Motion representation with spiking neural networks for grasping and manipulation

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
(Dr.-Ing.)

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
DISSERTATION

von

M.Sc.-Ing. Juan Camilo Vásquez Tieck
aus Medellin, Kolumbien

Tag der mündlichen Prüfung:	22. Oktober 2021
Referent:	Prof. Dr.-Ing. Rüdiger Dillmann
Korreferent:	Prof. Dr. Rainer Goebel

Credits

"What motivated explorers?... What inspires an explorer to undertake a voyage with no destination, to search with no objective, to travel with no itinerary other than the uncharted, the unfathomed, the unexpected?... Every explorer I have met has been driven – not coincidentally but quintessentially – by curiosity, by a single-minded, insatiable, and even jubilant need to know."
— Jacques-Yves Cousteau (The Human, the Orchid, and the Octopus)

The research presented in this thesis was carried out at the FZI Research Center for Information Technology in Karlsruhe and was funded by the Human Brain Project (HBP). There are many who helped me and supported me along the way on this crazy journey. I want to take a moment to thank them. This list of acknowledgements can only capture a small fraction of the people I am thankful for. I send my deep thanks to all.

I would like to thank my supervisor Prof. Rüdiger Dillmann for his support and guidance during this research. He gave me enough freedom to explore the field, but also showed me a clear direction to remain in focus. I would also like to thank Prof. Rainer Goebel for his encouragement as well as his insightful feedback which helped me widen my research.

Within the HBP I meet brilliant people and made some great friends. I thank Prof. Wolfgang Maass and his group at Graz for showing me how to implement meaningful Spiking Neural Networks (SNNs) and specially for organizing the Fürberg workshops. I thank Prof. Walter Senn and his group at Bern for the great discussions and for spending time with me modelling grasping networks. I thank Prof. Sander Bohte for sharing his ideas on learning and for showing interest in my research. I thank Prof. Karlheinz Meier and his group at Heidelberg for leading CDP5 and for his inspiring talks. I thank Prof. Sacha van Albada for her mentoring and valuable advice. I thank Prof. Jörg Conradt for organizing the neurocomputation workshop where Jacques and I developed our first serious SNN models. I thank the INI group in Zurich for the Capocaccia summer school, where I spent long nights discussing and hacking SNNs. I would like to specially thank Terry Stewart for sharing his Nengo knowledge, for receiving me in Canada for the summer school, and for our collaboration over the years.

I am fortunate to be a part of the FZI. I want to thank my colleagues for the stimulating discussions, for being critical and honest about my work and giving me great feedback, for the sleepless nights we worked together before deadlines, and for all the fun we had in the last years. I want to thank Arne R. for his leadership and support; Stefan U. for bringing me to the HBP; Sebastian K. and Andreas H. for their advice on science and collaboration; Georg H., Pascal B. and Marc E. for the good vibes; Felix M. and Christian J. for being awesome officemates; Stefan S. for the nice conversations; and the rest of the team for sharing your robotics knowledge. I'm extremely grateful to the students who worked with me, especially those who worked in the motor primitive framework. I want to acknowledge Lea S. and Tristan S. for their work with the robot arm, Heiko D. and Katharina S. for their work with the anthropomorphic hand, and Jacqueline R. for her work the six-legged robot.

Dr. Jacques Kaiser, thank you very much for your friendship and encouragement, you rock *caremonda*! Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. It was an amazing journey together from day one: we shared the office, we worked on hard projects, we went to conferences, we traveled together, we wrote an insane amount of papers, we never gave up on SNNs and the HBP, we did all the teaching, you even made it to our wedding, and we always got each other's back. I learned a lot from you, thank you again.

I wish to express my deepest gratitude to my friends and family for being there for me despite me being distracted and not having much free time. I am forever grateful for your patience and understanding. I hope to have time now to reconnect with each of you. Dear Truji, Polit, Muerzo, Danilindo, Luis, Tati and Alex thanks for keeping me sane, for climbing with me, for providing stimulating discussions, as well as happy distractions to rest my mind outside of my research. I am always thankful to my mother, my father and Alelo for being always there for me. Alex, thank you for the visits and your unconditional company.

Eli Imaginaria, I want to express the immense gratitude I have for you for your patience and support. You are probably the only other person as invested in my research as I am. These sentences in the acknowledgements will never do justice to your understanding and the sacrifices you have made. You gave up a lot for our family, and I really appreciate it. Thank you for putting up with me being sat in the office for hours, being busy for so long, and for sleeping next to me in the couch while I was coding or writing at night. There where ups and downs, but you have always stood by me through all my frustration, my absences, and my impatience. Thank you for your help, support, and love. I am the luckiest person in the world to have you in my life. But most of all, thank you for being my best friend. I love you. Maximo, chiquilin, thank you for filling our life with joy and happiness.

It is time to celebrate!

Abstract (English Version)

Nature takes advantage of millions of years of evolution to generate adaptive physical systems with efficient control strategies. In contrast to conventional robotics, humans do not just plan a motion and execute it; there is instead a combination of multiple control loops working together to move the arm and grasp an object with the hand. With the research on humanoid and biologically-inspired robots, complex kinematic structures and complicated actuator and sensor systems are being developed. These systems are difficult to control and program, and classical robotics methods cannot fully take advantage of their capabilities. Neuroscience research has made much progress towards understanding the different brain regions and their corresponding functions. Nevertheless, most of the models are based on large-scale simulations that focus on reproducing the connectivity and statistical neural activity. This opens a gap in applying different paradigms to validate brain mechanisms and learning principles, and develop functional models to control robots. One promising paradigm is event-based computation using SNNs. SNNs focus on the biological aspects of neurons, replicating the way real neurons work. They are designed for spike-based communication, enabling research on brain-like mechanisms for learning using plasticity. Spike-based communication enables hardware optimizations that allow low energy consumption and fast local operations using neuromorphic chips.

This work proposes different SNNs to perform motion control for manipulation and grasping tasks with a robotic arm and an anthropomorphic hand, based on biologically-inspired functional models of the human brain. A motor primitive is modeled in a parametric way with an activation parameter and a mapping function to the robot kinematics. The topology of the SNNs reflects the kinematic structure of the robot. The robots are controlled using the joint position interface. In order to model complex motions and behaviours, primitives are arranged in different layers in a hierarchy. This allows the combination and parameterization of the primitives and the reuse of low-level primitives for different motions. There are different activation mechanisms for the parameter that controls a motor primitive — voluntary, rhythmic, and reflexes. There are different ways to learn new motor primitives, either online or offline, and either modeling the motion as a function or learning from human demonstration. The SNNs can be integrated with other control systems or combined with other SNNs. Computation of the inverse kinematics or the validation of configurations for planning is not required because the motor primitive space has only feasable motions and contains no invalid configurations.

The scenarios considered for the evaluation are: pointing at different targets, following a trajectory, performing rhythmic or repetitive motions, performing reflexes, and grasping simple objects. Additionally, the arm and hand modeling are combined and extended to model multi-legged locomotion as a generalization use case of the motor primitives control architecture. As applications for an arm (3 Degrees of Freedoms (DoFs)), generating pointing motions and perception-driven target reaching were modeled. To generate pointing motions, one base primitive to point to the center of a plane was combined offline with a set of four correction primitives generating a new trajectory. For perception-driven target reaching, three primitives are combined online during the motion using a perception signal of a target. As applications for a five-finger hand (9 DoFs), individual finger activations and soft-grasping with compliant control were modeled. The grasping motions are modeled with motor primitives in a hierarchy, with finger primitives representing synergies between joints and hand primitives representing different affordances coordinating the fingers. Two reflexes are added for each finger: one to activate or stop the motion with contact and one to activate the compliant controller.

This approach provides flexibility, as motor primitives can be reused, parameterized, and combined in different ways. New primitives can be defined or learned. A key aspect of this thesis is that in contrast to deep learning and end-to-end learning methods, the SNNs do not require huge datasets to learn new motions. Using motor primitives, the same modeling approach can be used for different robots by redefining the mapping of the primitives to the robot kinematics. The experiments showed that by using motor primitives, the motor control could be simplified for manipulation, grasping, and locomotion. The use of SNNs for robotics applications is still a point of discussion. There is no state-of-the-art learning algorithm, there is no framework similar to those for deep learning, and the parametrization of SNNs is an art. Nevertheless, robotics applications – like manipulation and grasping – can provide benchmarking tasks and realistic scenarios to validate neuroscience models. Additionally, robotics can take advantage of the capabilities of event-based computation with SNNs and neuromorphic hardware. A physical imitation of a biological system implemented entirely with SNNs and evaluated with real robots can provide new insights into how humans perform motor control and sensor processing, and how it can be applied to robotics. Model-free motion controllers, inspired by human brain mechanisms, can improve the way robots are programmed by making the control more adaptive and flexible.

Abstract (German Version)

Die Natur bedient sich Millionen von Jahren der Evolution, um adaptive physikalische Systeme mit effizienten Steuerungsstrategien zu erzeugen. Im Gegensatz zur konventionellen Robotik plant der Mensch nicht einfach eine Bewegung und führt sie aus, sondern es gibt eine Kombination aus mehreren Regelkreisen, die zusammenarbeiten, um den Arm zu bewegen und ein Objekt mit der Hand zu greifen. Mit der Forschung an humanoiden und biologisch inspirierten Robotern werden komplexe kinematische Strukturen und komplizierte Aktor- und Sensorsysteme entwickelt. Diese Systeme sind schwierig zu steuern und zu programmieren, und die klassischen Methoden der Robotik können deren Stärken nicht immer optimal ausnutzen. Die neurowissenschaftliche Forschung hat große Fortschritte beim Verständnis der verschiedenen Gehirnregionen und ihrer entsprechenden Funktionen gemacht. Dennoch basieren die meisten Modelle auf groß angelegten Simulationen, die sich auf die Reproduktion der Konnektivität und der statistischen neuronalen Aktivität konzentrieren. Dies öffnet eine Lücke bei der Anwendung verschiedener Paradigmen, um Gehirnmechanismen und Lernprinzipien zu validieren und Funktionsmodelle zur Steuerung von Robotern zu entwickeln. Ein vielversprechendes Paradigma ist die ereignis-basierte Berechnung mit SNNs. SNNs fokussieren sich auf die biologischen Aspekte von Neuronen und replizieren deren Arbeitsweise. Sie sind für spike-basierte Kommunikation ausgelegt und ermöglichen die Erforschung von Mechanismen des Gehirns für das Lernen mittels neuronaler Plastizität. Spike-basierte Kommunikation nutzt hoch parallelisierten Hardware-Optimierungen mittels neuromorpher Chips, die einen geringen Energieverbrauch und schnelle lokale Operationen ermöglichen.

In dieser Arbeit werden verschiedene SNNs zur Durchführung von Bewegungssteuerung für Manipulations- und Greifaufgaben mit einem Roboterarm und einer anthropomorphen Hand vorgestellt. Diese basieren auf biologisch inspirierten funktionnalen Modellen des menschlichen Gehirns. Ein Motor-Primitiv wird auf parametrische Weise mit einem Aktivierungsparameter und einer Abbildungsfunktion auf die Roboterkinematik übertragen. Die Topologie des SNNs spiegelt die kinematische Struktur des Roboters wider. Die Steuerung des Roboters erfolgt über das Joint Position Interface. Um komplexe Bewegungen und Verhaltensweisen modellieren zu können, werden die Primitive in verschiedenen Schichten einer Hierarchie angeordnet. Dies ermöglicht die Kombination und Parametrisierung der Primitiven und die Wiederverwendung von einfachen Primitiven für verschiedene Bewegungen. Es gibt verschiedene Aktivierungsmechanismen für den Parameter, der ein Motorprimitiv

steuert — willkürliche, rhythmische und reflexartige. Außerdem bestehen verschiedene Möglichkeiten neue Motorprimitive entweder online oder offline zu lernen. Die Bewegung kann entweder als Funktion modelliert oder durch Imitation der menschlichen Ausführung gelernt werden. Die SNNs können in andere Steuerungssysteme integriert oder mit anderen SNNs kombiniert werden. Die Berechnung der inversen Kinematik oder die Validierung von Konfigurationen für die Planung ist nicht erforderlich, da der Motorprimitivraum nur durchführbare Bewegungen hat und keine ungültigen Konfigurationen enthält.

