Batch size
$ \mathbf{B} $	$1.000\,00 \times 10^5$	Size of the replay buffer in Algorithm 6
η	1.00×10^{-3}	Learning rate
β_1	9.00×10^{-1}	First moment exponential decay rate in the Adam optimizer
β_2	9.99×10^{-1}	Second raw moment exponential decay rate in the Adam optimizer
ε_{Adam}	1.00×10^{-7}	Denominator offset in the Adam optimizer
τ	1.00×10^{-2}	Target network update rate in Algorithm 6
γ	9.80×10^{-1}	Discount factor in Equation 6.27

List of Acronyms

AER	Address Event Representation
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BBP	Blue Brain Project
CEM	Cross-Entropy Method
CLE	Closed-Loop Engine
CNN	Convolutional Neural Network
CPG	Central Pattern Generator
CPU	Central Processing Unit
DBN	Deep Belief Network
DDPG	Deep Deterministic Policy Gradient
DNN	Deep Neural Network
DoF	Degrees of Freedom
EEG	Electroencephalography
fMRI	Functional Magnetic Resonance Imaging
GPU	Graphics Processing Unit
HBP	Human Brain Project
ICT	Information and Communications Technology
LIF	Leaky Integrate-and-Fire
LSTM	Long Short-Term Memory

List of Acronyms

MCU	Microcontroller Unit
MDP	Markov Decision Process
MHNN	Modular-Hierarchical Neural Network
MLE	Maximum Likelihood Estimation
MNN	Modular Neural Network
MSE	Mean Squared Error
NRP	Neurorobotics Platform
OS	Operating System
PCA	Perception – Cognition – Action
PPC	Probabilistic Population Code
PSS	Physical Symbol System
RBM	Restricted Boltzmann Machine
REST	Representational State Transfer
ROS	Robot Operating System
SDF	Simulation Description Format
SNIP	Sequential Neural Interfacing Process
SNN	Spiking Neural Network
SOM	Self-Organizing Map
SRDF	Semantic Robot Description Format
STDP	Spike Timing-Dependent Plasticity
SVM	Support Vector Machine
TF	Transfer Function
UAT	Universal Approximation Theorem
URDF	Unified Robot Description Format
XML	Extensible Markup Language

List of Figures

1.1	Illustration of the <i>robotics paradox</i>	4
1.2	A newborn exploring its environment	5
1.3	Example of a basic symbolic reasoning system	7
1.4	Example of a basic machine learning system for image classification	9
1.5	Example of a basic feature space transformation	10
1.6	Web front end of the HBP Neurorobotics Platform	15
1.7	Contribution of this work in the context of the three waves of AI as defined in Section 1.2	19
2.1	The complexity of the human brain in facts and figures	22
2.2	Drawings of the Brodmann areas for the lateral view of the human brain	24
2.3	Overview of different types of neural codes	29
2.4	Generation and conduction of action potentials in biological neurons	31
2.5	Functionally equivalent electrical circuit of a LIF neuron and the resulting membrane potential trace for a constant input current $I(t) = I_0$	33
2.6	Schema of a general analog neuron model	35
2.7	Main components of a Perceptron	37
2.8	Photo of the Mark I Perceptron	39
2.9	Classification of the XOR dataset with the Perceptron	40
2.10	Convolution-based image filters	43
3.1	Embedding of an embodied system in its environment	57
3.2	Schematic description of how an embodied system interacts with its environment in a closed PCA loop	57
3.3	Overview of a typical reservoir computing system	60
3.4	Main components of the neuromusculoskeletal system of the human body	61
3.5	Schema of the three constituents of an enactive cognitive system and their interrelations	63
3.6	Schematic overview of a neurorobotics experiment	72
3.7	Schematic overview of the system architecture of the HBP Neurorobotics Platform	75
3.8	Synchronization of the environment simulation and the brain simulation by the CLE of the NRP	77
3.9	Configuration file hierarchy of an NRP experiment	80

4.1	Parallel data collection for learning to grasp objects on multiple robots . . .	83
4.2	Architecture of the grasp prediction network $N(\mathbf{I}_0, \mathbf{I}_t, \nu)$	85
4.3	Digital twin of the robot grasping experiment in the NRP “Holodeck”, a digital lab space for virtual neurorobotics experiments	86
4.4	Overview of all components of the NRP Grasping Experiment	89
4.5	Default system architecture of the NRP with Docker-based deployment . . .	92
4.6	Extended system architecture of the NRP for parallel distributed learning . .	92
4.7	Document type definitions for the data base of the extended NRP system architecture	93
4.8	Dashboard for monitoring the data collection in parallel distributed NRP experiments	94
4.9	Parallel distributed learning in the NRP Grasping Experiment	95
4.10	Samples of grasp attempts collected with the NRP Grasping Experiment . . .	96
4.11	Box plot with success rates of random grasp attempts for different object numbers in the NRP Grasping Experiment	97
4.12	Extended system architecture of the NRP with support for Intel Loihi	99
4.13	Native support for Intel Loihi in the NRP CLE	100
4.14	Vision for the TUM Robot Mouse	102
4.15	Hind leg designs for the TUM Robot Mouse	104
4.16	Prototype of the spine for the TUM Robot Mouse	105
4.17	Initial version of the TUM Robot Mouse	106
4.18	Fully 3D printed leg and spine in a refined version of the TUM robot mouse	106
4.19	Prototype of an actuated head with cameras for the TUM Robot Mouse . . .	107
5.1	Examples of different neuron morphologies in the mouse brain	111
5.2	Illustration of the dorsal and ventral visual pathways	115
5.3	Example instantiation of the proposed MHNN architecture	123
5.4	Schematic illustration of lateral connectivity in MHNNs	127
5.5	NRP experiment for the collection of multimodal training data as required by deep multisensory neural maps	134
5.6	Workspace positions of all 102 000 samples in the multimodal data set D_{map} .	136
5.7	Visualization of selected samples from the multimodal data set D_{map}	137
5.8	Architectures of the component networks C_{vis} and C_{pro}	138
5.9	Input space mappings learned by the vision network C_{vis} and the proprioception network C_{pro} after supervised training on D_{map}^{vis} and D_{map}^{pro}	140
5.10	Overview of the four main machine learning paradigms	141
5.11	Visualization of the results of two selected runs of Algorithm 3 with losses \mathcal{L}_{vis}^{self} and \mathcal{L}_{pro}^{self} from Equations 5.24 and 5.25	145
5.12	Visualization of the learning progress in Example Run 1 from Figure 5.11 . .	146
5.13	Visualization of a sphere batch in the workspace and its latent space projection during an early training epoch	150
5.14	Visualization of the results of two selected runs of Algorithm 5 with the topographic loss \mathcal{L}^{top} from Equation 5.30	152

5.15	Progress of the training over selected epochs in Example Run 1 from Figure 5.14	153
5.16	Development of the neighborhood losses and separation losses during Example Run 1 from Figure 5.14	154
5.17	Visualization of the results of three selected runs of Algorithm 3 with different parameters for the topographic map losses \mathcal{L}_{vis}^{map} and \mathcal{L}_{pro}^{map} from Equations 5.31 and 5.32	156
5.18	Progress of the training over selected epochs in Example Run 3 from Figure 5.17	157
5.19	Development of the total loss during Example Run 1 and Example Run 3 from Figure 5.17	158
6.1	Stages in the development of the primary visual cortex of the mouse	163
6.2	Phases of functional development identified by Asada et al. [261] for research in cognitive developmental robotics	164
6.3	Schematic illustration of a training protocol	166
6.4	Example of neural population coding with tuning curves	170
6.5	Stimulus reconstruction from PPCs	170
6.6	Simulation of the PPC model in the NRP	173
6.7	Neural network architecture for a virtual neurorobotics experiment on audio-visual localization with PPCs	174
6.8	Simulation results for population codes and reconstructed stimulus estimates for vision, sound and multisensory integration	174
6.9	NRP experiment for staged reinforcement learning with training protocols	176
6.10	Visualization of the sampling space for target points	178
6.11	Increasing accuracy requirements in the target reaching experiment from Figure 6.9 over subsequent training stages	178
6.12	Schematic illustration of the PCA-loop in an MDP for reinforcement learning	180
6.13	Architectures of the actor network N_μ and the critic network N_q	186
6.14	Results from 100 training runs of Algorithm 6 with environment-based stages	188
6.15	Histogram of the target reaching success rates of the models trained in Figure 6.14	189
6.16	Results from 100 training runs of Algorithm 6 with robot-based stages	193
6.17	Results from 100 training runs of Algorithm 6 with robot-based stages and low positioning accuracy	194
6.18	Results from 100 training runs of Algorithm 6 with robot-based stages and both freeing and freezing of DoFs	195
6.19	Histogram of the target reaching success rates of the models trained in Figure 6.16	196
6.20	Histogram of the target reaching success rates of the models trained in Figure 6.17	196
6.21	Histogram of the target reaching success rates of the models trained in Figure 6.18	196
7.1	Modeling of body growth	203

List of Tables

2.1	Multiscale organization of the brain	26
2.2	Overview of selected large-scale brain simulations	47
2.3	Schematic diagrams of the neuromorphic chip designs SpiNNaker, BrainScaleS and Intel Loihi	51
3.1	The five steps in the development of a neurorobotics experiment	73
4.1	Comparison of the virtual and physical experiment setups for robot grasping	90
4.2	Selected implementations of the additional services defined in the extended NRP system architecture	91
6.1	A training protocol with environment-based stages	187
6.2	Evaluation of the models trained in Figure 6.14	188
6.3	A training protocol with robot-based stages and sequential freeing of DoFs .	192
6.4	A training protocol with robot-based stages and sequential freeing of DoFs with lower target reaching accuracy compared to Table 6.3	192
6.5	A training protocol with robot-based stages and non-sequential freeing of DoFs	192
6.6	Evaluation of the models trained in Figure 6.16	193
6.7	Evaluation of the models trained in Figure 6.17	194
6.8	Evaluation of the models trained in Figure 6.18	195
A.1	Parameters for the generation the multimodal data set D_{map} with Algorithm 2	205
A.2	Hyperparameters for Figure 5.9	206
A.3	Hyperparameters for Figures 5.11 and 5.12	206
A.4	Hyperparameters for Figures 5.14 and 5.15	207
A.5	Hyperparameters for Example Run 1 from Figure 5.17	207
A.6	Hyperparameters for Example Run 2 from Figure 5.17	208
A.7	Hyperparameters for Example Run 3 from Figure 5.17	208
A.8	Parameters of the LIF neurons in the neural population for multisensory integration from the network model shown in Figure 6.7	209
A.9	Hyperparameters for staged reinforcement learning with Algorithm 6	210

List of Algorithms

1	Learning Lateral Connections between Component Networks	129
2	Generation of the Multimodal Data Set D_{map}	136
3	Multimodal Self-Supervised Learning	143
4	Sphere Batching	149
5	Multimodal Unsupervised Learning with Sphere Batching	151
6	Staged DDPG Algorithm	184

Bibliography

- [1] Rodney Brooks. *Robots will invade our lives*. Ed. by TED Conferences. 2003. URL: https://www.ted.com/talks/rodney_brooks_robots_will_invade_our_lives (visited on 05/09/2021).
- [2] Richmond Lattimore. *The Iliad of Homer. Translated and with an Introduction by Richmond Lattimore*. 21st ed. Vol. P63. A Phoenix book. Chicago & London: University of Chicago Press, 1951. 527 pp. ISBN: 0226469409.
- [3] Dario Amodei. *AI and Compute*. 2018. URL: <https://openai.com/blog/ai-and-compute/> (visited on 02/26/2020).
- [4] Xue Fan and Henry Markram. "A Brief History of Simulation Neuroscience." In: *Frontiers in Neuroinformatics* 13 (2019), p. 32. ISSN: 1662-5196. DOI: 10.3389/fninf.2019.00032.
- [5] Alois Knoll and Marc-Oliver Gewaltig. "Neurorobotics: A strategic pillar of the Human Brain Project." In: *Brain-inspired intelligent robotics: The intersection of robotics and neuroscience*. Ed. by Jean Sanders and Jackie Oberst. Washington, DC: Science/AAAS, 2016, pp. 25–34.
- [6] Egidio Falotico et al. "Connecting Artificial Brains to Robots in a Comprehensive Simulation Framework: The Neurorobotics Platform." In: *Frontiers in Neuroinformatics* 11 (2017), p. 2. ISSN: 1662-5218. DOI: 10.3389/fninf.2017.00002.
- [7] Katrin Amunts et al. "The Human Brain Project: Creating a European Research Infrastructure to Decode the Human Brain." In: *Neuron* 92.3 (2016), pp. 574–581. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2016.10.046.
- [8] David Vernon. *Artificial Cognitive Systems. A Primer*. Cambridge, MA: The MIT Press, 2014. ISBN: 9780262028387.
- [9] Merriam-Webster. *Cognition*. Ed. by Merriam-Webster.com dictionary. URL: <https://www.merriam-webster.com/dictionary/cognition> (visited on 02/25/2020).
- [10] Rodney A. Brooks. "Intelligence without representation." In: *Artificial Intelligence* 47.1 (1991), pp. 139–159. ISSN: 00043702. DOI: 10.1016/0004-3702(91)90053-M.
- [11] Marvin Minsky. *The society of mind*. In collab. with Juliana Lee. 1st ed. A Touchstone Book. New York: Simon & Schuster, 1988. 339 pp. ISBN: 0671607405.
- [12] Hans P. Moravec. *Mind children. The future of robot and human intelligence*. Cambridge, Mass.: Harvard University Press, 1988. 214 pp. ISBN: 0674576187.
- [13] Murray Campbell et al. "Deep Blue." In: *Artificial Intelligence* 134.1 (2002), pp. 57–83. ISSN: 00043702. DOI: 10.1016/S0004-3702(01)00129-1.
- [14] David Silver et al. "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587 (2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961.
- [15] Noam Brown and Tuomas Sandholm. "Superhuman AI for multiplayer poker." In: *Science* 365.6456 (2019), p. 885. DOI: 10.1126/science.aay2400.

Bibliography

- [16] Brooke Chan. *OpenAI Five*. 2018. URL: <https://openai.com/blog/openai-five/> (visited on 02/26/2020).
- [17] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning.” In: *Nature* 575.7782 (2019), pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z.
- [18] David A. Ferrucci. “Introduction to “This is Watson.”” In: *IBM Journal of Research and Development* 56.3.4 (2012), 1:1–1:15. ISSN: 0018-8646. DOI: 10.1147/JRD.2012.2184356.
- [19] Cade Metz and Steve Lohr. “IBM Unveils System That ‘Debates’ With Humans.” In: *The New York Times* 2018 (June 19, 2018). URL: <https://www.nytimes.com/2018/06/18/technology/ibm-debater-artificial-intelligence.html> (visited on 02/26/2020).
- [20] Jun-Heui Cho. “Modelling of a Biologically Accurate Continuous Series of 3D Models of the Growing Human Body for Developmental Robotics.” Chair of Robotics, Artificial Intelligence and Real-Time Systems. Master’s Thesis. Garching: Technical University of Munich, 2017.
- [21] Rodney A. Brooks. “Elephants don’t play chess.” In: *Robotics and Autonomous Systems* 6.1 (1990), pp. 3–15. ISSN: 0921-8890. DOI: 10.1016/S0921-8890(05)80025-9.
- [22] John McCarthy et al. *A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence*. 1955. URL: <http://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html> (visited on 10/31/2018).
- [23] Stuart J. Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach*. Global Edition. 3rd ed. Prentice Hall Series in Artificial Intelligence. Pearson Education Limited, 2016. xviii, 1132. ISBN: 978-1-292-15396-4.
- [24] Nils J. Nilsson. *Shakey the Robot. Technical Note 323*. Ed. by AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025. 1984. (Visited on 09/06/2018).
- [25] Peter E. Hart et al. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. ISSN: 2168-2887. DOI: 10.1109/TSSC.1968.300136.
- [26] Richard E. Fikes and Nils J. Nilsson. “Strips: A new approach to the application of theorem proving to problem solving.” In: *Artificial Intelligence* 2.3 (1971), pp. 189–208. ISSN: 00043702. DOI: 10.1016/0004-3702(71)90010-5.
- [27] Dana S. Nau. “Current Trends in Automated Planning.” In: *AI Magazine* 28.4 (2007). DOI: 10.1609/aimag.v28i4.2067.
- [28] Allen Newell and Herbert A. Simon. “Computer Science as Empirical Inquiry: Symbols and Search.” In: *Commun. ACM* 19.3 (1976), pp. 113–126. ISSN: 0001-0782. DOI: 10.1145/360018.360022.
- [29] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. New York, NY: Springer, 2016. XX, 738. ISBN: 978-0-387-31073-2.
- [30] Y. Bengio et al. “Representation Learning: A Review and New Perspectives.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1798–1828. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2013.50.
- [31] Geoffrey E. Hinton et al. “A Fast Learning Algorithm for Deep Belief Nets.” In: *Neural Computation* 18.7 (2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527.
- [32] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity.” In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.
- [33] Jia Deng et al. *ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC 2012)*. 2012. URL: <http://image-net.org/challenges/LSVRC/2012/> (visited on 03/16/2020).