Für die Evaluierung wurden folgende Szenarien betrachtet, das Zeigen auf verschiedene Ziele, das Verfolgen einer Trajektorie, das Ausführen von rhythmischen oder sich wiederholenden Bewegungen, das Ausführen von Reflexen und das Greifen von einfachen Objekten. Zusätzlich werden die Modelle des Arms und der Hand kombiniert und erweitert, um die mehrbeinige Fortbewegung als Anwendungsfall der Steuerungsarchitektur mit Motorprimitiven zu modellieren. Als Anwendungen für einen Arm (3 DoFs) wurden die Erzeugung von Zeigebewegungen und das perzeptionsgetriebene Erreichen von Zielen modelliert. Zur Erzeugung von Zeigebewegungen wurde ein Basisprimitiv, das auf den Mittelpunkt einer Ebene zeigt, offline mit vier Korrekturprimitiven kombiniert, die eine neue Trajektorie erzeugen. Für das wahrnehmungsgesteuerte Erreichen eines Ziels werden drei Primitive online kombiniert unter Verwendung eines Zielsignals.

Als Anwendungen für eine Fünf-Finger-Hand (9 DoFs) wurden individuelle Fingeraktivierungen und Soft-Grasping mit nachgiebiger Steuerung modelliert. Die Greifbewegungen werden mit Motor-Primitiven in einer Hierarchie modelliert, wobei die Finger-Primitive die Synergien zwischen den Gelenken und die Hand-Primitive die unterschiedlichen Affordanzen zur Koordination der Finger darstellen. Für jeden Finger werden zwei Reflexe hinzugefügt, zum Aktivieren oder Stoppen der Bewegung bei Kontakt und zum Aktivieren der nachgiebigen Steuerung.

Dieser Ansatz bietet enorme Flexibilität, da Motorprimitive wiederverwendet, parametrisiert und auf unterschiedliche Weise kombiniert werden können. Neue Primitive können definiert oder gelernt werden. Ein wichtiger Aspekt dieser Arbeit ist, dass im Gegensatz zu Deep Learning und End-to-End-Lernmethoden, keine umfangreichen Datensätze benötigt werden, um neue Bewegungen zu lernen. Durch die Verwendung von Motorprimitiven kann der gleiche Modellierungsansatz für verschiedene Roboter verwendet werden, indem die Abbildung der Primitive auf die Roboterkinematik neu definiert wird. Die Experimente zeigen, dass durch Motorprimitive die Motorsteuerung für die Manipulation, das Greifen und die Lokomotion vereinfacht werden kann. SNNs für Robotikanwendungen ist immer noch ein Diskussionspunkt. Es gibt keinen State-of-the-Art-Lernalgorithmus, es gibt kein Framework ähnlich dem für Deep Learning, und die Parametrisierung von SNNs ist eine Kunst. Nichtsdestotrotz können Robotikanwendungen - wie Manipulation und Greifen - Benchmarks und realistische Szenarien liefern, um neurowissenschaftliche

Modelle zu validieren. Außerdem kann die Robotik die Möglichkeiten der ereignisbasierten Berechnung mit SNNs und neuromorpher Hardware nutzen. Die physikalische Nachbildung eines biologischen Systems, das vollständig mit SNNs implementiert und auf echten Robotern evaluiert wurde, kann neue Erkenntnisse darüber liefern, wie der Mensch die Motorsteuerung und Sensorverarbeitung durchführt und wie diese in der Robotik angewendet werden können. Modellfreie Bewegungssteuerungen, inspiriert von den Mechanismen des menschlichen Gehirns, können die Programmierung von Robotern verbessern, indem sie die Steuerung adaptiver und flexibler machen.

Contents

1. Introduction

1.1. Motivation

Nature takes advantage of millions of years of evolution to generate efficient control strategies, but they are difficult to understand and replicate. Humans developed advanced sensorimotor capabilities thanks to a combination of a flexible and adaptive body with efficient control provided by the nervous system (Pfeifer et al. [169]). The human body is very complex; it has kinematic redundancy, hands with haptic sensing, advanced vision, and muscles with proprioception able to perform compliant control. On top of it, the brain orchestrates the nervous system to control the body. The human capabilities to perform coordinated motions and interact with objects remain unmatched by robots, for example: walking, grasping unknown objects, hitting a tennis ball, playing the piano. In contrast to conventional robotics, humans do not just plan a motion and execute it; there is instead a combination of multiple control loops working together to move the arm and grasp an object with the hand. The arm and hand motions can quickly be adapted if the object moves or deforms based on sensory feedback. Humans can easily learn and remember motions already executed to modify and reuse them during execution based on the context. For example, to manipulate and grasp both known and unknown objects, perform in-hand manipulation, and interact with objects in motion. Humans excel in terms of prediction, adaptation, flexibility, and error tolerance while having very low energy consumption compared to robots.

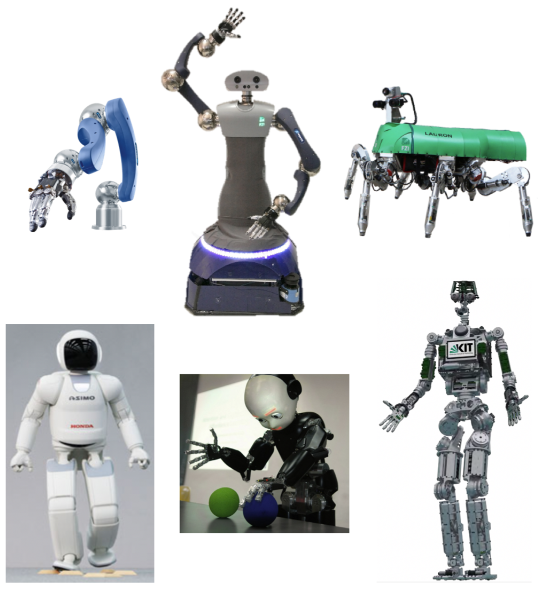
Today, robots are an essential part of the manufacturing industry, and robotics applications are increasing in almost every domain, providing new complex and challenging tasks. In such diverse scenarios, the ability to learn and to adapt efficiently and robustly is required. This ability is fundamental for humanoid and service robots operating in an environment shaped for humans and interacting with objects designed for human ergonomics.

There is a wide variety of robotic systems with advanced hardware, redundant components, and compliant control. Humanoids and other biologically-inspired robots are some examples (see Figure 1.1a). Nevertheless, the programming of these robots is complex, and classical robot programming methods are not always suitable to take advantage of their capabilities. In classical robotics, the problem of motion control is solved by calculating the Inverse Kinematics (IK) for the target point, then validating the configuration, and finally planning the trajectory. The complexity of the problem increases with the number of joints and Degrees of Freedom (DoF), making computational expensive the development of closed mathematical models and extensive exploration methods. An anthropomorphic five-finger robotic hand (Schunk [194] and ShadowRobot [197]) offers a great variety of grasping possibilities. Nevertheless, in industry, most robotic applications are designed for simple vacuum, 2-finger, or custom-made grippers (Wolf et al. [222]) which have few DoFs. These solutions are suitable and affordable for production applications but lack adaptability in other environments where grasping novel objects without knowing their exact properties is required. Deep Learning (DL) has the potential to solve most of these problems and limitations. Nevertheless, despite recent success, DL also has some drawbacks which can not be overlooked and are impractical for real robotics applications. To train a system based on DL a lot of training data and simulation time is required. Depending on the learning method also complex fitness and reward functions are needed. In Levine et al. [147], 800.000 samples were collected to train a robotic arm to perform reaching and grasping, and in Andrychowicz et al. [45] almost 400 computers are used to learn in-hand manipulation.

In contrast to that, how does biology solves these problems? How can the human brain control a complex system as the human body in such an efficient and adaptive way (see Figure 1.1b). The mechanism of how movement is represented and executed in biology is an active field of research. Neuroscience research has made much progress in understanding the different brain regions and their corresponding functions. A widely accepted hypothesis is that the central nervous system uses a small number of motor building blocks (Bernstein [54]) that are combined to produce motion (Bizzi et al. [55] and d'Avella et al. [84]). These building blocks are formed by *muscle synergies* (d'Avella et al. [84]) and are called *motor primitives* (Chinellato et al. [68]). In this context, the term synergy refers to the coupling of motor activation of different joints. Neuroscientists have found that these motor primitives are organized in a hierarchy and are combined by the Central Nervous System (CNS) to compose complex motions (Bizzi et al. [55]). The activation of the motor primitives can be made with different activation modalities, and it can change based on sensor feedback (Churchland et al. [70] and Byrne et al. [64]). There are studies on the evidence of muscle synergies for reaching (d'Avella et al. [82] and Scott [195]) and for grasping (Santello et al. [186] and Sburlea et al. [189]).

These insights from neuroscience have been successfully applied in robotics, for instance with the concepts of the *dynamic movement primitives* (Ijspeert et al. [129]) and the *eigengrasps* (Ciocarlie [72]). Nevertheless, robotics still relies mainly on the classical model-based methods. The existing models from neuroscience are not designed with functionality in mind and can not control a robot. The problem is that the models (see Figure 1.1b) are very complex, require a lot of computational power, and focus mainly on reproducing biological data or replicating the statistics of the neural activity and connectivity of brain areas (Markram et al. [155]).

Neuroscience principles can be applied to control complex biological inspired robots,


(a)

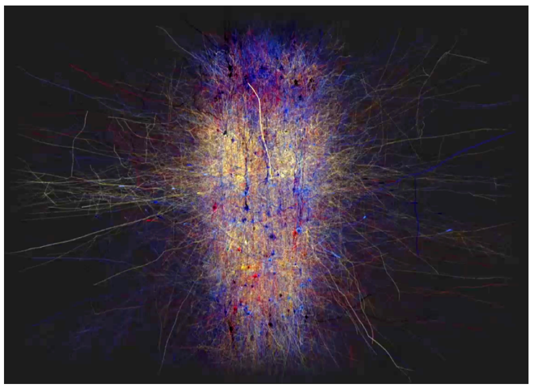

(b)

Figure 1.1.: Motivation. (a) Different biologically-inspired robots: Schunk SVH 5-finger hand (Schunk [194]), HoLLiE (Hermann et al. [123]), LAURON V (Roennau et al. [180]), ASIMO (Shigemi [200]), iCub (Metta et al. [158]) and ARMAR-4 (Asfour et al. [48]). (b) Digital reconstruction and simulation of a part of a rat's neocortex (Markram et al. [155]).

and robots can be used to validate and understand brain mechanisms and learning processes. These models coming from neuroscience need to be simplified first and then implemented in a way suitable for robotics. This opens a gap in using different paradigms to validate brain mechanisms and learning principles and develop functional models to control robots. One such paradigm is event-based computation using Spiking Neural Networks (SNNs). SNNs are modeled closer to the biological aspects of real neurons, replicating the way real neurons work (Maass [151]). They are designed for spike-based communication, enabling research on the brain's learning mechanisms and information representation. SNNs can encode temporal information in their signals and operate in continuous time. With the use of neuromorphic hardware (Furber et al. [104], Pfeil et al. [171], and Höppner et al. [127]), SNNs can be scaled up to take advantage of the characteristics of event-based computation with great power efficiency (Zambrano et al. [228]).

This work focuses on the biological concepts for motion representation and generation using motor primitives representing synergies as building blocks. It incorporates different motion activation modalities, reflexes, and adaptive control. Another key element is how motion is represented in a hierarchy in the nervous system that allows the reuse and combination of different motions.

1.2. Problem statement

The main goal of this thesis is to study SNNs to perform motion control for manipulation and grasping tasks with a robotic arm and hand (see Figure 1.2). The approach is based on bio-inspired functional models of the human brain. Five objectives describe the main goal:

1. Model motions with brain-inspired mechanisms using SNNs.
2. Combine simple motions to generate complex motions.
3. Activate and adapt motions based on sensor feedback.
4. Coordinate the motion for robots with multiple kinematic chains.
5. Generalize and extend the modeling to different robots and combine it with other control systems.

Accordingly, the motion representation is hierarchical and model-free. The motion generation does not require calculating the IK and the validation of configurations for planning. The methods proposed in this thesis are evaluated in simulation and with real robots using an industrial robot arm with an anthropomorphic five-finger hand in various manipulation and grasping scenarios. The evaluation experiments are: pointing at different targets, following a trajectory, performing rhythmic or repetitive motions, and grasping simple objects. Additionally, to demonstrate the potential of the modeling approach, the mechanisms proposed to represent the motion of the robot arm and the hand are combined and extended for multi-legged locomotion.

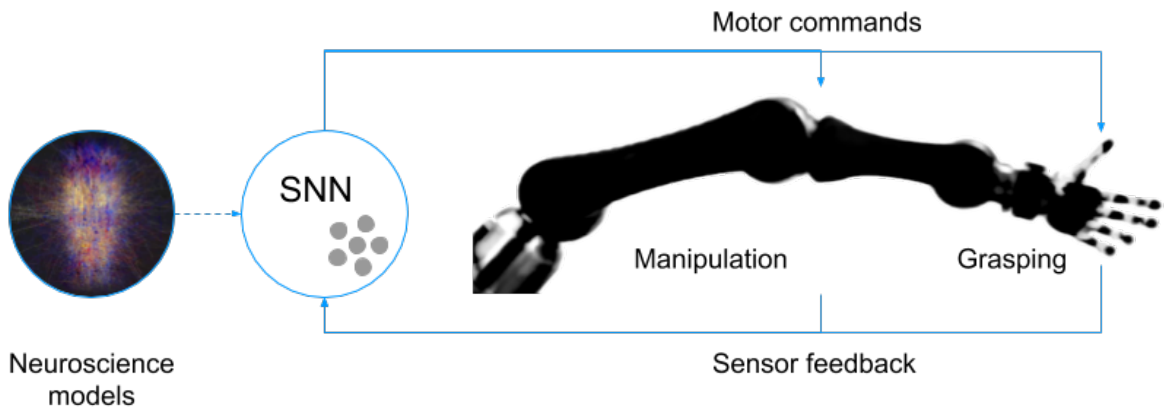


Figure 1.2.: Problem definition. Implement SNNs to control different robots in closed-loop for manipulation and grasping using brain-inspired mechanisms.

1.3. Research questions

This work builds on the research of the different processes involved in low-level movement generation. In particular, the modeling of motion with motor primitives, the parameterization and combination of motions, the different activation modalities, the reflex mechanisms to activate or inhibit motions, and the adaptation of motions according to sensor feedback. The key questions to develop a system for robot motion control in the context of biologically motivated SNNs are:

- How does the central nervous system store, remember, and execute motions?
- Which types of network topologies and learning mechanisms with SNNs are suitable to control robots?
- How can motions be modeled with motor primitives to represent joint synergies using SNNs?
- How to reuse and parameterize existing motor primitives to generate complex motions?
- How to activate and adapt motor primitives based on sensor feedback or reflexes?
- How to coordinate and synchronize the motion of multiple kinematic chains at the same time?

1.4. Key contributions

The key contributions of this thesis are summarized in the following and they are organized by chapter.

- **Modeling and generating motion with SNNs using motor primitives**: The building blocks for modeling motion with motor primitives and robot motion control with SNNs are introduced. This includes the schemas for encoding and decoding spike information with the closed-loop control pipeline and the communication interfaces.A motor primitive is modeled in a parametric way with an activation parameter and a mapping function.The mapping of a motor primitive to the robot kinematics is robot specific, and it can be modified. The topology of the SNNs reflects the kinematic structure of the robot. There are different activation mechanisms for the parameter that controls a motor primitive. It can be activated either voluntary or intentional, rhythmic or repetitive, or as a reflex.There are also neural circuits for contact detection and selective disinhibition.In order to model complex motions and behaviours, motor primitives are arranged in different layers in a hierarchy.This structure allows the combination and parameterization of different motor primitives. Thus, the reuse of low-level primitives for different motions is possible. There are different ways to learn new motor primitives, either online or offline, and either modeling the motion as a function or using human demonstrated motions.These building blocks are combined and evaluated for robot grasping and manipulation and multi-legged locomotion as a generalization use case.

- **Combination and activation of motor primitives to control a robotic arm**: The motion of a robot arm is modeled with SNNs using motor primitives to control three joints — two in the shoulder and one in the elbow. The experiments show that by using motor primitives, the control can be simplified, and planning and calculating the IK or the Jacobian is not necessary. A SNN that allows activation, combination, and parameterization of motor primitives is proposed.The network is based on motor primitives in a hierarchy to generate complex arm motions using different activation signals. This method shows interesting properties of biological systems as different primitives can be activated in different ways simultaneously. Additionally, a neural intercircuit is proposed to detect contact and trigger a reflex. A SNN is presented for pointing motions with a target on a plane. The control architecture is based on recent theories about motion generation in the motor cortex.The network is trained incrementally to learn a base pointing motion and four correction primitives. The activation of the correction primitives depends on the target's position on the plane, and it is performed with selective disinhibition. The network can pre-shape motions and generate new pointing trajectories before execution. The motions are used to control a real humanoid robot in real-time with predictive control in open-loop. A SNN to perform online target reaching in 3D space is proposed extending the architecture for multimodal activation.Three motor primitives are defined to move the robot arm in three different directions left-right, up-down and far-near. The target is represented as a discrete error signal indicating the corrective direction of the motion. The network can combine the motions online using the error signal to drive the three motor primitives to provide continuous control for the robot in a closed-loop. Experiments with a robot arm in simulation are presented to cover the whole working space extensively by going to different points and returning to the start point, and going to boundary targets and random points in sequence.

- **Coordination of motor primitives and compliant control for an anthropomorphic robotic hand**: The motion of the 5-finger hand is modeled with SNNs using motor primitives as a network that coordinates finger sub-networks. The motion of each finger is modeled independently, in analogy to the robot arm. It is possible to model grasping affordances by coordinating the activation of the fingers and using reflex activations to detect contact and adapt the grasping motions to the object. A proof of concept of a biologically inspired SNN control architecture for learning with Spike Time Dependent Plasticity (STDP) different types of grasp motions is proposed.The finger motions reflect synergies between the joints, and reflexes stop the motion of individual fingers if there is contact. With a hierarchy of a hand sub-network coordinating finger sub-networks, it is possible to reuse and combine motor primitives of individual finger movements. A SNN that activates motion reflexes of a robotic hand based on human EMG data extending the architecture for grasping motions is presented.The network classifies the EMG signals to detect which finger was activated. Based on the classification, single-finger reflexes are triggered. The finger reflexes are modeled with motion primitives and control an anthropomorphic 5-finger robot hand. Finally, a SNN that organizes motor primitives in a hierarchy of joints, fingers, reflexes, and grasping affordances to perform soft-grasping is proposed.The compliant control is implemented in the same SNN using a cascaded PI effort controller triggered by the contact detection intercircuit. Soft-grasping is mainly made with mechanical features in the robot hardware, but not all robots are hardware compliant and do not have force sensors, like the hand used in the experiment. This experiment combines the elements for grasping motions and triggering finger reflexes, and implements online learning for adaptive control in closed-loop.

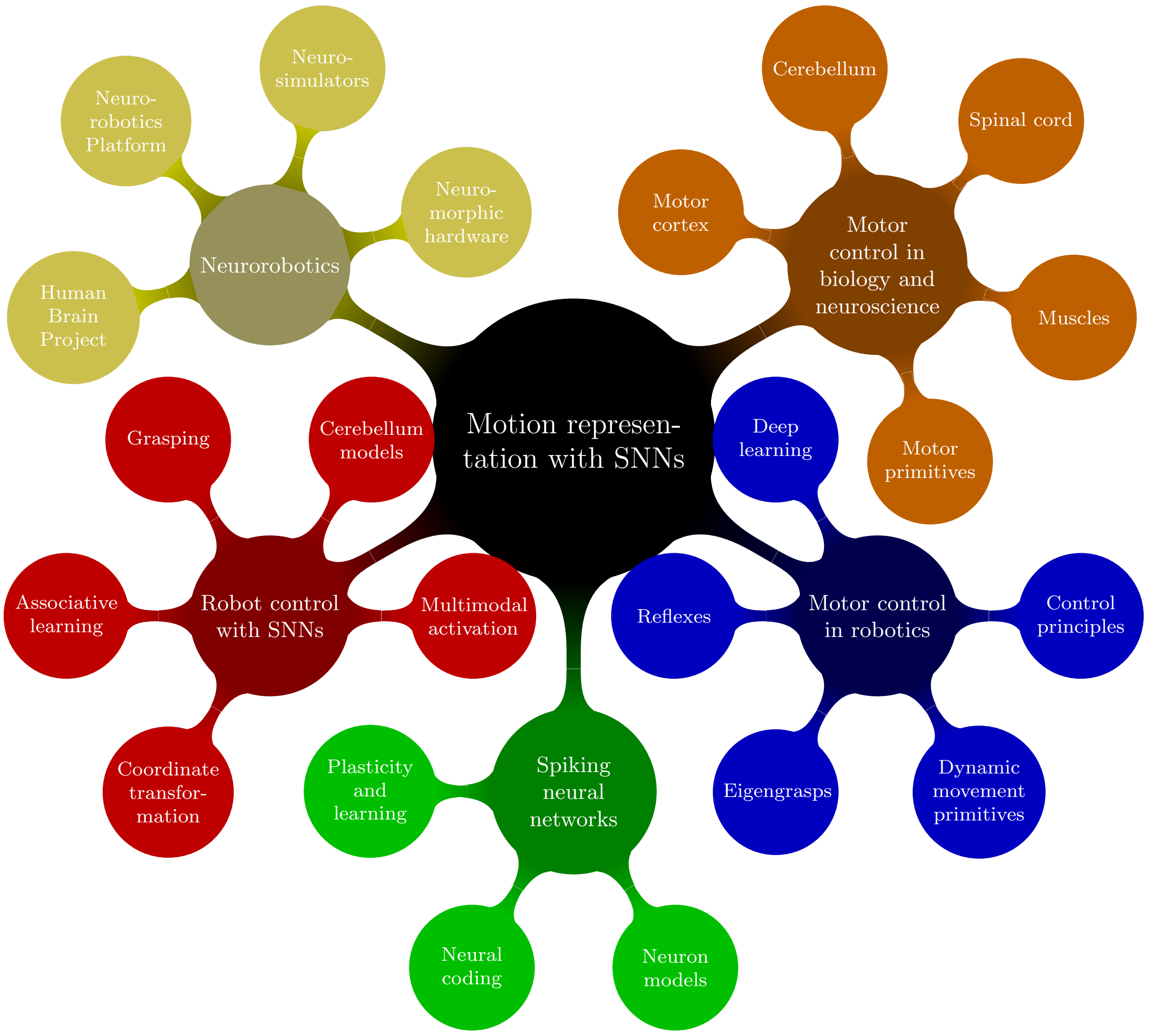
- **Generalization and extension use case for multi-legged locomotion**: The modeling of legs is similar to that of fingers but with more active joints. Multi-legged locomotion is modeled in analogy to grasping with the robot hand. The legs are coordinated with walking behaviours similar to affordances. These experiments show that the modeling approaches with motor primitives for the arm and the hand can be generalized and extended for other motor control tasks with different robots.A system is proposed that combines classical behaviour-based control with motor primitives implemented with SNN for multi-legged locomotion. The walking behaviours are modeled with rhythmic or repetitive activations and controlled by different pattern generators. This approach allows flexibility because motions can be reused and combined in different ways. Different walking behaviours and experiments on a 6-legged walking robot are demonstrated and discussed.

1.5. Structure of this thesis

This thesis is organized in the following way. An overview of related work and its relation to this thesis is presented in Chapter 2. This chapter describes principles for biological motor control, robot motor control, fundamentals for SNN, neurorobotics, approaches with SNN, the Neurorobotics Platform (NRP) from the Human Brain Project (HBP), and an overview on neurosimulators and neuromorphic hardware. The model-free methods for representing motion with SNNs using motor primitives are described in Chapter 3. This chapter presents the coding schemas, the primitive formalization, the different activation modalities, the combination and parameterization of motor primitives, the mechanisms for learning new primitives, the implementation details, and other building blocks. After that, there are three chapters with experiments to apply the methods to arm, hand, and multi-legged motion. The modeling and experiments for robot arm motion are described in Chapter 4. This chapter presents the modeling of the motion of the arm, an experiment for multimodal motion activation, and an experiment for target reaching, and an experiment for motion adaption. The modeling and experiments for hand motion are described in Chapter 5. This chapter presents the modeling of the motion of the hand, an experiment to generate grasping motions, an experiment for triggering finger reflexes, and an experiment for soft-grasping. The modeling and experiments for multi-leg motion as a generalization use case are described in Chapter 6. This chapter presents the modeling of the motion of multiple legs and an experiment for multi-legged locomotion. Finally, the discussion in Chapter 7 presents a summary of the contributions and limitations, the open problems, and the outlook with final considerations.