-
- [34] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge.” In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y.
- [35] Adriana Kovashka et al. “Crowdsourcing in Computer Vision.” In: *Foundations and Trends® in Computer Graphics and Vision* 10.3 (2016), pp. 177–243. ISSN: 1572-2740. DOI: 10.1561/06000000071.
- [36] Daisuke Wakabayashi. “Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam.” In: *The New York Times* (Mar. 19, 2018). URL: <https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html> (visited on 03/18/2020).
- [37] Niraj Chokshi. “Tesla Autopilot System Found Probably at Fault in 2018 Crash.” In: *The New York Times* (Feb. 25, 2020). URL: <https://www.nytimes.com/2020/02/25/business/tesla-autopilot-ntsb.html?searchResultPosition=1> (visited on 03/18/2020).
- [38] Christian Szegedy et al. *Intriguing properties of neural networks*. 2013. URL: <https://arxiv.org/pdf/1312.6199>.
- [39] Ian J. Goodfellow et al. *Explaining and Harnessing Adversarial Examples*. 2014. URL: <https://arxiv.org/pdf/1412.6572>.
- [40] Alexey Kurakin et al. *Adversarial examples in the physical world*. 2016. URL: <https://arxiv.org/pdf/1607.02533>.
- [41] Anish Athalye et al. “Synthesizing Robust Adversarial Examples.” In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 284–293. URL: <http://proceedings.mlr.press/v80/athalye18b.html>.
- [42] K. Eykholt et al. “Robust Physical-World Attacks on Deep Learning Visual Classification.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1625–1634. DOI: 10.1109/CVPR.2018.00175.
- [43] Adam Gleave et al. *Adversarial Policies: Attacking Deep Reinforcement Learning*. 2020. URL: <https://arxiv.org/pdf/1905.10615>.
- [44] Battista Biggio and Fabio Roli. “Wild patterns: Ten years after the rise of adversarial machine learning.” In: *Pattern Recognition* 84 (2018), pp. 317–331. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2018.07.023.
- [45] Wojciech Samek et al., eds. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-28954-6.
- [46] Chris Olah et al. “The Building Blocks of Interpretability.” In: *Distill* 3.3 (2018), e10. ISSN: 2476-0757. DOI: 10.23915/distill.00010.
- [47] W. Xiang et al. “Output Reachable Set Estimation and Verification for Multilayer Neural Networks.” In: *IEEE Transactions on Neural Networks and Learning Systems* 29.11 (2018), pp. 5777–5783. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2018.2808470.
- [48] Chih-Hong Cheng et al. “Maximum Resilience of Artificial Neural Networks.” In: *Automated Technology for Verification and Analysis* (Cham, 2017). Ed. by Deepak D’Souza and K. Narayan Kumar. Cham: Springer International Publishing, 2017, pp. 251–268. ISBN: 978-3-319-68167-2.
- [49] Timon Gehr et al. “AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation.” In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 3–18. DOI: 10.1109/SP.2018.00058.
- [50] Youcheng Sun et al. “Structural Test Coverage Criteria for Deep Neural Networks.” In: *ACM Trans. Embed. Comput. Syst.* 18.5s (2019), Article 94. ISSN: 1539-9087. DOI: 10.1145/3358233.

Bibliography

- [51] Robert M. French. “Catastrophic forgetting in connectionist networks.” In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135. ISSN: 1364-6613. DOI: 10.1016/S1364-6613(99)01294-2.
- [52] Daniel L. K. Yamins and James J. DiCarlo. “Using goal-driven deep learning models to understand sensory cortex.” In: *Nature Neuroscience* 19.3 (2016), pp. 356–365. ISSN: 1546-1726. DOI: 10.1038/nn.4244.
- [53] Tim C. Kietzmann et al. “Deep Neural Networks in Computational Neuroscience.” English. In: *Oxford Research Encyclopedia of Neuroscience* (2019). DOI: 10.1093/acrefore/9780190264086.013.46. (Visited on 03/24/2020).
- [54] Demis Hassabis et al. “Neuroscience-Inspired Artificial Intelligence.” In: *Neuron* 95.2 (2017), pp. 245–258. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2017.06.011.
- [55] Adam H. Marblestone et al. “Toward an Integration of Deep Learning and Neuroscience.” In: *Frontiers in Computational Neuroscience* 10 (2016), p. 94. ISSN: 1662-5188. DOI: 10.3389/fncom.2016.00094.
- [56] Blake A. Richards et al. “A deep learning framework for neuroscience.” In: *Nature Neuroscience* 22.11 (2019), pp. 1761–1770. ISSN: 1546-1726. DOI: 10.1038/s41593-019-0520-2.
- [57] Ian Goodfellow et al. *Deep Learning*. Cambridge, Massachusetts: MIT Press, 2016. 785 pp. ISBN: 9780262035613.
- [58] Henry Markram. “The Blue Brain Project.” In: *Nature Reviews Neuroscience* 7.2 (2006), pp. 153–160. ISSN: 1471-0048. DOI: 10.1038/nrn1848.
- [59] Henry Markram et al. “Reconstruction and Simulation of Neocortical Microcircuitry.” In: *Cell* 163.2 (2015), pp. 456–492. ISSN: 0092-8674. DOI: 10.1016/j.cell.2015.09.029.
- [60] Katrin Amunts et al. “The Human Brain Project—Synergy between neuroscience, computing, informatics, and brain-inspired technologies.” In: *PLOS Biology* 17.7 (2019), e3000344. DOI: 10.1371/journal.pbio.3000344.
- [61] Florian Walter et al. “Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks.” In: *Neural Networks* 72 (2015), pp. 152–167. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2015.07.004.
- [62] Mike Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning.” In: *IEEE Micro* 38.1 (2018), pp. 82–99. ISSN: 1937-4143. DOI: 10.1109/MM.2018.112130359.
- [63] Florian Walter et al. “Computation by Time.” In: *Neural Processing Letters* 44.1 (2016), pp. 103–124. ISSN: 1573-773X. DOI: 10.1007/s11063-015-9478-6.
- [64] Peter Blouw et al. “Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware.” In: *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*. Albany, NY, USA: Association for Computing Machinery, 2019, Article 1. ISBN: 9781450361231. DOI: 10.1145/3320288.3320304.
- [65] Elon Musk. “An Integrated Brain-Machine Interface Platform With Thousands of Channels.” In: *J Med Internet Res* 21.10 (2019), e16194. ISSN: 1438-8871. DOI: 10.2196/16194.
- [66] Bente Pakkenberg et al. “Aging and the human neocortex.” In: *Experimental Gerontology* 38.1 (2003), pp. 95–99. ISSN: 0531-5565. DOI: 10.1016/S0531-5565(02)00151-1.
- [67] Valentino Braitenberg. “Brain.” In: *Scholarpedia* 2.11 (2007), p. 2918. ISSN: 1941-6016. DOI: 10.4249/scholarpedia.2918. (Visited on 03/27/2020).
- [68] Frederico A.C. Azevedo et al. “Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain.” In: *Journal of Comparative Neurology* 513.5 (2009), pp. 532–541. DOI: 10.1002/cne.21974.
- [69] Ferris Jabr. *Does Thinking Really Hard Burn More Calories?* Scientific American. 2012. URL: <https://www.scientificamerican.com/article/thinking-hard-calories/> (visited on 03/27/2020).

-
- [70] Matthew F. Glasser et al. “A multi-modal parcellation of human cerebral cortex.” In: *Nature* 536.7615 (2016), pp. 171–178. ISSN: 1476-4687. DOI: 10.1038/nature18933.
- [71] Tegan McCaslin. *Transmitting fibers in the brain: Total length and distribution of lengths*. 2018. URL: <https://aiimpacts.org/transmitting-fibers-in-the-brain-total-length-and-distribution-of-lengths/> (visited on 03/27/2020).
- [72] Sanger Brown. “Responsibility in Crime from the Medical Standpoint.” In: *The Popular Science Monthly*. Ed. by William Jay Youmans. 46. New York, NY: D. Appleton and Company, 1895, pp. 154–164.
- [73] R. P. Feldman and James T. Goodrich. “The Edwin Smith Surgical Papyrus.” In: *Child’s Nervous System* 15.6 (1999), pp. 281–284. ISSN: 1433-0350. DOI: 10.1007/s003810050395.
- [74] Javier DeFelipe. “The dendritic spine story: an intriguing process of discovery.” eng. In: *Frontiers in neuroanatomy* 9 (2015), p. 14. ISSN: 1662-5129. DOI: 10.3389/fnana.2015.00014.
- [75] Bruce Edmonds et al. “Different Modelling Purposes.” In: *Journal of Artificial Societies and Social Simulation* 22.3 (2019), p. 6. ISSN: 1460-7425. DOI: 10.18564/jasss.3993.
- [76] Peter Dayan and Laurence F. Abbott. *Theoretical Neuroscience. Computational and Mathematical Modeling of Neural Systems*. First paperback ed. Computational neuroscience. Cambridge, Mass.: MIT Press, 2005. 460 pp. ISBN: 9780262541855.
- [77] Daniel Levenstein et al. *On the role of theory and modeling in neuroscience*. 2020. URL: <https://arxiv.org/pdf/2003.13825>.
- [78] David Marr. *Vision. A computational investigation into the human representation and processing of visual information*. Cambridge, Mass: MIT Press, 2010. 403 pp. ISBN: 9780262514620. URL: <http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10397658>.
- [79] David Daniel Cox and Thomas Dean. “Neural Networks and Neuroscience-Inspired Computer Vision.” In: *Current Biology* 24.18 (2014), R921–R929. ISSN: 0960-9822. DOI: 10.1016/j.cub.2014.08.026.
- [80] Korbinian Brodmann. *Vergleichende Lokalisationslehre der Großhirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Leipzig: Barth, 1909. X, 324 S.
- [81] American Psychological Association. *Cytoarchitecture*. APA Dictionary of Psychology. 2020. URL: <https://dictionary.apa.org/cytoarchitecture> (visited on 04/05/2020).
- [82] Katrin Amunts et al. “BigBrain: An Ultrahigh-Resolution 3D Human Brain Model.” In: *Science* 340.6139 (2013), p. 1472. DOI: 10.1126/science.1235381.
- [83] A. Ailamaki et al. *The Human Brain Project. A Report to the European Commission*. Ed. by Richard Walker. Lausanne, 2012.
- [84] Maya Shamir et al. “SnapShot: Timescales in Cell Biology.” In: *Cell* 164.6 (2016), 1302–1302.e1. ISSN: 0092-8674. DOI: 10.1016/j.cell.2016.02.058.
- [85] Henry Gray. *Anatomy of the human body*. 20th ed. Philadelphia: Lea & Febiger, 1918. URL: <http://worldcatlibraries.org/wcpa/oclc/610374762>.
- [86] David Vernon et al. “A Survey of Artificial Cognitive Systems: Implications for the Autonomous Development of Mental Capabilities in Computational Agents.” In: *IEEE Transactions on Evolutionary Computation* 11.2 (2007), pp. 151–180. ISSN: 1941-0026. DOI: 10.1109/TEVC.2006.890274.
- [87] Dimitris Pinotsis et al. “Neural masses and fields: modeling the dynamics of brain activity.” In: *Frontiers in Computational Neuroscience* 8 (2014), p. 149. ISSN: 1662-5188. DOI: 10.3389/fncom.2014.00149.
- [88] S. Coombes. “Waves, bumps, and patterns in neural field theories.” In: *Biological Cybernetics* 93.2 (2005), pp. 91–108. ISSN: 1432-0770. DOI: 10.1007/s00422-005-0574-y.

- [89] P. A. Robinson et al. “A Multiscale “Working Brain” Model.” In: *Validating Neuro-Computational Models of Neurological and Psychiatric Disorders*. Ed. by Basabdatta Sen Bhattacharya and Fahmida N. Chowdhury. Cham: Springer International Publishing, 2015, pp. 107–140. ISBN: 978-3-319-20037-8. doi: 10.1007/978-3-319-20037-8_5.
- [90] Yulia Sandamirskaya and Mikhail Burtsev. “NARLE: Neurocognitive architecture for the autonomous task recognition, learning, and execution.” In: *Biologically Inspired Cognitive Architectures 13* (2015), pp. 91–104. ISSN: 2212-683X. doi: 10.1016/j.bica.2015.06.007.
- [91] Gregor Schöner. “Dynamical Systems Approaches to Cognition.” In: *The Cambridge Handbook of Computational Psychology*. Ed. by Ron Sun. Cambridge Handbooks in Psychology. Cambridge: Cambridge University Press, 2008, pp. 101–126. ISBN: 9780521857413. doi: 10.1017/CBO9780511816772.007.
- [92] Andreas V. M. Herz et al. “Modeling Single-Neuron Dynamics and Computations: A Balance of Detail and Abstraction.” In: *Science* 314.5796 (2006), pp. 80–85. doi: 10.1126/science.1127240.
- [93] Christian Rössert et al. *Automated point-neuron simplification of data-driven microcircuit models*. 2017. URL: <https://arxiv.org/pdf/1604.00087> (visited on 04/24/2020).
- [94] Wulfram Gerstner. *Neuronal Dynamics. From Single Neurons To Networks And Models Of Cognition*. Cambridge: Cambridge University Press, 2014. 577 pp. ISBN: 978-1107635197. doi: 10.1017/CBO9781107447615.
- [95] David Ferster and Nelson Spruston. “Cracking the Neuronal Code.” In: *Science* 270.5237 (1995), p. 756. doi: 10.1126/science.270.5237.756.
- [96] Esther P. Gardner and Kenneth O. Johnson. “21: Sensory Coding.” In: *Principles of Neural Science*. Ed. by Eric R. Kandel et al. 5th ed. New York, Lisbon, and London: McGraw-Hill Medical, 2013. ISBN: 978-0071390118.
- [97] David H. Hubel and Torsten N. Wiesel. “Receptive fields of single neurones in the cat’s striate cortex.” In: *The Journal of Physiology* 148.3 (1959), pp. 574–591. ISSN: 00223751. doi: 10.1113/jphysiol.1959.sp006308.
- [98] David H. Hubel and Torsten N. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex.” eng. In: *The Journal of Physiology* 160 (1962), pp. 106–154. ISSN: 00223751. doi: 10.1113/jphysiol.1962.sp006837.
- [99] Torkel Hafting et al. “Microstructure of a spatial map in the entorhinal cortex.” In: *Nature* 436.7052 (2005), pp. 801–806. ISSN: 1476-4687. doi: 10.1038/nature03721.
- [100] Roger M. Enoka and Jacques Duchateau. “Rate Coding and the Control of Muscle Force.” In: *Cold Spring Harbor Perspectives in Medicine* 7.10 (2017). doi: 10.1101/cshperspect.a029702.
- [101] Simon Thorpe et al. “Speed of processing in the human visual system.” In: *Nature* 381.6582 (1996), pp. 520–522. ISSN: 1476-4687. doi: 10.1038/381520a0.
- [102] Rufin van Rullen and Simon J. Thorpe. “Rate Coding Versus Temporal Order Coding: What the Retinal Ganglion Cells Tell the Visual Cortex.” In: *Neural Computation* 13.6 (2001), pp. 1255–1283. ISSN: 0899-7667. doi: 10.1162/08997660152002852.
- [103] M. Abeles et al. “Spatiotemporal firing patterns in the frontal cortex of behaving monkeys.” In: *Journal of Neurophysiology* 70.4 (1993), pp. 1629–1638. ISSN: 0022-3077. doi: 10.1152/jn.1993.70.4.1629.
- [104] Emiliano Torre et al. “Synchronous Spike Patterns in Macaque Motor Cortex during an Instructed-Delay Reach-to-Grasp Task.” In: *The Journal of Neuroscience* 36.32 (2016), p. 8329. doi: 10.1523/JNEUROSCI.4375-15.2016.
- [105] Peter Uhlhaas et al. “Neural synchrony in cortical networks: history, concept and current status.” In: *Frontiers in Integrative Neuroscience* 3 (2009), p. 17. ISSN: 1662-5145. doi: 10.3389/neuro.07.017.2009.