2. Related Work: motion representation with SNNs

There are many studies on biologically inspired motion control mechanisms and their possible applications to robotics. In this section, research is presented that either details or presents evidence of biological control mechanisms, its successful implementation in robotics, or explores the field of neurorobotics using SNNs.



2.1. Motor control principles in biology and neuroscience

The representation and execution of motion activities are an ongoing research subject in biology and neuroscience (see Figure 2.1). The control of a rich repertoire of motor behaviors like humans and animals is a significant accomplishment of evolution. Mastering the activity of one arm for interactions like target reaching or grasping is already complex due to the number of muscles and joints that have to be coordinated. One important principle of the muscle-motor system is that the control mechanisms require sensory input to predict and execute movements accurately. This control principle applies to low levels of the hierarchy, such as spinal reflexes, and higher levels, such as the cerebellum and the cortex. Motor control shows a considerable increase in complexity with the number of DoF, non-linear dynamics, and sensory delays (d'Avella et al. [81]). The animal and human nervous systems provide efficient methods to overcome such problems enabling a versatile execution of motions adaptive to external circumstances. Thus, neuroscience and robotics can learn in a bi-directional way to understand these complex motor control mechanisms. This section presents principles from biology and neuroscience relevant to robot control.

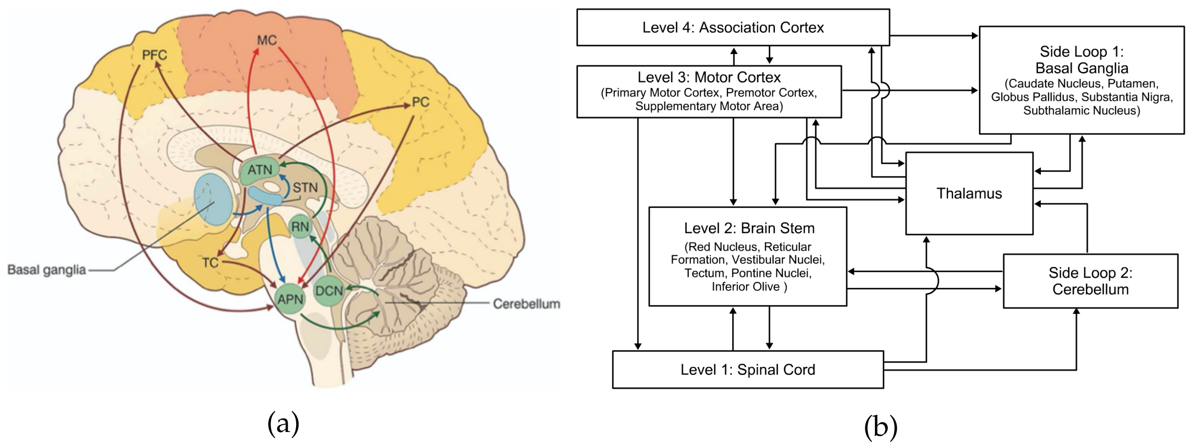

PFC
MC
PC
ATN
STN
RN
TC
APN
DCN
Basal ganglia
Cerebellum
(a)
Level 4: Association Cortex
Level 3: Motor Cortex
(Primary Motor Cortex, Premotor Cortex, Supplementary Motor Area)
Side Loop 1: Basal Ganglia
(Caudate Nucleus, Putamen, Globus Pallidus, Substantia Nigra, Subthalamic Nucleus)
Thalamus
Level 2: Brain Stem
(Red Nucleus, Reticular Formation, Vestibular Nuclei, Tectum, Pontine Nuclei, Inferior Olive)
Side Loop 2: Cerebellum
Level 1: Spinal Cord
(b)

Figure 2.1.: Motor control in biology. (a) Motor control anatomy and connectivity in the brain (D'Angelo et al. [80]). (b) Motor control functional diagram. Schematic representation of different levels and basic signal flow within the motor system hierarchy (Byrne et al. [64]).

2.1.1. The role of the Motor Cortex

The Motor Cortex is the functional center for voluntary (intentional) movement, generating complex movements out of simple motor primitives (Bernstein [54] and Scott [195]). The motor cortex is subdivided into three areas: the primary motor cortex, the premotor cortex, and the supplementary motor area. The primary motor cortex is involved in the execution of voluntary movements. It sends signals down to the spinal cord and encodes the force, direction, and speed of movement (Byrne et al. [64]). The premotor cortex is involved in the selection and preparation of appropriate motor plans for voluntary movements. It sends signals to the primary motor cortex and the spinal cord and performs sensory prediction, context representation, and evaluates motor actions (Byrne et al. [64]). The supplementary motor area is involved in planning complex sequences of movements and coordinating bilateral movements. It selects movements based on remembered sequences of movements and transforms kinematic to dynamic information (Byrne et al. [64]).

Recent studies provide insights into the mechanisms for motion generation in the motor cortex. During motions reaching a target with the arm, the activity in the motor cortex as a whole shows a short but strong rotational component, see Figure 2.2 (Churchland et al. [70] and Russo et al. [184]). Instead of encoding parameters of movement in single neurons, the motor cortex as a whole behaves like a dynamical control system that drives motion (Shenoy et al. [198]). An initial state is produced externally in higher brain areas. The system naturally relaxes while producing motor activity, projected down the spinal cord to inter-neurons and motor-neurons (Churchland et al. [70] and Russo et al. [184]). Neural activity in the motor cortex shows a strong and amplified but stable response to initial activation (Hennequin et al. [119]). There are different hypotheses on the role of the motor cortex in generating voluntary movements. Nevertheless, neural correlations during many different types of arm movements have been found in the motor cortex (Kalaska [134]).

This behavior can be replicated by an Artificial Neural Networks (ANN) with strong recurrent connections balanced by strong inhibitory connections (Hennequin et al. [119]). Neural activity in the resulting network closely resembles activity in the motor cortex and can be used as an engine for complex transient motions (Hennequin et al. [119]). For example, in Ayaso [49] an architecture is proposed to generate motor commands for arm motions and show how learning and adaptation can be achieved by changing the gain.

2.1.2. The role of the Cerebellum

The cerebellum is located directly under the cerebrum on the backside of the head (see Figure 2.3a), and it is involved in the coordination of movement and its fine-tuning, unconscious planning and visualization of movements, and learning motion sequences (Byrne et al. [64]). The cerebellum of humans contains around 69 billion neurons which are about 80% of all neurons in the brain, and its weight is only 154g which is only about 10% of the brain (Gray [114]). The cerebellum has always been associated with the body's motor functions since people with injured cerebellum suffer from motoric and posture dysfunctionalities.

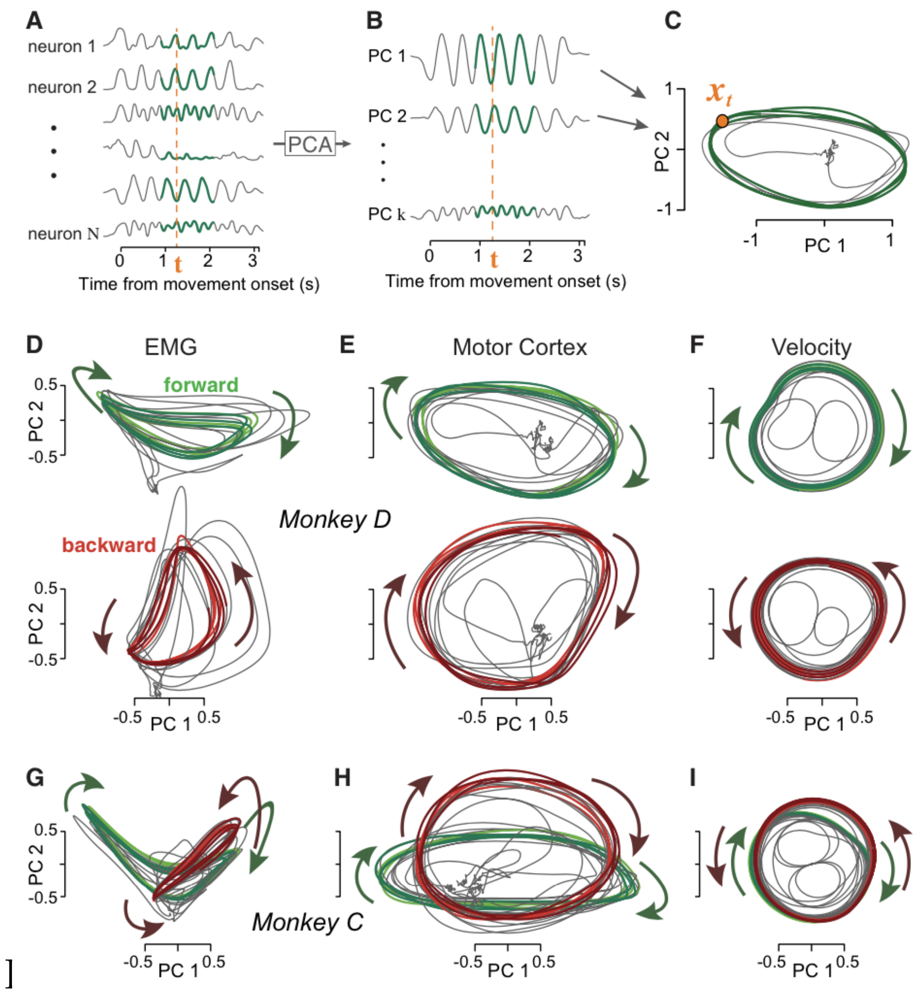
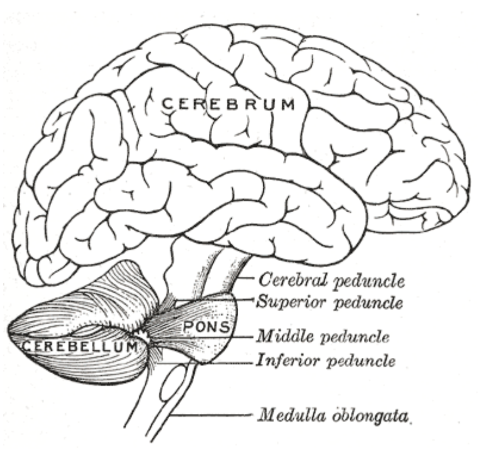
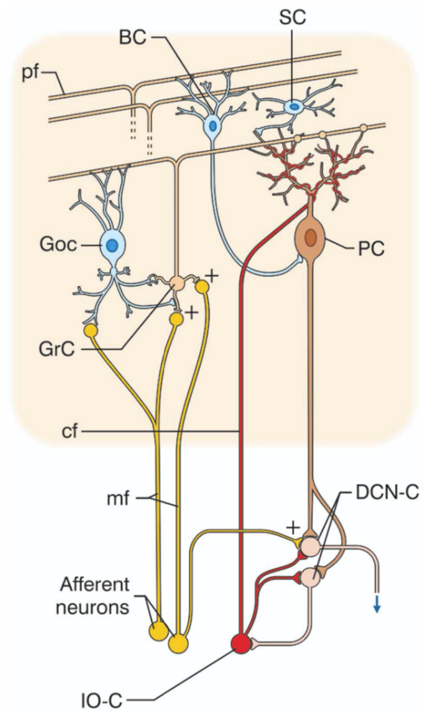


Figure 2.2.: Motor cortex activity for arm movements in monkeys (Russo et al. [184]). (A) Principal Component Analysis (PCA) operates on a population of neuron responses, then the activity is (B) projected onto the principal components to generate (C) a corresponding neural trajectory. For two subjects, the (D) muscle trajectories with the corresponding (E) neural trajectories and (F) hand-velocity trajectories.

The cerebellum does not initiate movements but instead performs corrections and adapts the movement to achieve higher accuracy. The cerebellum is involved in maintaining the balance and posture of the body and the coordination of voluntary movements. It synchronizes the activation timing and the required forces for all contributing muscle groups to accomplish smooth and natural movements (Byrne et al. [64]). Body schema and motion learning are also essential functions of the cerebellum, which allows the body to fine-tune and adapt to previously unknown body movements. The cerebellum behaves like an adaptive feed-forward controller since it can predict the fast movement control of the different body parts (D'Angelo et al. [80]). A classical feedback controller would fail in this task because of its considerable



(a)



(b)

Figure 2.3.: The cerebellum. (a) Anatomy (Gray [114]). (b) Modular cerebellar micro-circuit model (D'Angelo et al. [80]).

sensory delays. Many experiments have shown that the cerebellum learns to adapt in previously unknown scenarios, where new muscle group configurations and their exact activation timing are learned by experience.