-
- [106] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models.” In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(97)00011-7.
- [107] Hongkui Zeng and Joshua R. Sanes. “Neuronal cell-type classification: challenges, opportunities and the path forward.” eng. In: *Nature reviews. Neuroscience* 18.9 (2017), pp. 530–546. DOI: 10.1038/nrn.2017.85.
- [108] BruceBlaus. *Drawing of a Multipolar Neuron*. Ed. by Wikimedia Commons, Licensed under CC BY 3.0. 2013. URL: https://commons.wikimedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png (visited on 09/24/2019).
- [109] A. L. Hodgkin and A. F. Huxley. “Action Potentials Recorded from Inside a Nerve Fibre.” In: *Nature* 144.3651 (1939), pp. 710–711. ISSN: 1476-4687. DOI: 10.1038/144710a0.
- [110] A. L. Hodgkin and A. F. Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve.” In: *The Journal of Physiology* 117.4 (1952), pp. 500–544. ISSN: 00223751. DOI: 10.1113/jphysiol.1952.sp004764.
- [111] Louis Lapicque. “Recherches quantitatives sur l’excitation électrique des nerfs traitée comme une polarisation.” In: *Journal de Physiologie et de Pathologie Générale* 9 (1907), pp. 620–635.
- [112] L.F. Abbott. “Lapicque’s introduction of the integrate-and-fire model neuron (1907).” In: *Brain Research Bulletin* 50.5 (1999), pp. 303–304. ISSN: 0361-9230. DOI: 10.1016/S0361-9230(99)00161-6.
- [113] Louis Lapicque. “Quantitative investigations of electrical nerve excitation treated as polarization.” eng. In: *Biological Cybernetics* 97.5-6 (2007), pp. 341–349. ISSN: 1432-0770. DOI: 10.1007/s00422-007-0189-6.
- [114] Jakob Jordan et al. *NEST 2.18.0*. Zenodo, 2019. DOI: 10.5281/ZENODO.2605422.
- [115] Frank Rosenblatt. *The Perceptron - A Perceiving and Recognizing Automaton*. Cornell Aeronautical Laboratory, 1957.
- [116] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519.
- [117] Connor Shorten and Taghi M. Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning.” In: *Journal of Big Data* 6.1 (2019), p. 1106. DOI: 10.1186/s40537-019-0197-0.
- [118] Frank Rosenblatt. “Perceptron Simulation Experiments.” In: *Proceedings of the IRE* 48.3 (1960), pp. 301–309. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1960.287598.
- [119] John C. Hay et al. *Mark I Perceptron Operator’s Manual (Projekt PARA)*. Cornell Aeronautical Laboratory, Inc., 1960. (Visited on 10/16/2018).
- [120] Kevin P. Murphy. *Machine learning. A probabilistic perspective*. Adaptive Computation and Machine Learning Series. Cambridge and London: The MIT Press, 2012. xxix, 1071 Seiten. ISBN: 9780262018029.
- [121] Bernard Widrow. *An adaptive ‘ADALINE’ neuron using chemical ‘memistors’*. Stanford University, 1960.
- [122] Bernard Widrow. “Generalization and Information Storage in Networks of Adaline ‘Neurons’” In: *Self-Organizing Systems*. Symposium Proceedings. Ed. by M. C. Yovitz et al. Washington, DC: Spartan Books, 1962, pp. 435–461.
- [123] “NEW NAVY DEVICE LEARNS BY DOING.” In: *New York Times* 1958 (July 8, 1958), p. 25. URL: <https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html>.
- [124] “Electronic ‘Brain’ Teaches Itself.” In: *New York Times* 1958 (July 13, 1958), p. 9. URL: <https://www.nytimes.com/1958/07/13/archives/electronic-brain-teaches-itself.html>.

Bibliography

- [125] J. J. Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” eng. In: *Proceedings of the National Academy of Sciences of the United States of America* 79.8 (1982), pp. 2554–2558. ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554.
- [126] Geoffrey E. Hinton and Terrence J. Sejnowski. “Optimal Perceptual Inference.” In: *IEEE Conference on Computer Vision and Pattern Recognition* (Washington, D. C.). Ed. by Takeo Kanade and Dana Ballard. 1983.
- [127] Teuvo Kohonen. “Self-organized formation of topologically correct feature maps.” In: *Biological Cybernetics* 43.1 (1982), pp. 59–69. ISSN: 1432-0770. DOI: 10.1007/BF00337288.
- [128] Kurt Hornik et al. “Multilayer feedforward networks are universal approximators.” In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8.
- [129] Jürgen Schmidhuber. “Deep learning in neural networks: an overview.” eng. In: *Neural networks : the official journal of the International Neural Network Society* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.
- [130] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.” In: *Biological Cybernetics* 36.4 (1980), pp. 193–202. ISSN: 1432-0770. DOI: 10.1007/BF00344251.
- [131] Y. Lecun et al. “Backpropagation Applied to Handwritten Zip Code Recognition.” In: *Neural Computation* 1.4 (1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.
- [132] Wikipedia, ed. *Kernel (image processing)*. en. 2020. URL: [https://en.wikipedia.org/w/index.php?title=Kernel_\(image_processing\)&oldid=951378908](https://en.wikipedia.org/w/index.php?title=Kernel_(image_processing)&oldid=951378908) (visited on 05/02/2020).
- [133] Yann Lecun et al. “Deep learning.” In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.
- [134] Alex Krizhevsky et al. “ImageNet Classification with Deep Convolutional Neural Networks.” In: *Advances in Neural Information Processing Systems* 25. Ed. by F. Pereira et al. Curran Associates, Inc, 2012, pp. 1097–1105.
- [135] Martin Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning.” In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. OSDI’16*. USA: USENIX Association, 2016, pp. 265–283. ISBN: 9781931971331.
- [136] Microsoft. *ONNX Runtime*. 2020. URL: <https://microsoft.github.io/onnxruntime/> (visited on 05/08/2020).
- [137] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” In: *Advances in Neural Information Processing Systems* 32. Ed. by H. Wallach et al. Curran Associates, Inc, 2019, pp. 8026–8037.
- [138] Huawei Technologies. *MindSpore*. 2020. URL: <https://github.com/mindspore-ai/mindspore> (visited on 05/08/2020).
- [139] Baidu Research. *Baidu PaddlePaddle Releases 21 New Capabilities to Accelerate Industry-Grade Model Development*. 2019. URL: <http://research.baidu.com/Blog/index-view?id=126> (visited on 05/08/2020).
- [140] Sony. *Neural Network Libraries by Sony (NNAblabla)*. 2020. URL: <https://nnaplab.org/> (visited on 05/06/2020).
- [141] Norman P. Jouppi et al. “In-Datacenter Performance Analysis of a Tensor Processing Unit.” In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. Toronto, ON, Canada: Association for Computing Machinery, 2017, pp. 1–12. ISBN: 9781450348928. DOI: 10.1145/3079856.3080246.
- [142] Huawei Technologies. *Ascend 910*. 2020. URL: <https://e.huawei.com/us/products/cloud-computing-dc/atlas/ascend-910> (visited on 05/11/2020).

-
- [143] Intel Corporation. *Intel Distribution of OpenVino Toolkit*. en. 2020. URL: <https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html> (visited on 05/08/2020).
- [144] Zhe Jia et al. *Dissecting the Graphcore IPU Architecture via Microbenchmarking*. 2019. URL: <https://arxiv.org/pdf/1912.03413> (visited on 05/11/2020).
- [145] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory.” In: *Neural Computation* 9.8 (1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.
- [146] Romain Brette. *What is computational neuroscience? (I) Definitions and the data-driven approach*. 2012. URL: <http://romainbrette.fr/what-is-computational-neuroscience-i-definitions-and-the-data-driven-approach/> (visited on 05/12/2020).
- [147] Sridevi Polavaram and Giorgio Ascoli. “Neuroinformatics.” In: *Scholarpedia* 10.11 (2015), p. 1312. ISSN: 1941-6016. DOI: 10.4249/scholarpedia.1312.
- [148] James Bower and David Beeman. “GENESIS (simulation environment).” In: *Scholarpedia* 2.3 (2007), p. 1383. ISSN: 1941-6016. DOI: 10.4249/scholarpedia.1383.
- [149] M. L. Hines and N. T. Carnevale. “The NEURON Simulation Environment.” In: *Neural Computation* 9.6 (1997), pp. 1179–1209. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.6.1179.
- [150] Dan Goodman and Romain Brette. “Brian: a simulator for spiking neural networks in Python.” In: *Frontiers in Neuroinformatics* 2 (2008), p. 5. ISSN: 1662-5196. DOI: 10.3389/neuro.11.005.2008.
- [151] Peter Lynch. “The origins of computer weather prediction and climate modeling.” In: *Journal of Computational Physics* 227.7 (2008), pp. 3431–3444. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2007.02.034.
- [152] Florian Kramer, ed. *Passive Sicherheit von Kraftfahrzeugen: Biomechanik — Simulation — Sicherheit im Entwicklungsprozess*. Wiesbaden: Vieweg+Teubner, 2009. ISBN: 978-3-8348-9254-6.
- [153] Shy Genel et al. “Introducing the Illustris project: the evolution of galaxy populations across cosmic time.” In: *Monthly Notices of the Royal Astronomical Society* 445.1 (2014), pp. 175–200. ISSN: 0035-8711. DOI: 10.1093/mnras/stu1654.
- [154] R. D. Traub and R. K. Wong. “Cellular mechanism of neuronal synchronization in epilepsy.” In: *Science* 216.4547 (1982), p. 745. DOI: 10.1126/science.7079735.
- [155] J. C. Pearson et al. “Plasticity in the organization of adult cerebral cortical maps: a computer simulation based on neuronal group selection.” In: *The Journal of Neuroscience* 7.12 (1987), p. 4209. DOI: 10.1523/JNEUROSCI.07-12-04209.1987.
- [156] Roger D. Traub et al. “Large Scale Simulations of the Hippocampus.” In: *IEEE Engineering in Medicine and Biology Magazine* 7.4 (1988), pp. 31–38. ISSN: 1937-4186. DOI: 10.1109/51.20378.
- [157] L. H. Finkel and G. M. Edelman. “Integration of distributed cortical systems by reentry: a computer simulation of interactive functionally segregated visual areas.” In: *The Journal of Neuroscience* 9.9 (1989), p. 3188. DOI: 10.1523/JNEUROSCI.09-09-03188.1989.
- [158] E. D. Lumer et al. “Neural dynamics in a model of the thalamocortical system. I. Layers, loops and the emergence of fast synchronous rhythms.” In: *Cerebral Cortex* 7.3 (1997), pp. 207–227. ISSN: 1047-3211. DOI: 10.1093/cercor/7.3.207.
- [159] Javier F. Medina et al. “Timing Mechanisms in the Cerebellum: Testing Predictions of a Large-Scale Computer Simulation.” In: *The Journal of Neuroscience* 20.14 (2000), p. 5516. DOI: 10.1523/JNEUROSCI.20-14-05516.2000.
- [160] Eugene M. Izhikevich and Gerald M. Edelman. “Large-scale model of mammalian thalamocortical systems.” In: *Proceedings of the National Academy of Sciences* 105.9 (2008), p. 3593. DOI: 10.1073/pnas.0712231105.

- [161] Rajagopal Ananthanarayanan et al. “The Cat is out of the Bag: Cortical Simulations with 109 Neurons, 1013 Synapses.” In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. SC '09. New York, NY, USA: Association for Computing Machinery, 2009. ISBN: 9781605587448. DOI: 10.1145/1654059.1654124.
- [162] Chris Eliasmith et al. “A Large-Scale Model of the Functioning Brain.” In: *Science* 338.6111 (2012), p. 1202. DOI: 10.1126/science.1225266.
- [163] Tobias C. Potjans and Markus Diesmann. “The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model.” In: *Cerebral Cortex* 24.3 (2012), pp. 785–806. ISSN: 1047-3211. DOI: 10.1093/cercor/bhs358.
- [164] Robert Preissl et al. “Compass: A scalable simulator for an architecture for cognitive computing.” In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2012. 10 - 16 Nov. 2012, Salt Lake City, Utah. 2012 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (Salt Lake City, UT, Nov. 10, 2012–Nov. 16, 2012)*. Association for Computing Machinery et al. Piscataway, NJ: IEEE, 2012, pp. 1–11. ISBN: 978-1-4673-0805-2. DOI: 10.1109/SC.2012.34.
- [165] Stefano Casali et al. “Reconstruction and Simulation of a Scaffold Model of the Cerebellar Network.” In: *Frontiers in Neuroinformatics* 13 (2019), p. 37. ISSN: 1662-5196. DOI: 10.3389/fninf.2019.00037.
- [166] Ján Antolík et al. “A comprehensive data-driven model of cat primary visual cortex.” In: *bioRxiv* (2019), p. 416156. DOI: 10.1101/416156.
- [167] Yazan N. Billeh et al. “Systematic Integration of Structural and Functional Data into Multi-scale Models of Mouse Primary Visual Cortex.” In: *Neuron* 106.3 (2020), 388–403.e18. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2020.01.040.
- [168] Eugene M. Izhikevich. “Simple model of spiking neurons.” In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572. ISSN: 1941-0093. DOI: 10.1109/TNN.2003.820440.
- [169] Terrence C. Stewart and Chris Eliasmith. “Large-Scale Synthesis of Functional Spiking Neural Circuits.” In: *Proceedings of the IEEE* 102.5 (2014), pp. 881–898. ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2306061.
- [170] Paul A. Merolla et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface.” In: *Science* 345.6197 (2014), p. 668. DOI: 10.1126/science.1254642.
- [171] Marco Capogrosso et al. “A Computational Model for Epidural Electrical Stimulation of Spinal Sensorimotor Circuits.” In: *The Journal of Neuroscience* 33.49 (2013), p. 19326. DOI: 10.1523/JNEUROSCI.1688-13.2013.
- [172] E. de Schutter and J. M. Bower. “An active membrane model of the cerebellar Purkinje cell. I. Simulation of current clamps in slice.” In: *Journal of Neurophysiology* 71.1 (1994), pp. 375–400. ISSN: 0022-3077. DOI: 10.1152/jn.1994.71.1.375.
- [173] E. de Schutter and J. M. Bower. “An active membrane model of the cerebellar Purkinje cell II. Simulation of synaptic responses.” In: *Journal of Neurophysiology* 71.1 (1994), pp. 401–419. ISSN: 0022-3077. DOI: 10.1152/jn.1994.71.1.401.
- [174] Gopal P. Sarma et al. “OpenWorm: overview and recent advances in integrative biological simulation of *Caenorhabditis elegans*.” In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 373.1758 (2018), p. 20170382. DOI: 10.1098/rstb.2017.0382.
- [175] Padraig Gleeson et al. “c302: a multiscale framework for modelling the nervous system of *Caenorhabditis elegans*.” In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 373.1758 (2018), p. 20170379. DOI: 10.1098/rstb.2017.0379.
- [176] Carver Mead. “Neuromorphic Electronic Systems.” In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636. ISSN: 1558-2256. DOI: 10.1109/5.58356.

-
- [177] R. Douglas et al. “Neuromorphic Analogue VLSI.” In: *Annual Review of Neuroscience* 18.1 (1995), pp. 255–281. ISSN: 0147-006X. DOI: 10.1146/annurev.ne.18.030195.001351.
- [178] Steve Furber and Andrew Brown. “Biologically-Inspired Massively-Parallel Architectures – computing beyond a million processors.” In: *2009 Ninth International Conference on Application of Concurrency to System Design*. 2009, pp. 3–12. DOI: 10.1109/ACSD.2009.17.
- [179] J. Schemmel et al. “A wafer-scale neuromorphic hardware system for large-scale neural modeling.” In: *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2010, pp. 1947–1950. DOI: 10.1109/ISCAS.2010.5536970.
- [180] Steve B. Furber et al. “The SpiNNaker Project.” In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665. ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2304638.
- [181] S. B. Furber et al. “Overview of the SpiNNaker System Architecture.” In: *IEEE Transactions on Computers* 62.12 (2013), pp. 2454–2467. ISSN: 1557-9956. DOI: 10.1109/TC.2012.142.
- [182] Andrew P. Davison et al. *HBP Neuromorphic Computing Platform Guidebook*. 2020. URL: <https://flagship.kip.uni-heidelberg.de/jss/FileExchange/HBPNeuromorphicComputingPlatformGuidebook.pdf?fileID=1504&s=qqdXDg6HuX3&uID=65> (visited on 08/15/2020).
- [183] Sacha J. van Albada et al. “Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model.” In: *Frontiers in Neuroscience* 12 (2018), p. 291. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00291.
- [184] Johannes Fierens et al. “Realizing biological spiking network models in a configurable wafer-scale hardware system.” In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008, pp. 969–976. DOI: 10.1109/IJCNN.2008.4633916.
- [185] Johannes Schemmel et al. “Wafer-scale integration of analog neural networks.” In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008, pp. 431–438. DOI: 10.1109/IJCNN.2008.4633828.
- [186] Romain Brette and Wulfram Gerstner. “Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity.” In: *Journal of Neurophysiology* 94.5 (2005), pp. 3637–3642. ISSN: 0022-3077. DOI: 10.1152/jn.00686.2005.
- [187] Andreas Grübl et al. “Verification and Design Methods for the BrainScaleS Neuromorphic Hardware System.” In: *Journal of Signal Processing Systems* (2020). ISSN: 1939-8115. DOI: 10.1007/s11265-020-01558-7.
- [188] Chetan Singh Thakur et al. “Large-Scale Neuromorphic Spiking Array Processors: A Quest to Mimic the Brain.” In: *Frontiers in Neuroscience* 12 (2018), p. 891. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00891.
- [189] Mike Davies. “Benchmarks for progress in neuromorphic computing.” In: *Nature Machine Intelligence* 1.9 (2019), pp. 386–388. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0097-1.
- [190] James C. Knight and Thomas Nowotny. “GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model.” In: *Frontiers in Neuroscience* 12 (2018), p. 941. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00941.
- [191] Christian Mayr et al. *SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning*. 2019. URL: <https://arxiv.org/pdf/1911.02385> (visited on 08/18/2020).
- [192] Johannes Weis et al. *Inference with Artificial Neural Networks on Analog Neuromorphic Hardware*. 2020. URL: <https://arxiv.org/pdf/2006.13177> (visited on 08/18/2020).
- [193] Intel Corporation. *Taking Neuromorphic Computing to the Next Level with Loihi 2. Technology Brief*. 2021. URL: <https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf> (visited on 01/09/2022).