There are multiple studies on the cerebellum, but a generally accepted theory shows a modular structure containing multiple instances of the same neural micro-circuit, see Figure 2.3b (D'Angelo et al. [80]). The cerebellum has two main regions, the deep cerebellar nuclei (DCN), where all outputs originate, and the cerebellar cortex, where most of its neurons are. In each layer of the cerebellar cortex, there are different types of neural and interneural cells, but the most distinct ones are the Granule (GrC), Golgi (GoC), and the Purkinje (PC) cells. The cortex connectivity is relatively simple since there are two major input channels in the cerebellum: Mossy fibers (MF) and Climbing fibers (CF).

2.1.3. Other brain areas relevant for motor control

The Somatosensory Cortex is located next to the motor cortex. It is the center for sensory information processing and responds to changes in the body's sensory system. The cortex is topologically subdivided into interconnected regions representing different sensory-motor areas of the body (Penfield et al. [168] and Lorente de No [148]). The Basal Ganglia is responsible for reinforcing successful movements and supports different functions like action selection, initiative, intentional drive, sequential activity planning, and anticipation. Additionally, studies have shown that the human brain combines feedback information from vision and proprioception to control reaching and grasping movements (Saunders et al. [188] and Filimon et al. [102]).

2.1.4. Spinal cord

The Spinal Cord connects the periphery of the body with the brain, and it is located within the vertebral canal (see Figure 2.4). Besides forwarding signals, the spinal cord contains small neural circuits that trigger and execute reflexes. Reflexes are involuntary actions, either purposeful motoric or vegetative, executed as the response to sensorial stimulation. They are controlled by specific neural pathways, the reflex arcs, which have very short reaction times. This is because the response time to external stimuli is directly related to the number of neurons involved, and reflex arcs consist of a sensory neuron and a motor neuron connected to an interneuron in the spinal cord for local processing. A reflex is a non-intentional response to sensor stimulation and can execute or inhibit a motion immediately. An overview of spinal reflexes is given in Byrne et al. [64].

Of interest and relevance to this thesis are the withdrawal and the inhibitory reflexes. The withdrawal or flexor reflex initiates a withdrawal of one arm or leg when triggered by cutaneous and pain receptors (see Figure 2.4). The inhibitory reflex triggers a sudden autogenic inhibition of motor activation that causes muscle relaxation and stops the motion.

The *Reflexxes Motion Library* (Kröger [143]) is a reactive framework based on reflexes, which is a successful example of applying reflexes for robotic motor control. An extended mathematical model for central pattern generators in the spinal cord, generating different motion patterns by changing key parameters, is presented in Nassour et al. [161]. The architecture has separate layers for pattern formation, rhythmic activation generation, and motor neurons.

2.1.5. Synergies between muscles

It is still an open question whether muscles are controlled by the cortical motor system individually or collectively. An indication for the latter is the absence of a natural movement performed by only one muscle (Bizzi et al. [55]). Based on this evidence, the concept of muscle synergies was proposed to understand and model motor control systems. Muscle synergies refer to the temporal or spatial coherent neural activation of a group of muscles (d'Avella et al. [84]). In other words, there is a mechanism

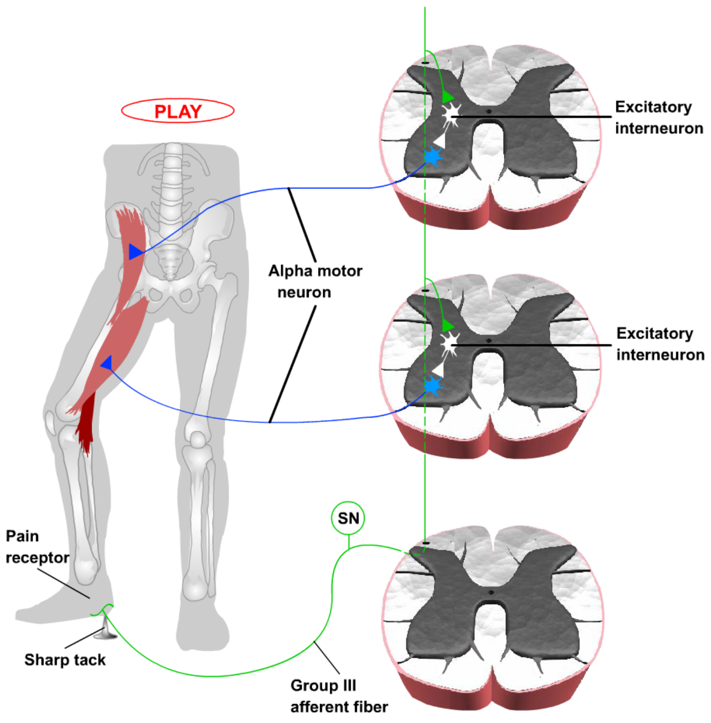


Figure 2.4.: Illustration of the flexor reflex of one leg and the micro-circuit in the spinal cord (Byrne et al. [64]).

for coordinated recruitment of several muscles, sharing the same activation profile. Muscle synergies contribute to a simpler understanding of the motor control system. In particular, the work in Rückert et al. [182] showed that using shared knowledge for related tasks with muscle synergies reduces the dimensionality of motor control. To underline this hypothesis, the works of Saltiel et al. [185] and Bizzi et al. [55] provides evidence of a modular organization of the spinal cord based on muscle synergies. The experiments compare screenings of healthy individuals before and after being detached from the central nervous system (CNS). There is a distinction between time-invariant and time-varying muscle synergies (d'Avella et al. [82]). In general, synergies represent muscle patterns and store the coordination between them. In spatial terms, across muscles, a synergy concerns the mutual activation of a group of muscles (see Figure 2.5). In temporal terms, this concerns either the time-varying or time-invariant relationship between muscles. Thus, synergies with a temporal component capture temporal regularities, and synergies without a temporal component capture spatial regularities in the motor output (d'Avella et al. [82]).

2.1.6. Motor primitives and hierarchical motion representation

One main problem in motor control is how to initiate a sufficient and flexible repertoire of actions combining muscle activity patterns initiated and supported by the

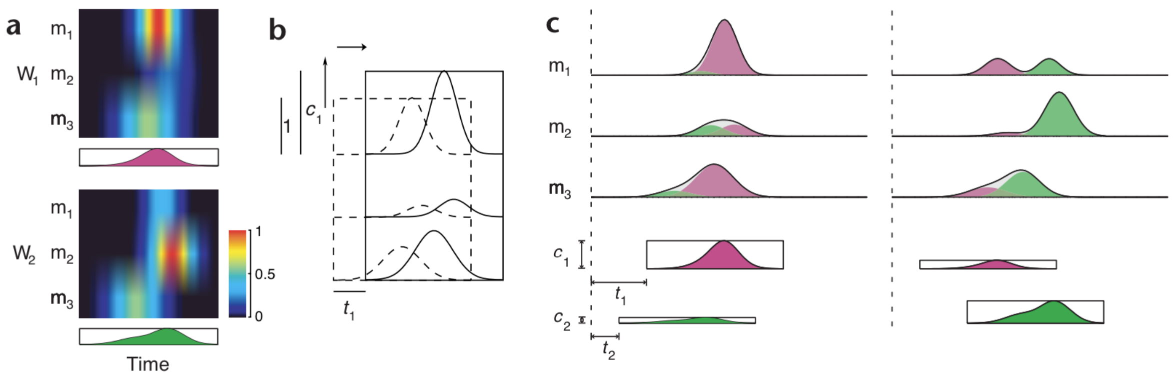


Figure 2.5.: Time-varying synergies model (d'Avella et al. [84]). In this simulated example, two time-varying synergies (a) are scaled in amplitude and shifted in time (b) and then combined to construct two different patterns (c).

CNS. This originates the three major challenges of motor control: dimensionality, caused by many joints and muscles; versatility, due to the multiple motor skills required; and optimality, to ensure minimal energy consumption and performance with low effort and error (d'Avella et al. [84]). A widely accepted hypothesis is that the CNS of vertebrates use a small number of movement modules as motor building blocks (Bernstein [54]). These building blocks are formed by the *muscle synergies* that are active during a movement and are called *motor primitives* (Chinellato et al. [68]). These motor primitives are organized in a hierarchy (d'Avella et al. [81]) and are combined to produce complex motions (Bizzi et al. [55]). Additionally, different modalities can activate the synergies and use sensor feedback to produce adaptive motions (Churchland et al. [70] and Byrne et al. [64]). Therefore, a small set of time-varying muscle synergies can be combined, adjusted in amplitude and timing to enable flexible control of direction and speed (d'Avella et al. [82]).

In Bizzi et al. [55] experiments are presented, indicating that a wide range of different movements can be generated by combining these fundamental building blocks. The experiments extracted five time-varying muscle synergies from muscle patterns in frogs. Different combinations of these synergies are presented to generate different behaviors – jumping, swimming, and walking (Figure 2.6). Konczak [141] reviews the difference between reflexes and primitives and states that motor primitives can descend from primitive reflexes. Evidence of motor primitives in the spinal cord of vertebrates is presented in Flash et al. [103], Hart et al. [118], and Bizzi et al. [55] with neural data from a range of methods and species. The neuron activity in the intermediate zone of the spinal cord resembles motor primitives rather than individual muscle activations (Hart et al. [118]). There are also studies on the evidence of muscle synergies for reaching (d'Avella et al. [82, 83] and Scott [195]) and for grasping (Santello et al. [186], Sburlea et al. [189], and Castellini et al. [67]).

The concepts of motor primitives have been applied in robotics with the *dynamic movement primitives* (Ijspeert et al. [129]) and *eigengrasps* (Ciocarlie et al. [71]).