Bibliography

- [194] Jeffrey L. Krichmar. “Neurorobotics—A Thriving Community and a Promising Pathway Toward Intelligent Cognitive Robots.” In: *Frontiers in Neurobotics* 12 (2018), p. 42. ISSN: 1662-5218. DOI: 10.3389/fnbot.2018.00042.
- [195] W. Grey Walter. “An Imitation Of Life.” In: *Scientific American* 182.5 (1950), pp. 42–45. ISSN: 00368733.
- [196] W. Grey Walter. “A Machine That Learns.” In: *Scientific American* 185.2 (1951), pp. 60–64. ISSN: 00368733.
- [197] George N. Reeke et al. “Synthetic Neural Modeling: The “Darwin” Series of Recognition Automata.” In: *Proceedings of the IEEE* 78.9 (1990), pp. 1498–1530. ISSN: 1558-2256. DOI: 10.1109/5.58327.
- [198] Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think. A new view of intelligence*. A Bradford book. Cambridge, Mass: MIT Press, 2007. XXIV, 394 S. ISBN: 0262281554.
- [199] Norbert Wiener. *Cybernetics: or control and communication in the animal and the machine*. 2nd ed. Cambridge, Mass.: MIT Press, 1985. 212 pp. ISBN: 0-262-23007-0.
- [200] Dirk Kerzel. “Perception in Vision.” In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN: 978-3-540-29678-2.
- [201] Rolf Pfeifer et al. “Morphological computation for adaptive behavior and cognition.” In: *International Congress Series* 1291 (2006), pp. 22–29. ISSN: 0531-5131. DOI: 10.1016/j.ics.2005.12.080.
- [202] Vincent C. Müller and Matej Hoffmann. “What Is Morphological Computation? On How the Body Contributes to Cognition and Control.” In: *Artificial Life* 23.1 (2017), pp. 1–24. ISSN: 1064-5462. DOI: 10.1162/ARTL_a_00219.
- [203] Helmut Hauser et al. “Towards a theoretical foundation for morphological computation with compliant bodies.” In: *Biological Cybernetics* 105.5 (2011), pp. 355–370. ISSN: 1432-0770. DOI: 10.1007/s00422-012-0471-0.
- [204] Mantas Lukoševičius and Herbert Jaeger. “Reservoir computing approaches to recurrent neural network training.” In: *Computer Science Review* 3.3 (2009), pp. 127–149. ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2009.03.005.
- [205] Herbert Jaeger. “The “echo state” approach to analysing and training recurrent neural networks – with an erratum note.” In: *German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), p. 13.
- [206] Wolfgang Maass et al. “Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations.” In: *Neural Computation* 14.11 (2002), pp. 2531–2560. ISSN: 0899-7667. DOI: 10.1162/089976602760407955.
- [207] Gouhei Tanaka et al. “Recent advances in physical reservoir computing: A review.” In: *Neural Networks* 115 (2019), pp. 100–123. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.03.005.
- [208] Encyclopædia Britannica. “Human Ear.” In: *Britannica Academic*. 2020. (Visited on 10/06/2020).
- [209] The Library of Congress. *What is the strongest muscle in the human body?* eng. 2020. URL: <https://www.loc.gov/everyday-mysteries/item/what-is-the-strongest-muscle-in-the-human-body> (visited on 10/05/2020).
- [210] Scott L. Delp et al. “OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement.” In: *IEEE Transactions on Biomedical Engineering* 54.11 (2007), pp. 1940–1950. ISSN: 1558-2531. DOI: 10.1109/TBME.2007.901024.
- [211] Konstantinos Dalamagkidis et al. *MYROBOTICS. A framework for musculoskeletal robot development*. en. European Commission Publication Office CORDIS. 2012. URL: <https://cordis.europa.eu/project/id/288219> (visited on 10/08/2020).
- [212] Francisco J. Varela et al. *The embodied mind. Cognitive science and human experience*. Cambridge, Mass., 1991. xx, 308 s. ; ISBN: 9780262220422.

-
- [213] F. G. Varela et al. "Autopoiesis: The organization of living systems, its characterization and a model." In: *Biosystems* 5.4 (1974), pp. 187–196. ISSN: 0303-2647. DOI: 10.1016/0303-2647(74)90031-8.
- [214] Robert A. Wilson and Lucia Foglia. "Embodied Cognition." In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2017. Metaphysics Research Lab, Stanford University, 2017. (Visited on 10/13/2020).
- [215] Humberto R. Maturana and Francisco J. Varela. *The Tree of Knowledge. The Biological Roots of Human Understanding*. 1st ed. New Science Library. Boston: Shambala Publications, 1987. 263 pp. ISBN: 0877733732.
- [216] Jakob von Uexküll. *Umwelt und Innenwelt der Tiere*. Zweite Vermehrte und Verbesserte Auflage. Berlin, Heidelberg and s.l.: Springer Berlin Heidelberg, 1921. 226 pp. ISBN: 978-3-662-22877-7. DOI: 10.1007/978-3-662-24819-5.
- [217] Eleanor J. Gibson et al. "Affordances." In: *The MIT Encyclopedia of the Cognitive Sciences*. Ed. by Robert Andrew Wilson and Frank C. Keil. Cambridge, Mass.: MIT Press, 2001. ISBN: 9780262731447.
- [218] Joseph L. Jones. "Robots at the Tipping Point. The Road to the iRobot Roomba." In: *IEEE Robotics Automation Magazine* 13.1 (2006), pp. 76–78. ISSN: 1558-223X. DOI: 10.1109/MRA.2006.1598056.
- [219] Max Lungarella et al. "Methods for quantifying the informational structure of sensory and motor data." In: *Neuroinformatics* 3.3 (2005), pp. 243–262. ISSN: 1559-0089. DOI: 10.1385/NI:3:3:243.
- [220] Max Lungarella and Olaf Sporns. "Information Self-Structuring: Key Principle for Learning and Development." In: *Proceedings. The 4th International Conference on Development and Learning, 2005*. 2005, pp. 25–30. DOI: 10.1109/DEVLRN.2005.1490938.
- [221] Max Lungarella and Olaf Sporns. "Mapping Information Flow in Sensorimotor Networks." In: *PLoS Computational Biology* 2.10 (2006), e144. DOI: 10.1371/journal.pcbi.0020144.
- [222] Andras J. Pellionisz. "Sensorimotor Operations: A Ground for the Co-Evolution Of Brain Theory with Neurobotics and Neurocomputers." In: *IEEE First International Conference on Neural Networks* (San Diego, California, June 21–24, 1987). 4. 1987, pp. 593–600.
- [223] Panagiotis Artemiadis. *Neuro-robotics. From brain machine interfaces to rehabilitation robotics*. Vol. 2. Trends in Augmentation of Human Performance. Dordrecht: Springer, 2014. 448 pp. ISBN: 978-94-017-8931-8. DOI: 10.1007/978-94-017-8932-5.
- [224] Samir Menon et al. "Controlling Articulated Robots in Task-Space with Spiking Silicon Neurons." In: *5th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechanics*. 2014, pp. 181–186. DOI: 10.1109/BIOROB.2014.6913773.
- [225] Oussama Khatib. "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation." In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53. ISSN: 2374-8710. DOI: 10.1109/JRA.1987.1087068.
- [226] Ben Varkey Benjamin et al. "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations." In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716. ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2313565.
- [227] Alexandros Bouganis and Murray Shanahan. "Training a Spiking Neural Network to Control a 4-DoF Robotic Arm based on Spike Timing-Dependent Plasticity." In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. 2010, pp. 1–8. DOI: 10.1109/IJCNN.2010.5596525.
- [228] Sen Song et al. "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity." In: *Nature Neuroscience* 3.9 (2000), pp. 919–926. ISSN: 1546-1726. DOI: 10.1038/78829.
- [229] Filip Ponulak and John Hopfield. "Rapid, parallel path planning by propagating wavefronts of spiking neural activity." In: *Frontiers in Computational Neuroscience* 7 (2013), p. 98. ISSN: 1662-5188. DOI: 10.3389/fncom.2013.00098.

- [230] Howie M. Choset. *Principles of Robot Motion. Theory, Algorithms, and Implementations*. Intelligent Robotics and Autonomous Agents. Cambridge, Mass: MIT Press, 2005. 630 pp. ISBN: 9780262033275.
- [231] Valentino Braitenberg. *Vehicles. Experiments in Synthetic Psychology*. 9th ed. Bradford Book. Cambridge, Mass.: MIT Press, 2004. 168 pp. ISBN: 9780262521123.
- [232] Dario Floreano and Claudio Mattiussi. "Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots." In: *Evolutionary Robotics. From Intelligent Robotics to Artificial Life* (Berlin, Heidelberg, 2001). Ed. by Takashi Gomi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 38–61. ISBN: 978-3-540-45502-8.
- [233] Ezequiel Di Paolo. "Spike-Timing Dependent Plasticity for Evolved Robots." en. In: *Adaptive Behavior* 10.3-4 (2002), pp. 243–263. ISSN: 1059-7123.
- [234] Răzvan V. Florian. "Spiking Neural Controllers for Pushing Objects Around." English. In: *From Animals to Animals 9*. Ed. by Stefano Nolfi et al. Vol. 4095. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 570–581. ISBN: 978-3-540-38608-7. DOI: 10.1007/11840541_47.
- [235] Xiuqing Wang et al. "A behavior controller based on spiking neural networks for mobile robots." en. In: *Neurocomputing* 71.4–6 (2008), pp. 655–666. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2007.08.025.
- [236] Paolo Arena et al. "Learning Anticipation via Spiking Networks: Application to Navigation Control." en. In: *IEEE Transactions on Neural Networks* 20.2 (2009), pp. 202–216. DOI: 10.1109/TNN.2008.2005134.
- [237] Santiago Brandi et al. "Learning of Motor Sequences Based on a Computational Model of the Cerebellum." English. In: *Biomimetic and Biohybrid Systems*. Ed. by NathanF Lepora et al. Vol. 8064. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 356–358. ISBN: 978-3-642-39801-8. DOI: 10.1007/978-3-642-39802-5_33.
- [238] André Cyr et al. "Operant Conditioning: A Minimal Components Requirement in Artificial Spiking Neurons Designed for Bio-Inspired Robot's Controller." en. In: *Frontiers in Neurobotics* 8 (2014). ISSN: 1662-5218. DOI: 10.3389/fnbot.2014.00021.
- [239] Hiroyuki Miyamoto et al. "Feedback-error-learning neural network for trajectory control of a robotic manipulator." In: *Neural Networks* 1.3 (1988), pp. 251–265. ISSN: 0893-6080. DOI: 10.1016/0893-6080(88)90030-5.
- [240] G. M. Edelman et al. "Synthetic neural modeling applied to a real-world artifact." In: *Proceedings of the National Academy of Sciences* 89.15 (1992), p. 7267. DOI: 10.1073/pnas.89.15.7267.
- [241] Nikolaus Almásson et al. "Behavioral constraints in the development of neuronal properties: a cortical model embedded in a real-world device." In: *Cerebral Cortex* 8.4 (1998), pp. 346–361. ISSN: 1047-3211. DOI: 10.1093/cercor/8.4.346.
- [242] Jason G. Fleischer et al. "Retrospective and prospective responses arising in a modeled hippocampus during maze navigation by a brain-based device." In: *Proceedings of the National Academy of Sciences* 104.9 (2007), p. 3556. DOI: 10.1073/pnas.0611571104.
- [243] B. R. Cox and J. L. Krichmar. "Neuromodulation as a Robot Controller as a Robot Controller. A Brain-Inspired Strategy for Controlling Autonomous Robots." In: *IEEE Robotics & Automation Magazine* 16.3 (2009), pp. 72–80. DOI: 10.1109/MRA.2009.933628.
- [244] Dimitri Probst et al. "Liquid Computing in a Simplified Model of Cortical Layer IV: Learning to Balance a Ball." In: *Artificial Neural Networks and Machine Learning – ICANN 2012* (Berlin, Heidelberg, 2012). Ed. by Alessandro E. P. Villa et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 209–216. ISBN: 978-3-642-33269-2.
- [245] Jun Tani. "Self-Organization and Compositionality in Cognitive Brains: A Neurobotics Study." en. In: *Proceedings of the IEEE* 102.4 (2014), pp. 586–605. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2308604.

-
- [246] Jörg Conradt et al. “Event-based neural computing on an autonomous mobile platform.” en. In: *IEEE International Conference on Robotics and Automation (ICRA)* (Hong Kong, China). 2014, pp. 2862–2867. doi: 10.1109/ICRA.2014.6907270.
- [247] Jörg Conradt et al. “Trainable sensorimotor mapping in a neuromorphic robot. Emerging Spatial Competences: From Machine Perception to Sensorimotor Intelligence.” en. In: *Robotics and Autonomous Systems* 71 (2015), pp. 60–68. issn: 0921-8890. doi: 10.1016/j.robot.2014.11.004.
- [248] Auke Jan Ijspeert et al. “From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model.” en. In: *Science* 315.5817 (2007), pp. 1416–1420. doi: 10.1126/science.1138353.
- [249] Auke Jan Ijspeert. “Central pattern generators for locomotion control in animals and robots: A review.” en. In: *Neural Networks* 21.4 (2008), pp. 642–653. issn: 0893-6080. doi: 10.1016/j.neunet.2008.03.014.
- [250] Auke Jan Ijspeert. “A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander.” English. In: *Biological Cybernetics* 84.5 (2001), pp. 331–348. issn: 1432-0770. doi: 10.1007/s004220000211.
- [251] Ludovic Righetti and Auke Jan Ijspeert. “Design methodologies for central pattern generators: an application to crawling humanoids.” en. In: *Proceedings of Robotics: Science and Systems*. Philadelphia, USA, 2006.
- [252] Theresa J. Klein and M. Anthony Lewis. “A physical model of sensorimotor interactions during locomotion.” In: *Journal of Neural Engineering* 9.4 (2012), p. 046011. issn: 1741-2552. doi: 10.1088/1741-2560/9/4/046011.
- [253] Yoshihiko Nakamura et al. “Somatosensory Computation for Man–Machine Interface From Motion-Capture Data and Musculoskeletal Human Model.” en. In: *IEEE Transactions on Robotics* 21.1 (2005), pp. 58–66. doi: 10.1109/TRO.2004.833798.
- [254] Akihiko Murai et al. “Modeling and Identifying the Somatic Reflex Network of the Human Neuromuscular System.” en. In: *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*. 2007, pp. 2717–2721. doi: 10.1109/IEMBS.2007.4352890.
- [255] Manish Sreenivasa et al. “Modeling and identification of the human arm stretch reflex using a realistic spiking neural network and musculoskeletal model.” en. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, pp. 329–334. doi: 10.1109/IROS.2013.6696372.
- [256] Dario Floreano et al. “Active Vision and Receptive Field Development in Evolutionary Robots.” In: *Evolutionary Computation* 13.4 (2005), pp. 527–544. issn: 1063-6560. doi: 10.1162/106365605774666912.
- [257] Mototaka Suzuki et al. “Constraints on body movement during visual development affect behavior of evolutionary robots.” In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*. Vol. 5. 2005, 2778–2783 vol. 5. doi: 10.1109/IJCNN.2005.1556365.
- [258] Mototaka Suzuki et al. “The contribution of active body movement to visual development in evolutionary robots.” In: *Neural Networks* 18.5 (2005), pp. 656–665. issn: 0893-6080. doi: 10.1016/j.neunet.2005.06.043.
- [259] Ralf Der and Georg Martius. “Novel plasticity rule can explain the development of sensorimotor intelligence.” In: *Proceedings of the National Academy of Sciences* 112.45 (2015), E6224. doi: 10.1073/pnas.1508400112.
- [260] Georg Martius et al. “Compliant control for soft robots: Emergent behavior of a tendon driven anthropomorphic arm.” In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2016, pp. 767–773. doi: 10.1109/IROS.2016.7759138.