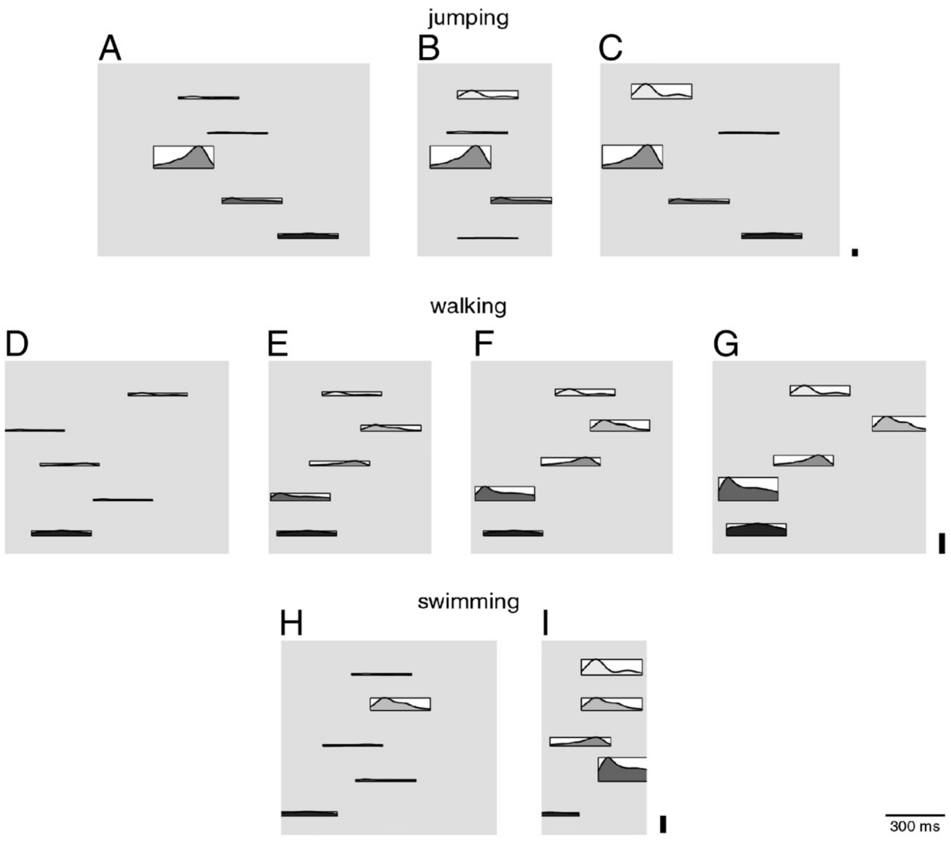


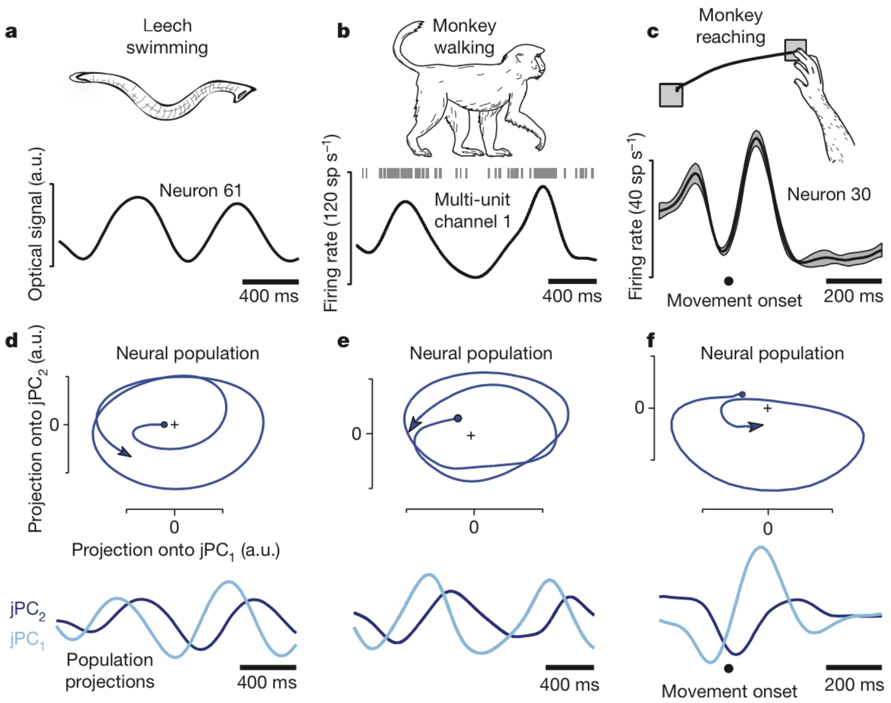
Figure 2.6.: Scaling of synergy recruitment in different frog behaviors (Bizzi et al. [55]). Each panel represents the recruitment of five synergies during a jump (A–C), a walking cycle (D to G), or a swimming cycle (H and I).

2.1.7. Rhythmic motion generation and central pattern generators

Many human motions originate from rhythmically generated patterns and are referred to as rhythmic or repetitive movements. Examples of this are walking, cycling, swiping, but also vegetative motions, like breathing and chewing, At the beginning of the last century, two theories existed about the origin of rhythmic motions. Sherrington [199] suggested a chain of reflexes triggered by sensory feedback as the source. Brown [60], on the other hand, identified local circuits in the spinal cord as contributing to locomotion. He proposed that rhythmic movements are generated centrally through neural circuits without direct feedback control. He introduced the half-center oscillator model, where two inhibitory coupled neural populations produce rhythmic output (Ijspeert [128]). While voluntary or intentional motions are primarily initiated by the brain, the spinal cord is directly involved in coordination and motor control for rhythmic motion generation.

Evidence for rhythmic or oscillating circuits independent of feedback control has been extensively shown in literature for invertebrates and vertebrates. For example, Cohen et al. [74] showed that a spinal cord isolated from the brain of lamprey could produce rhythmic patterns when stimulated electrically. Similar results were achieved in salamanders, frogs, and embryos (Ijspeert [128]). In Churchland et al. [70], the generation of rhythmic and oscillatory activity has been observed in monkeys, whereby repetitive cortical responses could be mapped on limb motions (see Figure 2.7).

The local circuits first identified by Brown [60] are also known as Central Pattern Generators (CPGs). The term central means that sensory feedback from the peripheral nervous system is not required (Ijspeert [128]). CPGs are a set of inter and motor neurons able to create patterns of rhythmic output signals without rhythmic input through sensory feedback. Descending signals from the the motor cortex, the cerebellum, and the basal ganglia, use sensory feedback to coordinate and synchronize these patterns. This initiates and stops rhythmic motions (Nassour et al. [161]). In short, the spinal cord is responsible for generating basic rhythmic patterns, and higher-level centers shape these patterns according to external conditions.



]

Figure 2.7.: Oscillation of neural firing rates during three movement types (Churchland et al. [70]). (a-c) Recorded response of a population of neurons for different movements. (d-f) Projection of the neural population response into the two-dimensional PCA space. The two dimensions are first plotted against each other and then versus time.

2.2. Motor control principles in robotics

Control in robotics is based on system theory and the concept of basic building blocks for motion, dynamic control, and interaction. In the early stages, robot mechanisms were designed to be as stiff as possible to maintain the linearity of the system. This enables the control of simple tasks, such as material transfer. However, the execution of more advanced and complex tasks requires a different control approach, such as force and torque control, and different hardware designs to provide more flexibility. This requires a deeper understanding and modeling of the non-linear dynamics of the robots, which led to the development of more advanced control models, such as non-linear and adaptive. In classical robotics, the problem of motion generation is solved by calculating the IK for the target point, then validating the configuration, and finally planning the trajectory — observe, plan, act.

2.2.1. A short survey on basic control principles

This section presents a survey on basic control models. The main goal of controlling a system with input is to ensure the stability of the system's behavior and output. There are many concepts developed for different control tasks, but they can be classified into three control principles: open-loop, closed-loop, and feed-forward.

Basic control

Open-loop control represents an established control mechanism for controlling a system (see Figure 2.8). The controller receives information about the desired outcome of the system, but its internal state is independent of the process output. Such controllers require complete model knowledge and predictive capabilities.

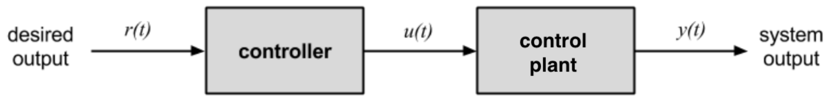


Figure 2.8.: Open-loop control model. The controller receives the desired output as a reference signal $r(t)$. Depending on the system input $u(t)$, the system generates an output $y(t)$.

Closed-loop control mechanisms are in systems theory and in robotics the standard approach to solve a control problem in general. It requires the desired system behavior, model knowledge about the plant, observations to identify the state of the plant, and a controller processing sensor feedback. The system dynamics are to be processed continuously or in discrete intervals. The most typical controller for this approach is the PID controller (see Figure 2.9).

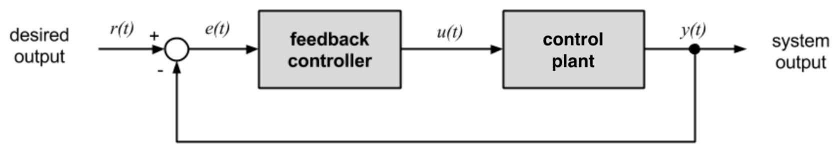


Figure 2.9.: Closed-loop or feedback control model. The error $e(t)$ calculated as a difference between the desired output $r(t)$ and over the feedback loop with the real system output $y(t)$. The controller minimizes the error $e(t)$ until the system meets the desired output.

Feed-forward control model combines the advantage of open-loop and closed-loop control strategies (see Figure 2.10). Feed-forward control is an open-loop control strategy, and it improves the tracking of reference points with the help of feedback controllers. It is used to compensate for noise and suppress unmeasured disturbances present in the real system. The feed-forward control can be used to predict the system's behavior, so a feed-forward controller can react before an error may occur. Usually, this requires an inverse model of the control plant. This shows its limitation since it will not be stable if the control plant changes its parameters or its structure. However, a solution for this issue has been found by using adaptive control concepts.

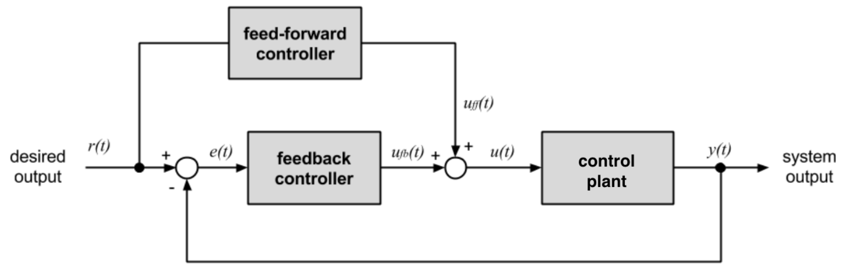


Figure 2.10.: Feed-forward control loop model. The feed-forward control loop is an open-loop control model, which is used together with the closed-loop model. The feedback controller tracks the referent point $r(t)$ by constantly reducing the error $e(t)$. In contrast, the feed-forward controller improves the performance of the control loop by summing the feed-forward $u_{ff}(t)$ with the feedback $u_{fb}(t)$ control signals together.

Basic PID controller principle

The goal of a PID controller (see Figure 2.11) is to minimize the difference of the error value $e(t)$, between the desired set-point $r(t)$ and the feedback value $y(t)$. The PID controller is the most commonly used in the industry and robotics. The controller is part of a feedback control system and consists of three terms: P proportional, I integral, and D derivative. Each of these three terms is defined in Equation (2.1) and contributes to the error correction.

$$u(t) = \underbrace{K_p e(t)}_{\text{P}} + K_i \underbrace{\int_0^t}_{\text{I}} + \underbrace{K_d \frac{\delta e(t)}{\delta t}}_{\text{D}} \tag{2.1}$$

Depending on the application, it is possible to set one or two of the K factors to zero and obtain P, PI, PD, P, or I controllers.

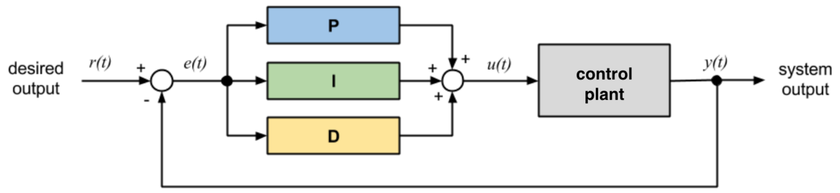


Figure 2.11.: A block diagram of PID Controller in a feedback loop.

Adaptive control

The conventional way of compensating the effects of disturbances and any uncertainty in the system is using a closed-loop control model. However, suppose there are changes in the system dynamics or any non-linear disturbances. In that case, the feedback loop is not enough to deal with these changes since the parametrization of the controller is for specific dynamics only. In contrast, an adaptive control system has two control loops. One represents the feedback loop with the controller and the control plant. Another one represents the adaptive mechanism for adjusting the controller's parameters. It can perform stable control if the control plant changes its parameters, dynamics, or structure (Landau et al. [144]). It has been applied in the control of autopilot systems for planes and ships, and in robotics to consider the dynamics of actuators, joint flexibility, and time-varying uncertainties of the system.

The advances in machine learning provide a new dimension to adaptive control. The use of ANNs has proven to be suitable for learning the non-linearities of a system (Suykens et al. [210]). The feed-forward control model is the most common approach for using an ANN for adaptive control (see Figure 2.12). Based on sensory information, the feed-forward controller learns the behavior of the system and predicts the correction in advance. Through the learning mechanism, the parameters of the controller are constantly improved over time.

2.2.2. Dynamic movement primitives

The modular organization in the spinal cord allows to support motor activities by combining muscle synergies simplifies the movement generation by reducing the re-

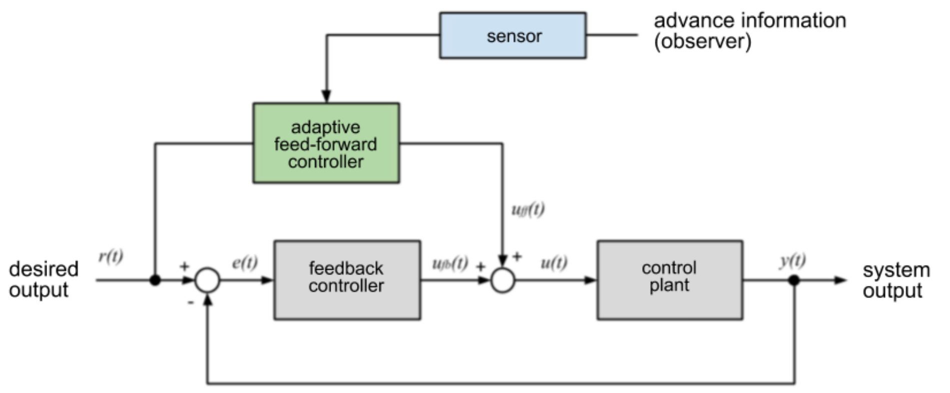


Figure 2.12.: Adaptive feed-forward control model. The feed-forward controller adapts its parameters according to the sensory inputs and the desired output $r(t)$, while the error $e(t)$ to the system output $y(t)$ is reduced and improved by summing the feed-forward $u_{ff}(t)$ with the feedback $u_{fb}(t)$ control signals to get $u(t)$.

quired DoFs (Bizzi et al. [55]). The hypotheses that the CNS hierarchically combines motor primitives to generate complex motions are applied in robotics with the Dynamic Movement Primitivess (DMPs). DMPs were introduced by Schiess et al. [193] and updated by Ijspeert et al. [129]. There is an implementation of DMPs for SNNs using Neuro Engineering Framework (NEF) and Nengo (Nengo) called neural DMPs (NDMPs) (DeWolf et al. [90]). DMPs introduces a representation of movement as a spring-damping dynamical system in which the goal state is an attractor that allows for easily adaptable complex motor behaviors, both for rhythmic and discrete motion. A unique characteristic, distinguishing DMPs from previously introduced frameworks, is that each primitive is a non-linear system. They represent a mathematical formalization of basic units for actions and are well suited for trajectory control and planning. The motivation is to represent adaptive motor actions in a stable and generalized form without manual parameter tuning. This is complex due to parameter sensitivity and phase transitions by changes that imply unpredictable long-term behavior. The main concept behind a DMPs consists of two components (see Figure 2.13). First, a dynamical system with a precisely defined and stable behavior is defined as a linear differential equation. Second, a function to follow a trajectory is defined as a forcing term that can be learned. DMPs either use point attractors as the base system, which results in discrete, point-to-point movements, or they rely on limit cycles leading to rhythmic motions (Schaal [191] and Ijspeert et al. [129]). Additionally, to the first two components, a DMPs has a transformation system (see Figure 2.13). These properties mean that DMPs are more than just a series of activation of muscles. As dynamic attractors, they can guide a system considering noise and perturbations. DMPs have considerable advantages, like straight-forward learning through a stable attractor system, and they can be successfully applied in high-dimensional continuous spaces. There are approaches to learn DMPs, for example, with imitation and reinforcement learning (Kober et al. [140]) or with a DMP representation employing parametrized basis functions (Rückert et al. [182]). However, they are not suited for reusing shared knowledge, so each new task implies learning a new set of parameters.

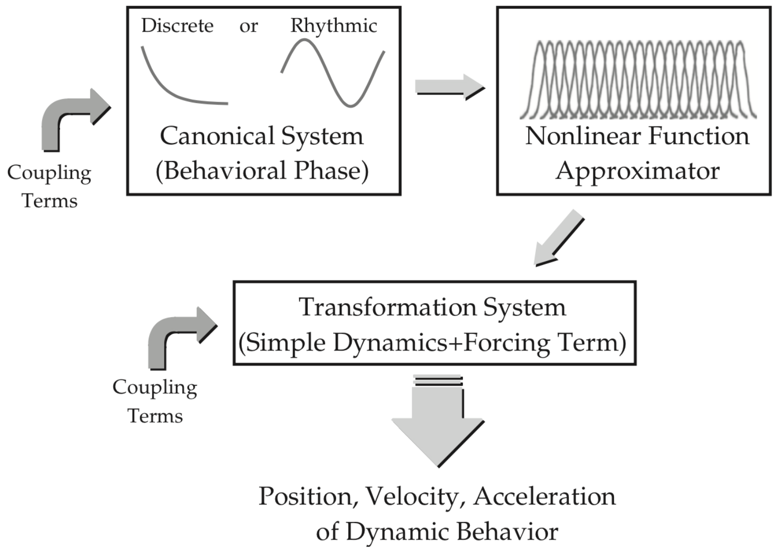

Discrete or Rhythmic
Canonical System (Behavioral Phase)
Nonlinear Function Approximator
Coupling Terms
Transformation System (Simple Dynamics+Forcing Term)
Coupling Terms
Position, Velocity, Acceleration of Dynamic Behavior

Figure 2.13.: Three components are required to create a DMP (Ijspeert et al. [129]). A canonical system, either point attractive or periodic, generating a behavioral phase variable. A non-linear function approximator, generating the forcing term. And a transformation system, a dynamical system easy to analyze and manipulate.

2.2.3. Grasping motion analysis with PCA and eigengrasps

The concepts of synergies and motor primitives have also been applied for grasping with the development of *eigengrasps* (Ciocarlie et al. [71] and Ciocarlie [72]). Santello et al. [186] observed several human subjects shaping their hand as if they were grasping different known objects. A principal component analysis was applied to the recorded data, and they discovered that the first two components represented about 80 percent of the observed variance of the movements. The other components provided additional information about the object to fine-tune the movements. This implies that the 24 DoF of the human hand can be significantly reduced for grasp planning (Cobos et al. [73]). Thus, a few postural synergies and local motor controls for small adjustments can be determined by the control of the hand posture. Ciocarlie et al. [71] and Ciocarlie [72] applied these results to directly reduce the configurations space of the hand and calculate pre-grasp poses by searching in the sub-space for stable grasps. This approach is an eigenvalue decomposition of a data set of pre-grasp, which reduces the search problem to two eigenvectors. An example of this parameterization generated with the *GraspIt! Simulator*[1] (Miller et al. [159]) is presented in Figure 2.14.

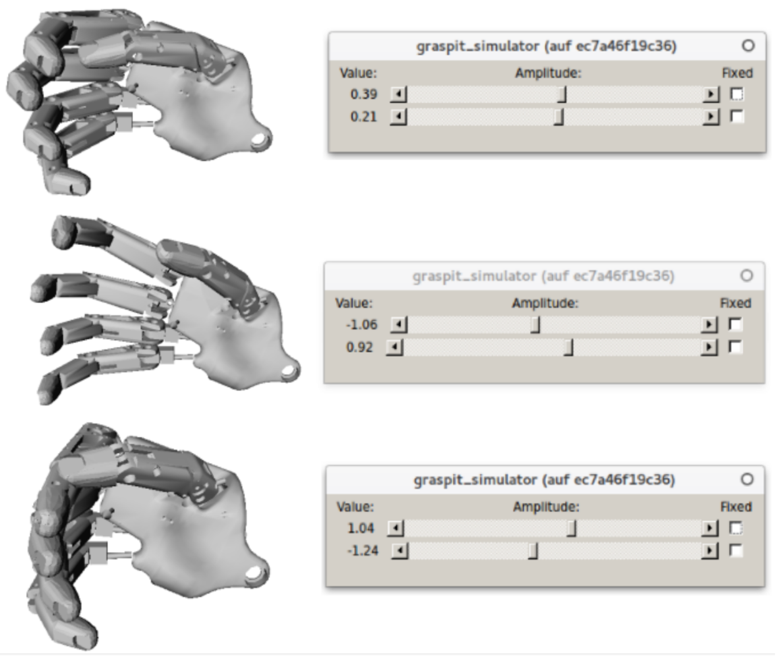

graspit_simulator (auf ec7a46f19c36)
Value:
Amplitude:
Fixed
0.39
0.21
graspit_simulator (auf ec7a46f19c36)
Value:
Amplitude:
Fixed
-1.06
0.92
graspit_simulator (auf ec7a46f19c36)
Value:
Amplitude:
Fixed
1.04
-1.24

Figure 2.14.: Three examples of eigengrasps. The sliders on the right represent the values of the first two eigenvalues of the pre-grasp data.

2.2.4. Reactive robot control using reflexes

The Reflexxes Motion Libraries, developed by Kröger [143], are based on the concept of natural reflexes in robotic motion generation. The approach considers sensor-guided and sensor-guarded motions calculated online during motion execution to complement predefined trajectories and combines both as shown in Figure 2.15. This method models deterministic, instantaneous reactions to sensor input and calculation of motions from any initial state, even during another movement. The underlying concepts for online motion generation within every control cycle, requiring only about a millisecond computation time, are outlined in Kroger et al. [142].

[1] `https://graspit-simulator.github.io`

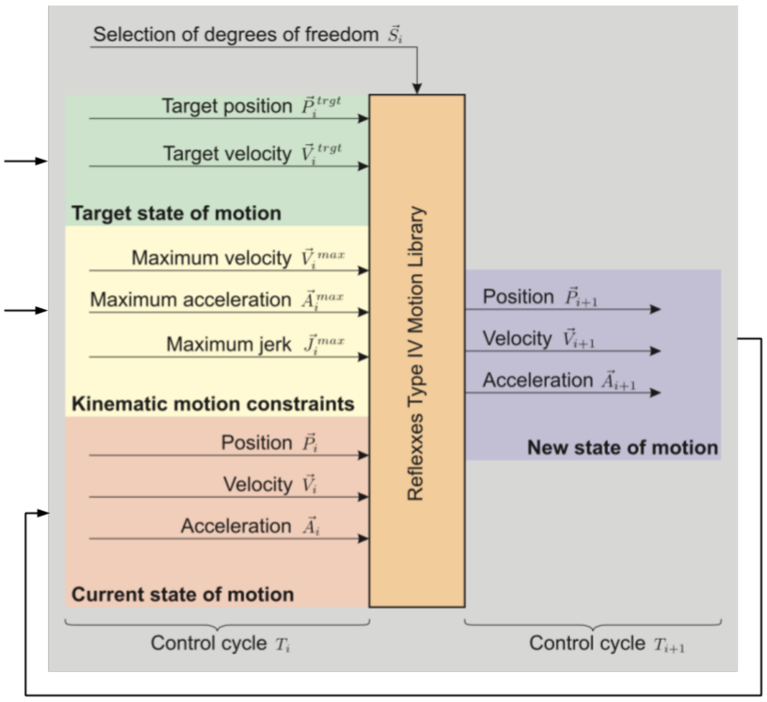
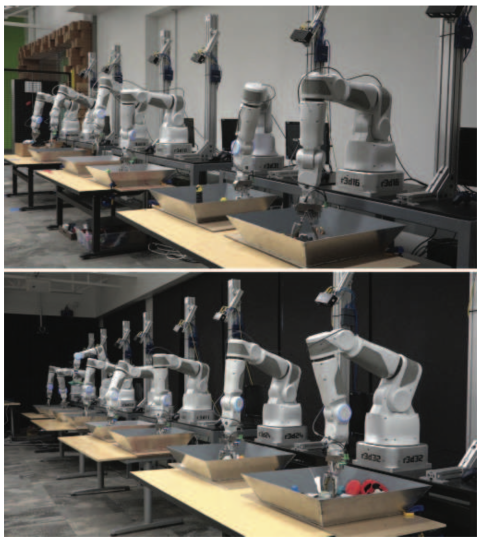

Selection of degrees of freedom $\vec{S}_i$
Target position $\vec{P}_i^{trgt}$
Target velocity $\vec{V}_i^{trgt}$
Target state of motion
Maximum velocity $\vec{V}_i^{max}$
Maximum acceleration $\vec{A}_i^{max}$
Maximum jerk $\vec{J}_i^{max}$
Kinematic motion constraints
Position $\vec{P}_i$
Velocity $\vec{V}_i$
Acceleration $\vec{A}_i$
Current state of motion
Reflexxes Type IV Motion Library
Position $\vec{P}_{i+1}$
Velocity $\vec{V}_{i+1}$
Acceleration $\vec{A}_{i+1}$
New state of motion
Control cycle T_i
Control cycle T_{i+1}

Figure 2.15.: The interface for the Reflexxes Motion Libraries (Kröger [143]). Based on the current state of motion and the kinematic motion constraints, a new state of motion is calculated, which lies precisely on the time-optimal trajectory to reach the desired target state of motion.