- [261] Minoru Asada et al. “Cognitive Developmental Robotics. A Survey.” In: *IEEE Transactions on Autonomous Mental Development* 1.1 (2009), pp. 12–34. ISSN: 1943-0604. DOI: 10.1109/TAMD.2009.2021702.
- [262] Yasuo Kuniyoshi and Shinji Sangawa. “Early motor development from partially ordered neural-body dynamics: experiments with a cortico-spinal-musculo-skeletal model.” English. In: *Biological Cybernetics* 95.6 (2006), pp. 589–605. ISSN: 1432-0770. DOI: 10.1007/s00422-006-0127-z.
- [263] Hiroki Mori and Yasuo Kuniyoshi. “A human fetus development simulation: Self-organization of behaviors through tactile sensation.” en. In: *IEEE 9th International Conference on Development and Learning (ICDL)*. 2010, pp. 82–87. DOI: 10.1109/DEVLRN.2010.5578860.
- [264] Yasunori Yamada et al. “Impacts of environment, nervous system and movements of preterms on body map development: Fetus simulation with spiking neural network.” In: *2013 IEEE Third Joint International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. 2013, pp. 1–7. DOI: 10.1109/DevLrn.2013.6652548.
- [265] Yasunori Yamada et al. “An Embodied Brain Model of the Human Foetus.” In: *Scientific Reports* 6.1 (2016), p. 27893. ISSN: 2045-2322. DOI: 10.1038/srep27893.
- [266] Yasunori Yamada et al. “A Fetus and Infant Developmental Scenario: Self-organization of Goal-directed Behaviors Based on Sensory Constraints.” In: *Modeling cognitive development in robotic systems. Proceedings of the Tenth International Conference on Epigenetic Robotics*. Ed. by Birger Johansson et al. Lund University Cognitive Studies 149. Örenäs, Glumslöv, Sweden, 2010. ISBN: 9789197738095.
- [267] Yasunori Yamada and Yasuo Kuniyoshi. “Embodiment guides motor and spinal circuit development in vertebrate embryo and fetus.” In: *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. 2012, pp. 1–6. DOI: 10.1109/DevLrn.2012.6400578.
- [268] Yukie Nagai et al. “Emergence of Mirror Neuron System: Immature vision leads to self-other correspondence.” In: *2011 IEEE International Conference on Development and Learning (ICDL)*. Vol. 2. 2011, pp. 1–6. DOI: 10.1109/DEVLRN.2011.6037335.
- [269] Michael A. Arbib et al. “Neurorobotics: From Vision to Action.” In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1453–1480. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_63.
- [270] Jeff Krichmar. “Neurorobotics.” In: *Scholarpedia* 3.3 (2008), p. 1365. ISSN: 1941-6016. DOI: 10.4249/scholarpedia.1365.
- [271] Dario Floreano et al. “Robotics and Neuroscience.” In: *Current Biology* 24.18 (2014), R910–R920. ISSN: 0960-9822. DOI: 10.1016/j.cub.2014.07.058.
- [272] Zhenshan Bing et al. “A Survey of Robotics Control Based on Learning-Inspired Spiking Neural Networks.” In: *Frontiers in Neurobotics* 12 (2018), p. 35. ISSN: 1662-5218. DOI: 10.3389/fnbot.2018.00035.
- [273] Frontiers Media S.A. *Frontiers in Neurobotics*. 2020. URL: <https://www.frontiersin.org/journals/neurobotics> (visited on 11/01/2020).
- [274] IEEE RAS Technical Committee on Neuro-Robotics Systems. 2020. URL: <http://www.ieee-nrs.org/> (visited on 11/01/2020).
- [275] Frederic Kaplan. “Neurorobotics: an experimental science of embodiment.” In: *Frontiers in Neuroscience* 2 (2008), p. 23. ISSN: 1662-453X. DOI: 10.3389/neuro.01.023.2008.
- [276] Norihiro Maruyama et al. “Designing a Robotic Platform Controlled by Cultured Neural Cells.” In: *Artificial Life Conference Proceedings* 26 (2014), pp. 769–770. DOI: 10.1162/978-0-262-32621-6-ch124.

-
- [277] Atsushi Masumori et al. "Emergence of Sense-Making Behavior by the Stimulus Avoidance Principle: Experiments on a Robot Behavior Controlled by Cultured Neuronal Cells." In: *Artificial Life Conference Proceedings* 27 (2015), pp. 373–380. doi: 10.1162/978-0-262-33027-5-ch067.
- [278] Harry A. Pierson and Michael S. Gashler. "Deep learning in robotics: a review of recent research." In: *Advanced Robotics* 31.16 (2017), pp. 821–835. issn: 0169-1864. doi: 10.1080/01691864.2017.1365009.
- [279] André Cyr et al. "AI-SIMCOG: a simulator for spiking neurons and multiple animats' behaviours." In: *Neural Computing and Applications* 18.5 (2009), pp. 431–446. issn: 1433-3058. doi: 10.1007/s00521-009-0254-2.
- [280] Marcello Mulas et al. "A simulated neuro-robotic environment for bi-directional closed-loop experiments." English. In: *Paladyn, Journal of Behavioral Robotics* 1.3 (2010), pp. 179–186. doi: 10.2478/s13230-011-0004-x.
- [281] Zachary Tosi and Jeffrey Yoshimi. "Simbrain 3.0: A flexible, visually-oriented neural network simulator." In: *Neural Networks* 83 (2016), pp. 1–10. issn: 0893-6080. doi: 10.1016/j.neunet.2016.07.005.
- [282] Cristian Jimenez-Romero and Jeffrey Johnson. "SpikingLab: modelling agents controlled by Spiking Neural Networks in Netlogo." In: *Neural Computing and Applications* 28.1 (2017), pp. 755–764. issn: 1433-3058. doi: 10.1007/s00521-016-2398-1.
- [283] David Gamez et al. "Two Simulation Tools for Biologically Inspired Virtual Robotics." In: *Proceedings of the 5th Chapter Conference on Advances in Cybernetic Systems* (Sheffield, United Kingdom). 2006, pp. 85–90.
- [284] David Gamez et al. "iSpike: a spiking neural interface for the iCub robot." eng. In: *Bioinspiration & Biomimetics* 7.2 (2012), p. 025008. doi: 10.1088/1748-3182/7/2/025008.
- [285] David Cofer et al. "AnimatLab: A 3D graphics environment for neuromechanical simulations." In: *Journal of Neuroscience Methods* 187.2 (2010), pp. 280–288. issn: 0165-0270. doi: 10.1016/j.jneumeth.2010.01.005.
- [286] Philipp Weidel et al. "Closed Loop Interactions between Spiking Neural Network and Robotic Simulators Based on MUSIC and ROS." In: *Frontiers in Neuroinformatics* 10 (2016), p. 31. issn: 1662-5196. doi: 10.3389/fninf.2016.00031.
- [287] Thomas Voegtlin. "CLONES : a closed-loop simulation framework for body, muscles and neurons." In: *BMC Neuroscience* 12.1 (2011), P363. issn: 1471-2202. doi: 10.1186/1471-2202-12-S1-P363.
- [288] Jakob Jordan et al. "A Closed-Loop Toolchain for Neural Network Simulations of Learning Autonomous Agents." In: *Frontiers in Computational Neuroscience* 13 (2019), p. 46. issn: 1662-5188. doi: 10.3389/fncom.2019.00046.
- [289] Andrew Davison et al. "PyNN: a common interface for neuronal network simulators." In: *Frontiers in Neuroinformatics* 2 (2009), p. 11. issn: 1662-5196. doi: 10.3389/neuro.11.011.2008.
- [290] Marc-Oliver Gewaltig and Markus Diesmann. "NEST (NEural Simulation Tool)." In: *Scholarpedia* 2.4 (2007), p. 1430. issn: 1941-6016. doi: 10.4249/scholarpedia.1430.
- [291] Trevor Bekolay et al. "Nengo: a Python tool for building large-scale functional brain models." In: *Frontiers in Neuroinformatics* 7 (2014), p. 48. issn: 1662-5196. doi: 10.3389/fninf.2013.00048.
- [292] Nathan Koenig and Andrew Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator." In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No. 04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. doi: 10.1109/IROS.2004.1389727.
- [293] Morgan Quigley et al. "ROS: an open-source Robot Operating System." In: *ICRA 2009 Workshop on Open Source Software in Robotics*. IEEE International Conference on Robotics and Automation. Ed. by Hirohisa Hirukawa and Alois Knoll. 2009. url: <https://ci.nii.ac.jp/naid/20001086858/en/>.

- [294] Georg Hinkel et al. “A Domain-Specific Language (DSL) for Integrating Neuronal Networks in Robot Control.” In: *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-Based Software-Engineering*. MORSE/VAO '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 9–15. ISBN: 9781450336147. DOI: 10.1145/2802059.2802060.
- [295] Anna Letizia Allegra Mascaro et al. “Experimental and Computational Study on Motor Control and Recovery After Stroke: Toward a Constructive Loop Between Experimental and Virtual Embodied Neuroscience.” In: *Frontiers in Systems Neuroscience* 14 (2020), p. 31. ISSN: 1662-5137. DOI: 10.3389/fnsys.2020.00031.
- [296] Christoph von der Malsburg. “Vorbild Gehirn – Randbedingungen für eine kognitive Architektur.” In: *Cognitive Computing: Theorie, Technik und Praxis*. Ed. by Edy Portmann and Sara D’Onofrio. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, pp. 3–30. ISBN: 978-3-658-27941-7. DOI: 10.1007/978-3-658-27941-7_1.
- [297] Sergey Levine et al. “Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection.” In: *The International Journal of Robotics Research* 37.4-5 (2017), pp. 421–436. ISSN: 0278-3649. DOI: 10.1177/0278364917710318.
- [298] Florian Walter et al. “Massively Parallel Robot Simulations with the HBP Neurorobotics Platform.” In: *Embodied AI Workshop*. Conference on Computer Vision and Pattern Recognition 2021 (CVPR). 2021.
- [299] Domenico Prattichizzo and Jeffrey C. Trinkle. “Grasping.” In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 671–700. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_29.
- [300] Jeffrey Mahler et al. “Dex-Net 1.0: A cloud-based network of 3D objects for robust grasp planning using a Multi-Armed Bandit model with correlated rewards.” In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1957–1964. DOI: 10.1109/ICRA.2016.7487342.
- [301] Sergey Levine et al. *Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection*. 2016. URL: <https://arxiv.org/pdf/1603.02199>.
- [302] Sergey Levine et al. “Learning Hand-Eye Coordination for Robotic Grasping with Large-Scale Data Collection.” In: *2016 International Symposium on Experimental Robotics* (Cham, 2017). Ed. by Dana Kulić et al. Cham: Springer International Publishing, 2017, pp. 173–184. ISBN: 978-3-319-50115-4.
- [303] Thomas Feix et al. “The GRASP Taxonomy of Human Grasp Types.” In: *IEEE Transactions on Human-Machine Systems* 46.1 (2016), pp. 66–77. ISSN: 2168-2305. DOI: 10.1109/THMS.2015.2470657.
- [304] Sergey Levine et al. *Grasping Dataset*. Google Brain Robotics Data. Google Inc. 2016. URL: <https://sites.google.com/site/brainrobotdata/home/grasping-dataset> (visited on 12/16/2020).
- [305] Pieter-Tjerk de Boer et al. “A Tutorial on the Cross-Entropy Method.” In: *Annals of Operations Research* 134.1 (2005), pp. 19–67. ISSN: 1572-9338. DOI: 10.1007/s10479-005-5724-z.
- [306] Gijs van der Hoorn and Shaun Edwards. *kuka_experimental*. ROS-Industrial Program. 2018. URL: http://wiki.ros.org/kuka_experimental (visited on 12/23/2020).
- [307] Nicolas Alt. *nalt/wsg50-ros-pkg*. ROS package for Schunk WSG-50 Gripper. 2020. URL: <https://github.com/nalt/wsg50-ros-pkg> (visited on 12/23/2020).
- [308] Blender Foundation. *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. 2020. URL: <https://www.blender.org/> (visited on 01/18/2021).
- [309] Google Inc. *Procedurally Generated Random Objects*. Google Brain Robotics Data. 2017. URL: <https://sites.google.com/site/brainrobotdata/home/models> (visited on 12/23/2020).
- [310] Deirdre Quillen et al. “Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods.” In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 6284–6291. DOI: 10.1109/ICRA.2018.8461039.

-
- [311] Sachin Chitta et al. “ros_control: A generic and simple control framework for ROS.” In: *Journal of Open Source Software* 2.20 (2017), p. 456. DOI: 10.21105/joss.00456.
- [312] Sachin Chitta and Ian A. Sucas. *MoveIt Motion Planning Framework*. 2021. URL: <https://moveit.ros.org/> (visited on 01/18/2021).
- [313] Greg Brockman et al. *OpenAI Gym*. 2016. URL: <https://arxiv.org/pdf/1606.01540>.
- [314] Itai Caspi et al. *Reinforcement Learning Coach*. en. Zenodo, 2017. DOI: 10.5281/ZENODO.1134898.
- [315] Iker Zamora et al. *Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo*. 2017. URL: <https://arxiv.org/pdf/1608.05742>.
- [316] Nestor Gonzalez Lopez et al. *gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and Gazebo*. 2019. URL: <https://arxiv.org/pdf/1903.06278>.
- [317] Diego Ferigo et al. “Gym-Ignition: Reproducible Robotic Simulations for Reinforcement Learning.” In: *2020 IEEE/SICE International Symposium on System Integration (SII)*. 2020, pp. 885–890. DOI: 10.1109/SII46433.2020.9025951.
- [318] Automationspraxis. *Marktgrößen wie ABB und Kuka verstärken sich mit Firmenübernahmen. Der Platz neben dem Werker ist hart umkämpft*. 2015. URL: <https://automationspraxis.industrie.de/cobot/der-platz-neben-dem-werker-ist-hart-umkaempft/> (visited on 01/21/2021).
- [319] Stephen Soltesz et al. “Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors.” In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. EuroSys ’07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 275–287. ISBN: 9781595936363. DOI: 10.1145/1272996.1273025.
- [320] Docker Inc. *Docker overview*. 2020. URL: <https://docs.docker.com/get-started/overview/> (visited on 01/25/2021).
- [321] MongoDB Inc. *MongoDB Wire Protocol. MongoDB Manual*. 2020. (Visited on 01/25/2021).
- [322] MinIO, Inc. *MinIO. High Performance, Kubernetes Native Object Storage*. 2020. URL: <https://min.io/> (visited on 01/25/2021).
- [323] MongoDB Inc. *MongoDB. The most popular database for modern apps*. 2020. URL: <https://www.mongodb.com/> (visited on 01/25/2021).
- [324] Chun-hao Hu. *mongo-express. Web-based MongoDB admin interface*. 2021. URL: <https://github.com/mongo-express/mongo-express> (visited on 01/25/2021).
- [325] Metabase, Inc. *Metabase*. 2021. URL: <https://www.metabase.com/> (visited on 01/25/2021).
- [326] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning.” In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 1928–1937. URL: <http://proceedings.mlr.press/v48/mniha16.html>.
- [327] Lasse Espeholt et al. “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures.” In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholm, Stockholm Sweden: PMLR, 2018, pp. 1407–1416. URL: <http://proceedings.mlr.press/v80/espeholt18a.html>.
- [328] Lasse Espeholt et al. “SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference.” In: *Eighth International Conference on Learning Representations (ICLR 2020)*. 2020.
- [329] Chit-Kwan Lin et al. “Programming Spiking Neural Networks on Intel’s Loihi.” In: *Computer* 51.3 (2018), pp. 52–61. ISSN: 1558-0814. DOI: 10.1109/MC.2018.157113521.

- [330] Pierre Krack. “Closed-Loop Neuromorphic Reinforcement Learning on the Neurorobotics Platform with Intel Loihi.” Chair of Robotics, Artificial Intelligence and Real-Time Systems. Bachelor’s Thesis. Garching: Technical University of Munich, 2019.
- [331] Bart Ellenbroek and Jiun Youn. “Rodent models in neuroscience research: is it a rat race?” In: *Disease Models & Mechanisms* 9.10 (2016), p. 1079. DOI: 10.1242/dmm.026120.
- [332] Ed S. Lein et al. “Genome-wide atlas of gene expression in the adult mouse brain.” In: *Nature* 445.7124 (2007), pp. 168–176. ISSN: 1476-4687. DOI: 10.1038/nature05453.
- [333] Seung Wook Oh et al. “A mesoscale connectome of the mouse brain.” In: *Nature* 508.7495 (2014), pp. 207–214. ISSN: 1476-4687. DOI: 10.1038/nature13186.
- [334] Csaba Erö et al. “A Cell Atlas for the Mouse Brain.” In: *Frontiers in Neuroinformatics* 12 (2018), p. 84. ISSN: 1662-5196. DOI: 10.3389/fninf.2018.00084.
- [335] Peer Lucas et al. *Design of a Biomimetic Rodent Robot*. TUM Chair of Robotics, Artificial Intelligence and Real-Time Systems, 2018.
- [336] Tim C. Lueth. “SG-Library: Entwicklung einer konstruktiven MATLAB-Toolbox zur räumlichen Modellierung von Körpern, Gelenken und Getrieben.” de. In: *11. Kolloquium Getriebetechnik*. Ed. by Tim C. Lüth et al. Lehrstuhl für Mikrotechnik und Medizingerätetechnik, TUM. Garching b. München, Deutschland, 2015, pp. 183–203. DOI: 10.14459/2015md1276136.
- [337] Martin S. Fischer et al. “Basic limb kinematics of small therian mammals.” In: *Journal of Experimental Biology* 205.9 (2002), p. 1315.
- [338] Xiaodong Zhou and Shusheng Bi. “A survey of bio-inspired compliant legged robot designs.” In: *Bioinspiration & Biomimetics* 7.4 (2012), p. 041001. DOI: 10.1088/1748-3182/7/4/041001.
- [339] Alessandro Crespi et al. “Salamandra Robotica II: An Amphibious Robot to Study Salamander-Like Swimming and Walking Gaits.” In: *IEEE Transactions on Robotics* 29.2 (2013), pp. 308–320. DOI: 10.1109/TRO.2012.2234311.
- [340] H. Witte et al. “Transfer of biological principles into the construction of quadruped walking machines.” In: *Proceedings of the Second International Workshop on Robot Motion and Control. RoMoCo’01 (IEEE Cat. No.01EX535)*. 2001, pp. 245–249. DOI: 10.1109/ROMOCO.2001.973462.
- [341] Alexander Spröwitz et al. “Towards dynamic trot gait locomotion: Design, control, and experiments with Cheetah-cub, a compliant quadruped robot.” In: *The International Journal of Robotics Research* 32.8 (2013), pp. 932–950. ISSN: 0278-3649. DOI: 10.1177/0278364913489205.
- [342] Eva Marie Siehmann. “Design of a Mouse-inspired Biomimetic Leg for a Walking Robot.” Chair of Robotics, Artificial Intelligence and Real-Time Systems. Master’s Thesis. Garching: Technical University of Munich, 2016.
- [343] Peer Konstantin Lucas. “Design, Construction and Validation of a Life-Size Robot Model for Biomimetic Locomotion in Small Mammals.” Chair of Robotics, Artificial Intelligence and Real-Time Systems. Master’s Thesis. Garching: Technical University of Munich, 2017.
- [344] T. Würtz et al. “The twisted string actuation system: Modeling and control.” In: *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. 2010, pp. 1215–1220. DOI: 10.1109/AIM.2010.5695720.
- [345] G. Palli et al. “The DEXMART hand: Mechatronic design and experimental evaluation of synergy-based control for human-like grasping.” In: *The International Journal of Robotics Research* 33.5 (2014), pp. 799–824. ISSN: 0278-3649. DOI: 10.1177/0278364913519897.
- [346] Hugo Gravato Marques et al. “ECCE1: The first of a series of anthropomorphic musculoskeletal upper torsos.” In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. 2010, pp. 391–396. DOI: 10.1109/ICHR.2010.5686344.

-
- [347] Rafael Hostettler. *Roboy. Project Website*. 2021. URL: <https://roboy.org/> (visited on 02/26/2021).
- [348] Yuki Asano et al. "Human mimetic musculoskeletal humanoid Kengoro toward real world physically interactive actions." In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. 2016, pp. 876–883. DOI: 10.1109/HUMANOIDS.2016.7803376.
- [349] K. Karakasiliotis et al. "From cineradiography to biorobots: an approach for designing robots to emulate and study animal locomotion." In: *Journal of The Royal Society Interface* 13.119 (2016), p. 20151089. DOI: 10.1098/rsif.2015.1089.
- [350] K. Weinmeister et al. "Cheetah-cub-S: Steering of a quadruped robot using trunk motion." In: *2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2015, pp. 1–6. DOI: 10.1109/SSRR.2015.7443021.
- [351] Thomas Krieger. "Design of an Actuated Mechanical Vertebral Column Model for a Biomimetic Mouse Robot." Chair of Robotics, Artificial Intelligence and Real-Time Systems. Master's Thesis. Garching: Technical University of Munich, 2017.
- [352] Peer Lucas et al. "Development of the Neurorobotic Mouse." In: *2019 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. 2019, pp. 299–304. DOI: 10.1109/CBS46900.2019.9114441.
- [353] Ruchi Parekh and Giorgio A. Ascoli. "Neuronal Morphology Goes Digital: A Research Hub for Cellular and System Neuroscience." In: *Neuron* 77.6 (2013), pp. 1017–1038. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2013.03.008.
- [354] Christoph von der Malsburg. "The What and Why of Binding: The Modeler's Perspective." In: *Neuron* 24.1 (1999), pp. 95–104. ISSN: 0896-6273. DOI: 10.1016/S0896-6273(00)80825-9.
- [355] Eric R. Kandel et al. "2: Nerve Cells, Neural Circuitry, and Behavior." In: *Principles of Neural Science*. Ed. by Eric R. Kandel et al. 5th ed. New York, Lisbon, and London: McGraw-Hill Medical, 2013. ISBN: 978-0071390118.
- [356] Nathan W. Gouwens et al. "Classification of electrophysiological and morphological neuron types in the mouse visual cortex." In: *Nature Neuroscience* 22.7 (2019), pp. 1182–1195. ISSN: 1546-1726. DOI: 10.1038/s41593-019-0417-0.
- [357] Richard H. Masland. "Neuronal cell types." In: *Current Biology* 14.13 (2004), R497–R500. ISSN: 0960-9822. DOI: 10.1016/j.cub.2004.06.035.
- [358] Hermina Nedelescu et al. "Regional differences in Purkinje cell morphology in the cerebellar vermis of male mice." In: *Journal of Neuroscience Research* 96.9 (2018), pp. 1476–1489. ISSN: 0360-4012. DOI: 10.1002/jnr.24206.
- [359] Andrew J. Trevelyan et al. "Modular Propagation of Epileptiform Activity: Evidence for an Inhibitory Veto in Neocortex." In: *The Journal of Neuroscience* 26.48 (2006), p. 12447. DOI: 10.1523/JNEUROSCI.2787-06.2006.
- [360] Joseph W. Goodliffe et al. "Differential changes to D1 and D2 medium spiny neurons in the 12-month-old Q175+/- mouse model of Huntington's Disease." In: *PLOS ONE* 13.8 (2018), e0200626. DOI: 10.1371/journal.pone.0200626.
- [361] Jee-Hyun Kong et al. "Diversity of ganglion cells in the mouse retina: Unsupervised morphological classification and its limits." In: *Journal of Comparative Neurology* 489.3 (2005), pp. 293–310. ISSN: 0021-9967. DOI: 10.1002/cne.20631.
- [362] Giorgio A. Ascoli et al. "NeuroMorpho.Org: A Central Resource for Neuronal Morphologies." In: *The Journal of Neuroscience* 27.35 (2007), p. 9247. DOI: 10.1523/JNEUROSCI.2055-07.2007.
- [363] Marwan Abdellah et al. "NeuroMorphoVis: a collaborative framework for analysis and visualization of neuronal morphology skeletons reconstructed from microscopy stacks." In: *Bioinformatics* 34.13 (2018), pp. i574–i582. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty231.

Bibliography

- [364] Klaus M. Stiefel and Terrence J. Sejnowski. “Mapping Function Onto Neuronal Morphology.” In: *Journal of Neurophysiology* 98.1 (2007), pp. 513–526. ISSN: 0022-3077. DOI: 10.1152/jn.00865.2006.
- [365] Ben Torben-Nielsen et al. “On the Neuronal Morphology-Function Relationship: A Synthetic Approach.” In: *Knowledge Discovery and Emergent Complexity in Bioinformatics* (Berlin, Heidelberg, 2007). Ed. by Karl Tuyls et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 131–144. ISBN: 978-3-540-71037-0.
- [366] Matthew Larkum. “A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex.” In: *Trends in Neurosciences* 36.3 (2013), pp. 141–151. ISSN: 0166-2236. DOI: 10.1016/j.tins.2012.11.006.
- [367] Robert Urbanczik and Walter Senn. “Learning by the Dendritic Prediction of Somatic Spiking.” In: *Neuron* 81.3 (2014), pp. 521–528. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2013.11.030.
- [368] João Sacramento et al. “Dendritic cortical microcircuits approximate the backpropagation algorithm.” In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc, 2018, pp. 8721–8732.
- [369] Joshua R. Sanes and Thomas M. Jessell. “55: Formation and Elimination of Synapses.” In: *Principles of Neural Science*. Ed. by Eric R. Kandel et al. 5th ed. New York, Lisbon, and London: McGraw-Hill Medical, 2013. ISBN: 978-0071390118.
- [370] Joshua R. Sanes and Thomas M. Jessell. “54: The Growth and Guidance of Axons.” In: *Principles of Neural Science*. Ed. by Eric R. Kandel et al. 5th ed. New York, Lisbon, and London: McGraw-Hill Medical, 2013. ISBN: 978-0071390118.
- [371] Michael W. Reimann et al. “An algorithm to predict the connectome of neural microcircuits.” In: *Frontiers in Computational Neuroscience* 9 (2015), p. 28. ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00120.
- [372] Joshua R. Sanes and Thomas M. Jessell. “56: Experience and the Refinement of Synaptic Connections.” In: *Principles of Neural Science*. Ed. by Eric R. Kandel et al. 5th ed. New York, Lisbon, and London: McGraw-Hill Medical, 2013. ISBN: 978-0071390118.
- [373] Gerald M. Edelman. “Neural Darwinism: Selection and reentrant signaling in higher brain function.” In: *Neuron* 10.2 (1993), pp. 115–125. ISSN: 0896-6273. DOI: 10.1016/0896-6273(93)90304-A.
- [374] Jose-Manuel Alonso and Yao Chen. “Receptive field.” In: *Scholarpedia* 4.1 (2009), p. 5393. ISSN: 1941-6016. DOI: 10.4249/scholarpedia.5393.
- [375] Mark D. Kirk et al. “A quantitative correlation of contour sensitivity with dendritic density in an identified visual neuron.” In: *Brain Research* 274.2 (1983), pp. 231–237. ISSN: 0006-8993. DOI: 10.1016/0006-8993(83)90700-X.
- [376] Mark F. Jacquin et al. “Structure-function relationships in rat brainstem subnucleus interpolaris: IV. Projection neurons.” In: *Journal of Comparative Neurology* 282.1 (1989), pp. 45–62. ISSN: 0021-9967. DOI: 10.1002/cne.902820105.
- [377] Sandra Single and Alexander Borst. “Dendritic Integration and Its Role in Computing Image Velocity.” In: *Science* 281.5384 (1998), p. 1848. DOI: 10.1126/science.281.5384.1848.
- [378] Hongbo Jia et al. “Dendritic organization of sensory input to cortical neurons in vivo.” In: *Nature* 464.7293 (2010), pp. 1307–1312. ISSN: 1476-4687. DOI: 10.1038/nature08947.
- [379] J. D. Pettigrew et al. “Cortical effect of selective visual experience: degeneration or reorganization?” In: *Brain Research* 51 (1973), pp. 345–351. ISSN: 0006-8993. DOI: 10.1016/0006-8993(73)90387-9.
- [380] Alexander Hecker et al. “Removing a single neuron in a vertebrate brain forever abolishes an essential behavior.” In: *Proceedings of the National Academy of Sciences* 117.6 (2020), p. 3254. DOI: 10.1073/pnas.1918578117.

-
- [381] Charles G. Gross. "Genealogy of the "Grandmother Cell"." In: *The Neuroscientist* 8.5 (2002), pp. 512–518. ISSN: 1073-8584. DOI: 10.1177/107385802237175.
- [382] R. Quiñan Quiroga et al. "Invariant visual representation by single neurons in the human brain." In: *Nature* 435.7045 (2005), pp. 1102–1107. ISSN: 1476-4687. DOI: 10.1038/nature03687.
- [383] Gabriel Kreiman et al. "Category-specific visual responses of single neurons in the human medial temporal lobe." In: *Nature Neuroscience* 3.9 (2000), pp. 946–953. ISSN: 1546-1726. DOI: 10.1038/78868.
- [384] T. N. Wiesel and D. H. Hubel. "Spatial and chromatic interactions in the lateral geniculate body of the rhesus monkey." In: *Journal of Neurophysiology* 29.6 (1966), pp. 1115–1156. ISSN: 0022-3077. DOI: 10.1152/jn.1966.29.6.1115.
- [385] Jeffery A. Winer et al. "Auditory thalamocortical transformation: structure and function." In: *Trends in Neurosciences* 28.5 (2005), pp. 255–263. ISSN: 0166-2236. DOI: 10.1016/j.tins.2005.03.009.
- [386] Melissa Saenz and Dave R.M. Langers. "Tonotopic mapping of human auditory cortex." In: *Hearing Research* 307 (2014), pp. 42–52. ISSN: 0378-5955. DOI: 10.1016/j.heares.2013.07.016.
- [387] J. O'Keefe and J. Dostrovsky. "The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat." In: *Brain Research* 34.1 (1971), pp. 171–175. ISSN: 0006-8993. DOI: 10.1016/0006-8993(71)90358-1.
- [388] J. S. Taube et al. "Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis." In: *The Journal of Neuroscience* 10.2 (1990), p. 420. DOI: 10.1523/JNEUROSCI.10-02-00420.1990.
- [389] Leonard F. Koziol et al. "Consensus Paper: The Cerebellum's Role in Movement and Cognition." In: *The Cerebellum* 13.1 (2014), pp. 151–177. ISSN: 1473-4230. DOI: 10.1007/s12311-013-0511-x.
- [390] Chris Rorden and Hans-Otto Karnath. "Using human brain lesions to infer function: a relic from a past era in the fMRI age?" In: *Nature Reviews Neuroscience* 5.10 (2004), pp. 812–819. ISSN: 1471-0048. DOI: 10.1038/nrn1521.
- [391] Avinash R. Vaidya et al. "Lesion Studies in Contemporary Neuroscience." In: *Trends in Cognitive Sciences* 23.8 (2019), pp. 653–671. ISSN: 1364-6613. DOI: 10.1016/j.tics.2019.05.009.
- [392] Leah Krubitzer. "The Magnificent Compromise: Cortical Field Evolution in Mammals." In: *Neuron* 56.2 (2007), pp. 201–208. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2007.10.002.
- [393] Daniel J. Felleman and David C. van Essen. "Distributed Hierarchical Processing in the Primate Cerebral Cortex." In: *Cerebral Cortex* 1.1 (1991), pp. 1–47. ISSN: 1047-3211. DOI: 10.1093/cercor/1.1.1-a.
- [394] Carl R. Olson and Carol L. Colby. "18: The Organization of Cognition." In: *Principles of Neural Science*. Ed. by Eric R. Kandel et al. 5th ed. New York, Lisbon, and London: McGraw-Hill Medical, 2013. ISBN: 978-0071390118.
- [395] David G. Amaral and Peter L. Strick. "15: The Organization of the Central Nervous System." In: *Principles of Neural Science*. Ed. by Eric R. Kandel et al. 5th ed. New York, Lisbon, and London: McGraw-Hill Medical, 2013. ISBN: 978-0071390118.
- [396] Mortimer Mishkin et al. "Object vision and spatial vision: two cortical pathways." In: *Trends in Neurosciences* 6 (1983), pp. 414–417. ISSN: 0166-2236. DOI: 10.1016/0166-2236(83)90190-X.
- [397] Laura Alonso Recio. "Dorsal Pathway." In: *Encyclopedia of Animal Cognition and Behavior*. Ed. by Jennifer Vonk and Todd Shackelford. Cham: Springer International Publishing, 2017, pp. 1–7. ISBN: 978-3-319-47829-6. DOI: 10.1007/978-3-319-47829-6_1235-1.
- [398] Roger B.H. Tootell et al. "New images from human visual cortex." In: *Trends in Neurosciences* 19.11 (1996), pp. 481–489. ISSN: 0166-2236. DOI: 10.1016/S0166-2236(96)10053-9.
- [399] Nikos K. Logothetis. "Vision: A Window on Consciousness." In: *Scientific American* 281.5 (1999), pp. 68–75. ISSN: 00368733.

Bibliography

- [400] L. M. Romanski et al. “Dual streams of auditory afferents target multiple domains in the primate prefrontal cortex.” In: *Nature Neuroscience* 2.12 (1999), pp. 1131–1136. ISSN: 1546-1726. DOI: 10.1038/16056.
- [401] M. Livingstone and D. Hubel. “Segregation of form, color, movement, and depth: anatomy, physiology, and perception.” In: *Science* 240.4853 (1988), p. 740. DOI: 10.1126/science.3283936.
- [402] Insub Kim et al. “Neural representations of perceptual color experience in the human ventral visual pathway.” In: *Proceedings of the National Academy of Sciences* 117.23 (2020), p. 13145. DOI: 10.1073/pnas.1911041117.
- [403] Massieh Moayedi and Karen D. Davis. “Theories of pain: from specificity to gate control.” In: *Journal of Neurophysiology* 109.1 (2012), pp. 5–12. ISSN: 0022-3077. DOI: 10.1152/jn.00457.2012.
- [404] Robert Andrew Wilson and Frank C. Keil, eds. *The MIT Encyclopedia of the Cognitive Sciences*. eng. Cambridge, Mass.: MIT Press, 2001. 964 pp. ISBN: 9780262731447.
- [405] W. Singer and C. M. Gray. “Visual Feature Integration and the Temporal Correlation Hypothesis.” In: *Annual Review of Neuroscience* 18.1 (1995), pp. 555–586. ISSN: 0147-006X. DOI: 10.1146/annurev.ne.18.030195.003011.
- [406] Christoph von der Malsburg. “A Vision Architecture.” In: *Proceedings of the International Conference on Neural Computation Theory and Applications*. Ed. by Kurosh Madani. SciTePress, 2014, pp. 345–350. ISBN: 9789897580543. DOI: 10.5220/0005158103450350.
- [407] Christoph von der Malsburg. *Concerning the Neural Code*. 2018. URL: <https://arxiv.org/pdf/1811.01199> (visited on 08/05/2020).
- [408] Olaf Sporns et al. “The Human Connectome: A Structural Description of the Human Brain.” In: *PLOS Computational Biology* 1.4 (2005), e42. DOI: 10.1371/journal.pcbi.0010042.
- [409] E. A. DeYoe and D. C. van Essen. “Concurrent processing streams in monkey visual cortex.” In: *Trends in Neurosciences* 11.5 (1988), pp. 219–226. ISSN: 0166-2236. DOI: 10.1016/0166-2236(88)90130-0.
- [410] Jonas M. D. Enander et al. “Ubiquitous Neocortical Decoding of Tactile Input Patterns.” In: *Frontiers in Cellular Neuroscience* 13 (2019), p. 140. ISSN: 1662-5102. DOI: 10.3389/fncel.2019.00140.
- [411] Thomas Elbert and Brigitte Rockstroh. “Reorganization of Human Cerebral Cortex: The Range of Changes Following Use and Injury.” In: *The Neuroscientist* 10.2 (2004), pp. 129–141. ISSN: 1073-8584. DOI: 10.1177/1073858403262111.
- [412] Richard E. Passingham et al. “The anatomical basis of functional localization in the cortex.” In: *Nature Reviews Neuroscience* 3.8 (2002), pp. 606–616. ISSN: 1471-0048. DOI: 10.1038/nrn893.
- [413] Róbert Csordás et al. *Are Neural Nets Modular? Inspecting Functional Modularity Through Differentiable Weight Masks*. 2020. URL: <https://arxiv.org/pdf/2010.02066> (visited on 03/06/2021).
- [414] Daniel L. K. Yamins et al. “Performance-optimized hierarchical models predict neural responses in higher visual cortex.” In: *Proceedings of the National Academy of Sciences* 111.23 (2014), p. 8619. DOI: 10.1073/pnas.1403112111.
- [415] Gasser Auda and Mohamed Kamel. “Modular Neural Networks. A Survey.” In: *International Journal of Neural Systems* 09.02 (1999), pp. 129–151. ISSN: 0129-0657. DOI: 10.1142/S0129065799000125.
- [416] P. Gallinari. “Modular neural net systems, training of.” In: *The Handbook of Brain Theory and Neural Networks*. Ed. by Michael A. Arbib. 1st ed. A Bradford book. Cambridge, Mass.: MIT Press, 1995. ISBN: 0262011484.
- [417] Farooq Azam. “Biologically inspired modular neural networks.” Virginia Tech, 2000.
- [418] Amanda J. C. Sharkey. “On Combining Artificial Neural Nets.” In: *Connection Science* 8.3-4 (1996), pp. 299–314. ISSN: 0954-0091. DOI: 10.1080/095400996116785.

-
- [419] Ke Chen. “Deep and Modular Neural Networks.” In: *Springer Handbook of Computational Intelligence*. Ed. by Janusz Kacprzyk and Witold Pedrycz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 473–494. ISBN: 978-3-662-43505-2. DOI: 10.1007/978-3-662-43505-2_28.
- [420] Mohammed Amer and Tomás Maul. “A review of modularization techniques in artificial neural networks.” In: *Artificial Intelligence Review* 52.1 (2019), pp. 527–561. ISSN: 1573-7462. DOI: 10.1007/s10462-019-09706-7.
- [421] German I. Parisi et al. “Continual lifelong learning with neural networks: A review.” In: *Neural Networks* 113 (2019), pp. 54–71. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.01.012.
- [422] Jonathan A. Michaels et al. “A goal-driven modular neural network predicts parietofrontal neural dynamics during grasping.” In: *Proceedings of the National Academy of Sciences* 117.50 (2020), p. 32124. DOI: 10.1073/pnas.2005087117.
- [423] Anders Krogh and Jesper Vedelsby. “Neural Network Ensembles, Cross Validation and Active Learning.” In: *Proceedings of the 7th International Conference on Neural Information Processing Systems*. NIPS’94. Cambridge, MA, USA: MIT Press, 1994, pp. 231–238.
- [424] Robert A. Jacobs et al. “Adaptive Mixtures of Local Experts.” In: *Neural Computation* 3.1 (1991), pp. 79–87. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.1.79.
- [425] Robert A. Jacobs et al. “Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks.” In: *Cognitive Science* 15.2 (1991), pp. 219–250. ISSN: 0364-0213. DOI: 10.1016/0364-0213(91)80006-Q.
- [426] Frédéric Gruau. “Automatic Definition of Modular Neural Networks.” In: *Adaptive Behavior* 3.2 (1994), pp. 151–183. ISSN: 1059-7123. DOI: 10.1177/105971239400300202.
- [427] Jacob Schrum and Risto Miikkulainen. “Evolving Multimodal Behavior with Modular Neural Networks in Ms. Pac-Man.” In: *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. GECCO ’14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 325–332. ISBN: 9781450326629. DOI: 10.1145/2576768.2598234.
- [428] Ali Sharif Razavian et al. “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition.” In: *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 512–519. DOI: 10.1109/CVPRW.2014.131.
- [429] Louis Kirsch et al. “Modular Networks: Learning to Decompose Neural Computation.” In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS’18. Red Hook, NY, USA: Curran Associates Inc, 2018, pp. 2414–2423.
- [430] Dan Cireşan et al. “A committee of neural networks for traffic sign classification.” In: *The 2011 International Joint Conference on Neural Networks*. 2011, pp. 1918–1921. DOI: 10.1109/IJCNN.2011.6033458.
- [431] Dan Claudiu Cireşan et al. “Convolutional Neural Network Committees for Handwritten Character Classification.” In: *2011 International Conference on Document Analysis and Recognition*. 2011, pp. 1135–1139. DOI: 10.1109/ICDAR.2011.229.
- [432] Jacob Andreas et al. “Learning to Compose Neural Networks for Question Answering.” In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, 2016, pp. 1545–1554. DOI: 10.18653/v1/N16-1181. URL: <https://www.aclweb.org/anthology/N16-1181>.
- [433] Andrei A. Rusu et al. *Progressive Neural Networks*. 2016. URL: <https://arxiv.org/pdf/1606.04671>.
- [434] Andrei A. Rusu et al. “Sim-to-Real Robot Learning from Pixels with Progressive Nets.” In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine et al. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 262–270. URL: <http://proceedings.mlr.press/v78/rusu17a.html>.

- [435] Alexander V. Terekhov et al. “Knowledge Transfer in Deep Block-Modular Neural Networks.” In: *Biomimetic and Biohybrid Systems* (Cham, 2015). Ed. by Stuart P. Wilson et al. Cham: Springer International Publishing, 2015, pp. 268–279. ISBN: 978-3-319-22979-9.
- [436] Coline Devin et al. “Learning modular neural network policies for multi-task and multi-robot transfer.” In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2169–2176. doi: 10.1109/ICRA.2017.7989250.
- [437] Jiquan Ngiam et al. “Multimodal Deep Learning.” In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. Ed. by Lise Getoor and Tobias Scheffer. ICML ’11. New York, NY, USA: ACM, 2011, pp. 689–696. ISBN: 978-1-4503-0619-5.
- [438] Harry McGurk and John MacDonald. “Hearing lips and seeing voices.” In: *Nature* 264.5588 (1976), pp. 746–748. ISSN: 1476-4687. doi: 10.1038/264746a0.
- [439] Sarath Chandar et al. “Correlational Neural Networks.” In: *Neural Computation* 28.2 (2016), pp. 257–285. ISSN: 0899-7667. doi: 10.1162/NECO_a_00801.
- [440] Alain Droniou et al. “Deep unsupervised network for multimodal perception, representation and classification.” In: *Robotics and Autonomous Systems* 71 (2015), pp. 83–98. ISSN: 0921-8890. doi: 10.1016/j.robot.2014.11.005.
- [441] A. Eitel et al. “Multimodal deep learning for robust RGB-D object recognition.” In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 681–687. doi: 10.1109/IROS.2015.7353446.
- [442] N. Rathi and K. Roy. “STDP Based Unsupervised Multimodal Learning With Cross-Modal Processing in Spiking Neural Networks.” In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 5.1 (2021), pp. 143–153. ISSN: 2471-285X. doi: 10.1109/TETCI.2018.2872014.
- [443] Stephane Lallee and Peter Ford Dominey. “Multi-modal convergence maps: from body schema and self-representation to mental imagery.” In: *Adaptive Behavior* 21.4 (2013), pp. 274–285. ISSN: 1059-7123. doi: 10.1177/1059712313488423.
- [444] Antonio R. Damasio. “Time-locked multiregional retroactivation: A systems-level proposal for the neural substrates of recall and recognition.” In: *Cognition* 33.1 (1989), pp. 25–62. ISSN: 0010-0277. doi: 10.1016/0010-0277(89)90005-X.
- [445] Cristian Axenie et al. “A Self-Synthesis Approach to Perceptual Learning for Multisensory Fusion in Robotics.” In: *Sensors* 16.10 (2016). doi: 10.3390/s16101751.
- [446] Lyes Khacef et al. “Brain-Inspired Self-Organization with Cellular Neuromorphic Computing for Multimodal Unsupervised Learning.” In: *Electronics* 9.10 (2020). doi: 10.3390/electronics9101605.
- [447] Gerald Edelman and Joseph Gally. “Reentry: a key mechanism for integration of brain function.” In: *Frontiers in Integrative Neuroscience* 7 (2013), p. 63. ISSN: 1662-5145. doi: 10.3389/fnint.2013.00063.
- [448] Jay G. Rueckl et al. “Why are “What” and “Where” Processed by Separate Cortical Visual Systems? A Computational Investigation.” In: *Journal of Cognitive Neuroscience* 1.2 (1989), pp. 171–186. ISSN: 0898-929X. doi: 10.1162/jocn.1989.1.2.171.
- [449] Grace W. Lindsay. “Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future.” In: *Journal of Cognitive Neuroscience* (2020), pp. 1–15. ISSN: 0898-929X. doi: 10.1162/jocn_a_01544.
- [450] Ed Bullmore and Olaf Sporns. “Complex brain networks: graph theoretical analysis of structural and functional systems.” In: *Nature Reviews Neuroscience* 10.3 (2009), pp. 186–198. ISSN: 1471-0048. doi: 10.1038/nrn2575.
- [451] Olaf Sporns. “Network attributes for segregation and integration in the human brain.” In: *Current Opinion in Neurobiology* 23.2 (2013), pp. 162–171. ISSN: 0959-4388. doi: 10.1016/j.conb.2012.11.015.

-
- [452] Gorka Zamora-López et al. “Cortical hubs form a module for multisensory integration on top of the hierarchy of cortical networks.” In: *Frontiers in Neuroinformatics* 4 (2010), p. 1. ISSN: 1662-5196. DOI: 10.3389/neuro.11.001.2010.
- [453] Davis Blalock et al. “What is the State of Neural Network Pruning?” In: *Proceedings of Machine Learning and Systems*. Ed. by I. Dhillon et al. Vol. 2. 2020, pp. 129–146. URL: <https://proceedings.mlsys.org/paper/2020/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf>.
- [454] Olaf Sporns et al. “Organization, development and function of complex brain networks.” In: *Trends in Cognitive Sciences* 8.9 (2004), pp. 418–425. ISSN: 1364-6613. DOI: 10.1016/j.tics.2004.07.008.
- [455] Peter R. Huttenlocher. “Synaptic density in human frontal cortex – Developmental changes and effects of aging.” In: *Brain Research* 163.2 (1979), pp. 195–205. ISSN: 0006-8993. DOI: 10.1016/0006-8993(79)90349-4.
- [456] Christoph Kayser and Nikos K. Logothetis. “Do early sensory cortices integrate cross-modal information?” In: *Brain Structure and Function* 212.2 (2007), pp. 121–132. ISSN: 1863-2661. DOI: 10.1007/s00429-007-0154-0.
- [457] Jon Driver and Toemme Noesselt. “Multisensory Interplay Reveals Crossmodal Influences on ‘Sensory-Specific’ Brain Regions, Neural Responses, and Judgments.” In: *Neuron* 57.1 (2008), pp. 11–23. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2007.12.013.
- [458] Eric Weisstein. “Bessel’s Correction.” In: *MathWorld. A Wolfram Web Resource*. Ed. by Wolfram Research, Inc. 2021. (Visited on 05/19/2021).
- [459] Eric Weisstein. “Statistical Correlation.” In: *MathWorld. A Wolfram Web Resource*. Ed. by Wolfram Research, Inc. 2021. (Visited on 05/19/2021).
- [460] B. P. Welford. “Note on a Method for Calculating Corrected Sums of Squares and Products.” In: *Technometrics* 4.3 (1962), pp. 419–420. ISSN: 0040-1706. DOI: 10.1080/00401706.1962.10490022.
- [461] Terrence J. Sejnowski and Gerald Tesauro. “6 - The Hebb Rule for Synaptic Plasticity: Algorithms and Implementations.” In: *Neural Models of Plasticity*. Ed. by John H. Byrne and William O. Berry. Academic Press, 1989, pp. 94–103. ISBN: 978-0-12-148955-7. DOI: 10.1016/B978-0-12-148955-7.50010-2.
- [462] Terrence Sejnowski et al. “Induction of Synaptic Plasticity by Hebbian Covariance in the Hippocampus.” In: *The Computing Neuron*. Ed. by Richard M. Durbin et al. Computation and Neural Systems Series. Redwood City, CA: Addison-Wesley, 1989, pp. 105–124. ISBN: 020118348X.
- [463] Matteo Carandini and David J. Heeger. “Normalization as a canonical neural computation.” In: *Nature Reviews Neuroscience* 13.1 (2012), pp. 51–62. ISSN: 1471-0048. DOI: 10.1038/nrn3136.
- [464] Tobias Jülg. “Self-Supervised Learning of Sensorimotor Representations in a Neurorobotics Experiment.” Chair of Robotics, Artificial Intelligence and Real-Time Systems. Guided Research Project Report. Garching: Technical University of Munich, 2021.
- [465] KUKA Roboter GmbH. *KUKA LBR iiwa Product Brochure*. 2017. URL: <https://www.kuka.com/de/de/produkte-leistungen/robotersysteme/industrieroboter/lbr-iiwa> (visited on 12/12/2021).
- [466] William Falcon. *PyTorch Lightning*. 2021. URL: <https://github.com/PyTorchLightning/pytorch-lightning> (visited on 06/02/2021).
- [467] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *3rd International Conference on Learning Representations (ICLR)* (San Diego, CA). Ed. by Yoshua Bengio et al. 2015. URL: <https://arxiv.org/pdf/1412.6980>.
- [468] Xueyuan Zhou and Mikhail Belkin. “Chapter 22 - Semi-Supervised Learning.” In: *Academic Press Library in Signal Processing*. Ed. by Sergios Theodoridis and Rama Chellappa. 1st ed. Vol. 1. Elsevier, 2013, pp. 1239–1269. ISBN: 9780124166349. DOI: 10.1016/B978-0-12-396502-8.00022-X.

Bibliography

- [469] Fayeem Aziz et al. “Semi-Supervised Manifold Alignment Using Parallel Deep Autoencoders.” In: *Algorithms* 12.9 (2019). doi: 10.3390/a12090186.
- [470] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning.” In: *Nature* 518.7540 (2015), pp. 529–533. issn: 1476-4687. doi: 10.1038/nature14236.
- [471] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning.” In: *4th International Conference on Learning Representations (ICLR)*. Ed. by Hugo Larochelle et al. 2016.
- [472] Rainer Goebel. “Functional Organization of the Primary Visual Cortex.” In: *Brain Mapping*. Ed. by Arthur W. Toga. Waltham: Academic Press, 2015, pp. 287–291. isbn: 978-0-12-397316-0. doi: 10.1016/B978-0-12-397025-1.00224-4.
- [473] American Psychological Association. *topographic organization*. *APA Dictionary of Psychology*. 2021. url: <https://dictionary.apa.org/topographic-organization> (visited on 06/10/2021).
- [474] Jon H. Kaas. “Topographic Maps are Fundamental to Sensory Processing.” In: *Brain Research Bulletin* 44.2 (1997), pp. 107–112. issn: 0361-9230. doi: 10.1016/S0361-9230(97)00094-4.
- [475] Jianhua Cang and David A. Feldheim. “Developmental Mechanisms of Topographic Map Formation and Alignment.” In: *Annual Review of Neuroscience* 36.1 (2013), pp. 51–77. issn: 0147-006X. doi: 10.1146/annurev-neuro-062012-170341.
- [476] Michele A. Basso and Paul J. May. “Circuits for Action and Cognition: A View from the Superior Colliculus.” In: *Annual Review of Vision Science* 3.1 (2017), pp. 197–226. issn: 2374-4642. doi: 10.1146/annurev-vision-102016-061234.
- [477] Jon Louis Bentley. “Multidimensional Binary Search Trees Used for Associative Searching.” In: *Commun. ACM* 18.9 (1975), pp. 509–517. issn: 0001-0782. doi: 10.1145/361002.361007.
- [478] Pim de Haan and Luca Falorsi. *Topological Constraints on Homeomorphic Auto-Encoding*. 2018. url: <https://arxiv.org/pdf/1812.10783>.
- [479] Michael Moor et al. “Topological Autoencoders.” In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 7045–7054. url: <http://proceedings.mlr.press/v119/moor20a.html>.
- [480] Rochel Gelman. “Cognitive Development.” In: *The MIT Encyclopedia of the Cognitive Sciences*. Ed. by Robert Andrew Wilson and Frank C. Keil. Cambridge, Mass.: MIT Press, 2001, pp. 128–129. isbn: 9780262731447.
- [481] Jean Piaget. *The Child’s Conception of the World*. In collab. with Joan Tomlinson and Andrew Tomlinson. London: Routledge and Kegan Paul Ltd, 1929. isbn: 0710030681.
- [482] Marc H. Bornstein et al. “human behavior.” In: *Britannica Academic*. 2020. (Visited on 06/14/2021).
- [483] Takao K. Hensch. “CRITICAL PERIOD REGULATION.” In: *Annual Review of Neuroscience* 27.1 (2004), pp. 549–579. issn: 0147-006X. doi: 10.1146/annurev.neuro.27.070203.144327.
- [484] J. Sebastian Espinosa and Michael P. Stryker. “Development and Plasticity of the Primary Visual Cortex.” In: *Neuron* 75.2 (2012), pp. 230–249. issn: 0896-6273. doi: 10.1016/j.neuron.2012.06.009.
- [485] Mijna Hadders-Algra. “Neural substrate and clinical significance of general movements: an update.” In: *Developmental Medicine & Child Neurology* 60.1 (2018), pp. 39–46. issn: 0012-1622. doi: 10.1111/dmcn.13540.
- [486] Angelo Cangelosi and Matthew Schlesinger. “From Babies to Robots: The Contribution of Developmental Robotics to Developmental Psychology.” In: *Child Development Perspectives* 12.3 (2018), pp. 183–188. issn: 1750-8592. doi: 10.1111/cdep.12282.
- [487] Alan M. Turing. “Computing Machinery and Intelligence.” In: *Mind* 59.236 (1950), pp. 433–460. issn: 00264423.

-
- [488] Max Lungarella et al. “Developmental robotics. A survey.” In: *Connection Science* 15.4 (2003), pp. 151–190. ISSN: 0954-0091. DOI: 10.1080/09540090310001655110.
- [489] Oliver Braddick and Janette Atkinson. “Development of human visual function.” In: *Vision Research 50th Anniversary Issue: Part 2* 51.13 (2011), pp. 1588–1609. ISSN: 0042-6989. DOI: 10.1016/j.visres.2011.02.018.
- [490] Gerald Turkewitz and Patricia A. Kenny. “Limitations on input as a basis for neural organization and perceptual development: A preliminary theoretical statement.” In: *Developmental Psychobiology* 15.4 (1982), pp. 357–368. ISSN: 0012-1630. DOI: 10.1002/dev.420150408.
- [491] Richard Held and Alan Hein. “Movement-Produced Stimulation in the Development of Visually Guided Behavior.” In: *Journal of Comparative and Physiological Psychology* 56.5 (1963), pp. 872–876. ISSN: 0021-9940. DOI: 10.1037/h0040546.
- [492] Nikolai Aleksandrovich Bernstein. *The Co-ordination and regulation of movements*. 1st ed. Oxford: Pergamon Press, 1967.
- [493] Anderson Nascimento Guimarães et al. “Freezing Degrees of Freedom During Motor Learning: A Systematic Review.” English. In: *Motor Control* 24.3 (2020), pp. 457–471. DOI: 10.1123/mc.2019-0060.
- [494] Gabriel Gómez et al. “Simulating development in a real robot. On the concurrent increase of sensory, motor, and neural complexity.” In: (2004).
- [495] Volodymyr Ivanchenko and Robert A. Jacobs. “A Developmental Approach Aids Motor Learning.” In: *Neural Computation* 15.9 (2003), pp. 2051–2065. ISSN: 0899-7667. DOI: 10.1162/089976603322297287.
- [496] M. Alex Meredith and Barry E. Stein. “Visual, auditory, and somatosensory convergence on cells in superior colliculus results in multisensory integration.” In: *Journal of Neurophysiology* 56.3 (1986), pp. 640–662. ISSN: 0022-3077. DOI: 10.1152/jn.1986.56.3.640.
- [497] Nicholas P. Holmes and Charles Spence. “Multisensory Integration: Space, Time and Superadditivity.” In: *Current Biology* 15.18 (2005), R762–R764. ISSN: 0960-9822. DOI: 10.1016/j.cub.2005.08.058.
- [498] Wei Ji Ma et al. “Bayesian inference with probabilistic population codes.” In: *Nature Neuroscience* 9.11 (2006), pp. 1432–1438. ISSN: 1546-1726. DOI: 10.1038/nn1790.
- [499] Florian Walter et al. “Multisensory Integration in the HBP Neurorobotics Platform.” In: *Proceedings of the 28th Annual Conference of the Japanese Neural Network Society*. 2018.
- [500] Marc O. Ernst and Martin S. Banks. “Humans integrate visual and haptic information in a statistically optimal fashion.” In: *Nature* 415.6870 (2002), pp. 429–433. ISSN: 1476-4687. DOI: 10.1038/415429a.
- [501] Seungchul Lee. *Parameter Estimation in Probabilistic Model*. Industrial AI Lab at POSTECH. 2020. URL: https://i-systems.github.io/teaching/ML/iNotes/18_Parameter_estimation.html (visited on 06/30/2021).
- [502] Jakob Jordan et al. “Conductance-Based Dendrites Perform Reliability-Weighted Opinion Pooling.” In: *Proceedings of the Neuro-Inspired Computational Elements Workshop*. NICE '20. New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450377188. DOI: 10.1145/3381755.3381767.
- [503] Giorgio Metta et al. “The iCub Humanoid Robot: An Open Platform for Research in Embodied Cognition.” In: *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*. PerMIS '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 50–56. ISBN: 9781605582931. DOI: 10.1145/1774674.1774683.
- [504] Clearpath Robotics Inc. *HUSKY™. Unmanned Ground Vehicle*. 2020. URL: <https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/> (visited on 07/03/2021).
- [505] Tanguy Fardet et al. *NEST 2.20.1*. Zenodo, 2020. DOI: 10.5281/ZENODO.4018718.

- [506] Tobias Jülg. “Development of a Training Scheduler for Curriculum-Based Reinforcement Learning in Robotics.” Chair of Robotics, Artificial Intelligence and Real-Time Systems. Bachelor’s Thesis. Garching: Technical University of Munich, 2020.
- [507] Joaquin Vanschoren. “Meta-Learning.” In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by Frank Hutter et al. Cham: Springer International Publishing, 2019, pp. 35–61. ISBN: 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5_2.
- [508] Yoshua Bengio et al. “Curriculum learning.” In: *Proceedings of the 26th Annual International Conference on Machine Learning*, the 26th Annual International Conference (Montreal, Quebec, Canada). Ed. by Andrea Danyluk. New York, NY: ACM, 2009, pp. 1–8. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380.
- [509] Martin Riedmiller et al. “Learning by Playing Solving Sparse Reward Tasks from Scratch.” In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4344–4353. URL: <http://proceedings.mlr.press/v80/riedmiller18a.html>.
- [510] Bowen Baker et al. “Emergent Tool Use From Multi-Agent Autocurricula.” In: *Eighth International Conference on Learning Representations (ICLR 2020)*. 2020.
- [511] Max Lungarella and Luc Berthouze. “Adaptivity via alternate freeing and freezing of degrees of freedom.” In: *ICONIP’02. Proceedings of the 9th International Conference on Neural Information Processing : computational intelligence for the E-age : November 18-22, 2002, Orchid Country Club, Singapore*. 9th International Conference on Neural Information Processing (Singapore, Nov. 18–22, 2002). Ed. by Lipo Wang. ICONIP (Conference) and Nanyang Technological University. Singapore: Nanyang Technological University School of Electrical and Electronic Engineering, 2002, pp. 482–487. ISBN: 981-04-7524-1. DOI: 10.1109/ICONIP.2002.1202217. (Visited on 02/17/2020).
- [512] Max Lungarella and Luc Berthouze. “On the Interplay Between Morphological, Neural, and Environmental Dynamics: A Robotic Case Study.” In: *Adaptive Behavior* 10.3-4 (2002), pp. 223–241. ISSN: 1059-7123. DOI: 10.1177/1059712302919993005.
- [513] Luc Berthouze and Max Lungarella. “Motor Skill Acquisition Under Environmental Perturbations: On the Necessity of Alternate Freezing and Freeing of Degrees of Freedom.” In: *Adaptive Behavior* 12.1 (2004), pp. 47–64. ISSN: 1059-7123. DOI: 10.1177/105971230401200104.
- [514] Piero Savastano and Stefano Nolfi. “Incremental Learning in a 14 DOF Simulated iCub Robot: Modeling Infant Reach/Grasp Development.” In: *Biomimetic and Biohybrid Systems* (Berlin, Heidelberg, 2012). Ed. by Tony J. Prescott et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 250–261. ISBN: 978-3-642-31525-1.
- [515] Freek Stulp and Pierre-Yves Oudeyer. “Emergent proximo-distal maturation through adaptive exploration.” In: *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*. 2012, pp. 1–6. DOI: 10.1109/DevLrn.2012.6400586.
- [516] Mark H. Lee et al. “Staged Competence Learning in Developmental Robotics.” In: *Adaptive Behavior* 15.3 (2007), pp. 241–255. ISSN: 1059-7123. DOI: 10.1177/1059712307082085.
- [517] Richard S. Sutton and Andrew Barto. *Reinforcement Learning. An Introduction*. 2nd ed. Adaptive Computation and Machine Learning Series. Cambridge, MA and London: The MIT Press, 2018. 526 pp. ISBN: 0262039249.
- [518] David Silver et al. “Deterministic Policy Gradient Algorithms.” In: *Proceedings of the 31st International Conference on Machine Learning* (Jan. 27, 2014). Ed. by Eric P. Xing and Tony Jebara. PMLR, 2014, pp. 387–395. URL: <http://proceedings.mlr.press/v32/silver14.html>.

-
- [519] Scott Fujimoto et al. “Addressing Function Approximation Error in Actor-Critic Methods.” In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1587–1596. URL: <https://proceedings.mlr.press/v80/fujimoto18a.html>.
- [520] François Chollet. *Keras*. <https://keras.io>. 2015.
- [521] Matthias Plappert. *keras-rl*. 2016. URL: <https://github.com/keras-rl/keras-rl> (visited on 07/11/2021).
- [522] Pierre Manceron. *ikpy. An Inverse Kinematics library aiming performance and modularity*. GitHub. 2021. URL: <https://github.com/Phylliade/ikpy> (visited on 12/19/2021).
- [523] Antonin Raffin. *rl-baselines3-zoo. A training framework for Stable Baselines3 reinforcement learning agents, with hyperparameter optimization and pre-trained agents included*. 2020. URL: <https://github.com/DLR-RM/rl-baselines3-zoo> (visited on 12/18/2021).
- [524] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations.” In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.
- [525] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. URL: <https://arxiv.org/pdf/1207.0580> (visited on 08/08/2020).
- [526] Martin Heisenberg. “What Do the Mushroom Bodies Do for the Insect Brain? An Introduction.” In: *Learning & Memory* 5.1 (1998), pp. 1–10. DOI: 10.1101/lm.5.1.1.
- [527] J. S. de Belle and M. Heisenberg. “Associative odor learning in *Drosophila* abolished by chemical ablation of mushroom bodies.” In: *Science* 263.5147 (1994), p. 692. DOI: 10.1126/science.8303280.
- [528] Larry Dignan. “Why AMD’s triple core Phenom is a bigger deal than you think.” In: *ZDNet 2007* (Sept. 20, 2007). URL: <https://www.zdnet.com/article/why-amds-triple-core-phenom-is-a-bigger-deal-than-you-think/> (visited on 08/08/2020).
- [529] James E. Tomayko. *The Story of Self-Repairing Flight Control Systems*. Ed. by Christian Gelzer. NASA Dryden Flight Research Center, 2003.
- [530] Manuel Bastioni et al. “Ideas and Methods for Modeling 3D Human Figures: The Principal Algorithms Used by MakeHuman and Their Implementation in a New Approach to Parametric Modeling.” In: *Proceedings of the 1st Bangalore Annual Compute Conference*. COMPUTE ’08. New York, NY, USA: Association for Computing Machinery, 2008. ISBN: 9781595939500. DOI: 10.1145/1341771.1341782.
- [531] Karl M. Newell and David E. Vaillancourt. “Dimensional change in motor learning.” In: *Human Movement Science* 20.4 (2001), pp. 695–715. ISSN: 0167-9457. DOI: 10.1016/S0167-9457(01)00073-2.
- [532] Matthias Plappert et al. *Asymmetric self-play for automatic goal discovery in robotic manipulation*. Ed. by OpenAI. 2021. URL: <https://arxiv.org/pdf/2101.04882> (visited on 07/12/2021).
- [533] Manuel Giuliani and Alois Knoll. “Integrating Multimodal Cues Using Grammar Based Models.” In: *Universal Access in Human-Computer Interaction. Ambient Interaction* (Berlin, Heidelberg, 2007). Ed. by Constantine Stephanidis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 858–867. ISBN: 978-3-540-73281-5.
- [534] Alois Knoll and Florian Walter. *Teleported AI. A Cloud Platform for AI-Based Teleprogramming of Robots*. Chair of Robotics, Artificial Intelligence and Real-Time Systems, 2020. URL: <https://mediatum.ub.tum.de/doc/1575022/1575022.pdf>.
- [535] NEST Initiative. *iaf_psc_alpha - Leaky integrate-and-fire neuron model. NEST Simulator User Documentation 3.0*. 2021. URL: https://nest-simulator.readthedocs.io/en/v3.0/models/iaf_psc_alpha.html (visited on 01/08/2022).