2.2.5. Applications of Deep Learning for manipulation and grasping

Different approaches apply DL to learn human-like manipulation and grasping skills for robots. Starke et al. [204] propose a constrained autoencoder model to generate and learn grasp representations based on observed human grasping data. In Andrychowicz et al. [45], domain randomization is performed in simulation to learn in-hand manipulation skills from visual information using Reinforcement Learning (RL). The learned model was successfully transferred to control a physical Shadow Dexterous Hand. For this work, extensive computation is performed with almost 400 computers, each with 16 CPU cores, generating about two years of experience per hour. In Santina et al. [187], a combination of an object classifier with reactive and anticipatory motor primitives are implemented. A human demonstration is used for classification to select the corresponding primitive, and the motion is executed with a soft-gripper. In Levine et al. [147], a convolutional network was trained to perform hand-eye coordination for reaching and grasping. The network estimates the success probability for a grasp and then controls the physical manipulator. For this work, about 800.000 samples were collected using 14 robotic manipulators.


(a)

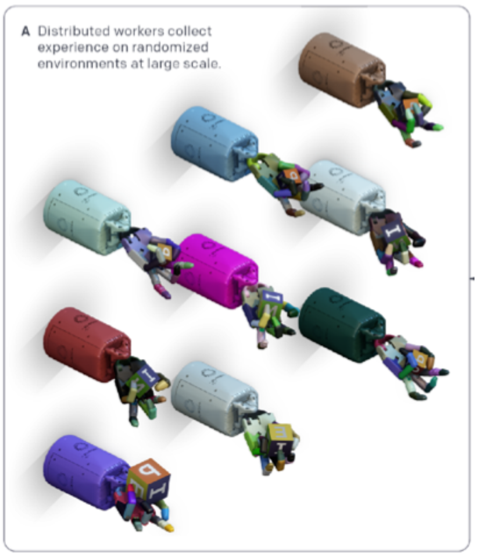

A Distributed workers collect experience on randomized environments at large scale.
(b)

Figure 2.16.: Two different approaches to learn stable grasping skills. (a) The approach from Google, is based on a large-scale data collection setup for real robot experiments, consisting of 14 robotic manipulators (Levine et al. [147]). (b) The approach from OpenAI is based on a large distribution of simulations with randomized parameters and appearances to collect data for both the control policy and vision-based pose estimation (Andrychowicz et al. [45]).

DL has the potential to solve most robotics problems and limitations. Nevertheless, despite recent success, DL also has some important drawbacks which can not be overlooked for real robotics applications. A large amount of training data and computational time for simulation is required to train a robot controller with the help of DL, which is in many cases impractical. Depending on the learning method also complex fitness, as well as reward functions, are needed.

2.3. Introduction to Spiking Neural Networks

In this section, the fundamentals for SNNs are summarized, covering the concepts of neuron models, neural coding, and plasticity. For more details about SNNs see Maass [151], Vreeken [220], Grüning et al. [115], Gerstner et al. [109], and Abbott [40]. SNNs is seen as the third generation of ANNs and model closer to biology the behavior of neurons. Artificial SNN models focus on replicating the way real neurons work and attempt to replicate their biological characteristics ranging from high dimensional models to reduced approximating models. Exploring the capabilities of SNN enables research on the modeling of spike-based communication, understanding of learning mechanisms, and information representation in the brain. Figure 2.17 shows the structure of a real neuron, a spiking neuron model with event-based communication using spikes, and the membrane potential affected by the spike activity.

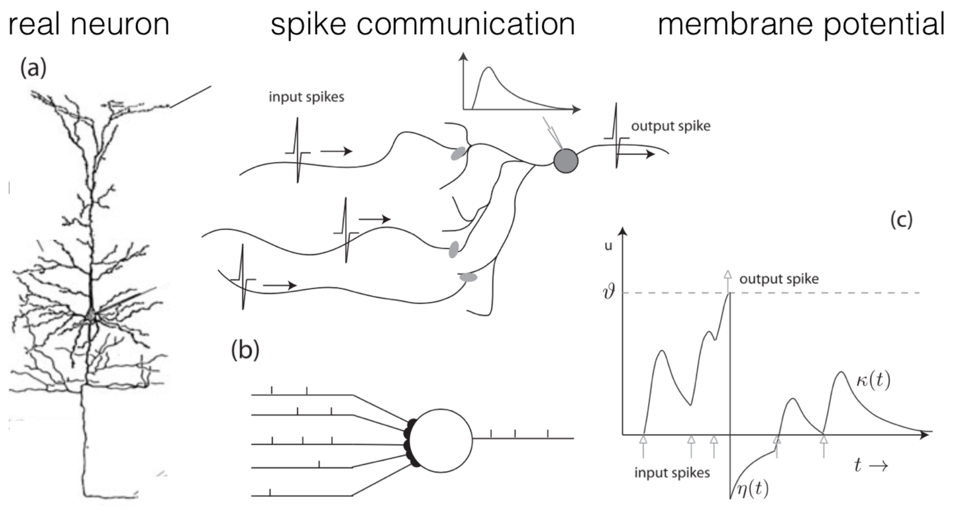


Figure 2.17.: Principles of SNNs (Grüning et al. [115]). (a) A real neuron. (b) Point neuron model, synapses, and spike-based communication. (c) Membrane potential over time.

Before introducing the concepts of SNNs, a few considerations of working with SNNs are presented. To train a SNN to perform a specific function is a big challenge. Frameworks like TensorFlow or PyTorch supporting the programming of DL models are not available for SNN yet. Nengo and NEF are probably the most mature technology to program functional SNNs. However, the recent advances on biologically plausible Error Backpropagation (backpropagation) alternative learning rules with spiking neurons offer a new dimension for applications using SNNs (Neftci et al. [162], Kaiser et al. [5], Bellec et al. [53], and Schiess et al. [193]). Additionally, the parametrization of SNNs is complex. A deep understanding of neuroscience is required to model functional networks, and the validation of these models is hard. Existing models from neuroscience focus on replicating spike activity and neural connectivity, but they are not functional for complex systems like robots. These models are related to specific problems and brain mechanisms. However, they are not generalized, and the functionality does not scale up. Finally, SNNs simulations can also be executed on neuromorphic hardware like BrainScales or SpiNNaker.

2.3.1. Neuron models

Classical spiking neuron models are defined with differential equations with respect to time. The models describe how the membrane potential of the neuron behaves.

Figure 2.18 shows the membrane potential changing over time. The resting potential is where the neuron stabilizes given enough time and if there are no input spikes. When spikes arrive, the membrane potential increases by a certain amount, and then it "leaks" energy and falls back to the resting potential. If there is enough incoming spike activity, the membrane potential increases and crosses the firing threshold, in which case the neuron produces a spike. After spiking, the neuron remains quiet during a refractory period.

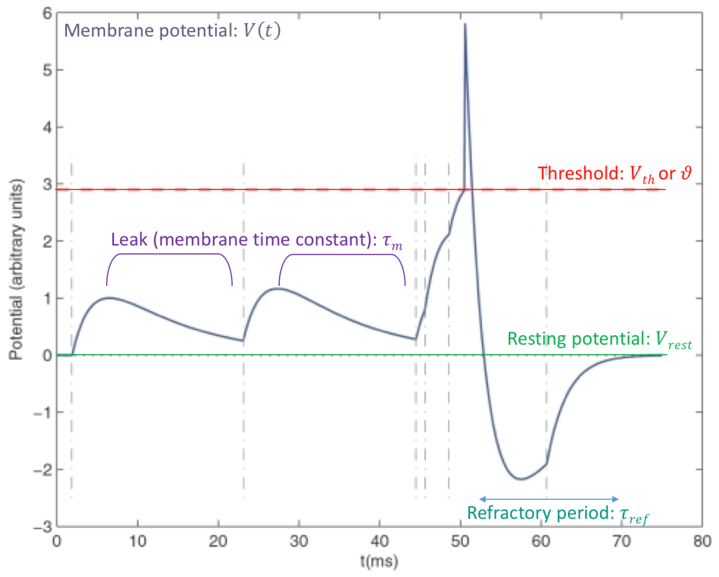


Figure 2.18.: The membrane potential of a Leaky-Integrate-and-Fire (LIF) spiking neuron changing over time. The characteristic parameters are the membrane time constant, refractory period, resting potential, and firing threshold. Adapted from Masquelier et al. [156].

Spiking neuron models range from simple functions up to complex high-dimensional and biologically realistic models. The four most popular neuron models are LIF, Hodgkin-Huxley, Izhikevich, and Spike Response. The LIF model can be represented as a simple electric circuit and it focuses on precise spike-time (Abbott [40]). The Hodgkin-Huxley model is a biologically realistic model that incorporates complex physiologic characteristics, which requires extensive computation time (Hodgkin et al. [125]). The Izhikevich model has a good compromise between biological plausibility and computation time, and it can produce several spike patterns (Izhikevich [130]). The Spike Response model is a powerful yet simple and general model that can reproduce neuron activity with a purely mathematical basis (Gerstner [108]).

The LIF is the most popular model for neuro-engineering because it provides great performance and can be implemented in software and hardware. This work uses the LIF or variants of LIF neuron model for all approaches. As a simple electronic circuit can represent it (see Figure 2.19a), it can be implemented very easily on the available neuromorphic hardware. The standard form of the LIF model is described in Gerstner

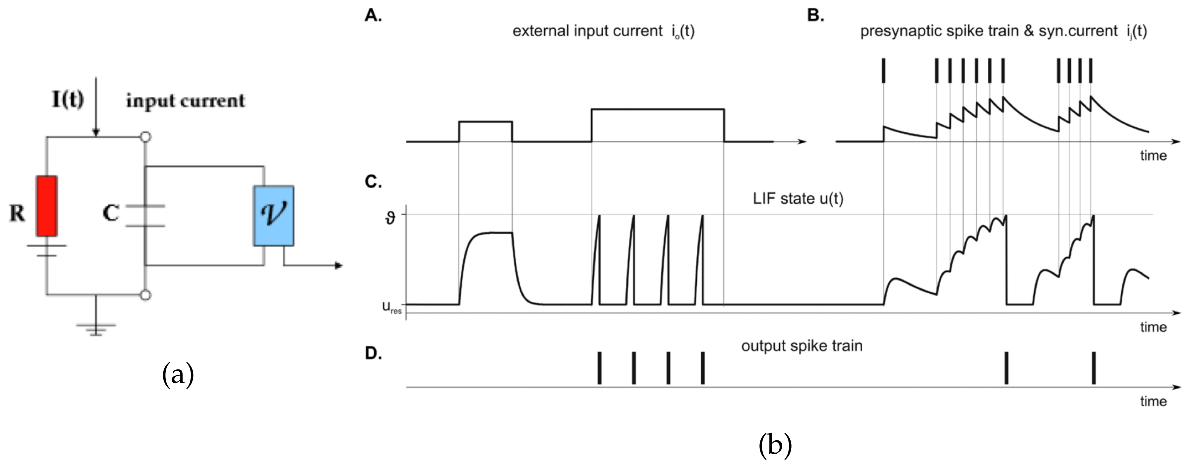


Figure 2.19.: LIF spiking neuron model. (a) Circuit representation (Paugam-Moisy et al. [167]). (b) Membrane potential driven by continuous current input (*top*) and by spike activity (*bottom*) (Ponulak et al. [173]).

et al. [109], and it is defined as

$$\tau_m \frac{d_u}{d_t} = -[u(t) - u_{rest}] + RI(t). \tag{2.2}$$

Where R is the cell membrane resistance, I the driving current, u is the membrane potential, and τ_m is the membrane time constant of the neuron. The LIF model is fast and straightforward to compute. However, it does not model the shape of action potentials. For more details about the LIF model see (Gerstner et al. [109] and Abbott [40]). The behaviour of the LIF model can be seen in Figure 2.19b.

2.3.2. Neural coding

With classical ANNs, encoding and decoding of information are solved problems compared to SNNs. Usually, an input vector is mapped to another vector, whereas for SNNs a schema needs to be defined to make sense of spike-trains. The brain uses spikes to encode stimuli, process information, and activate muscles to produce behavior. Neural coding can be done with individual neurons or with groups neurons. The most used coding schemas are rate, binary, and time (see Figure 2.20). A tool for analyzing Spatio-temporal patterns in spike trains is elephant (Yegenoglu et al. [227]). The details on the coding schemas used in this thesis are presented in Section 3.5.

Rate coding works by processing the firing rate of a neuron to represent values by fixing the input current (see Figure 2.20a). The spike rate is computed over discrete intervals. Computing the spike rates is slow and inefficient. This schema is applied to convert traditional networks to spiking. Because of this, SNNs can be seen as a

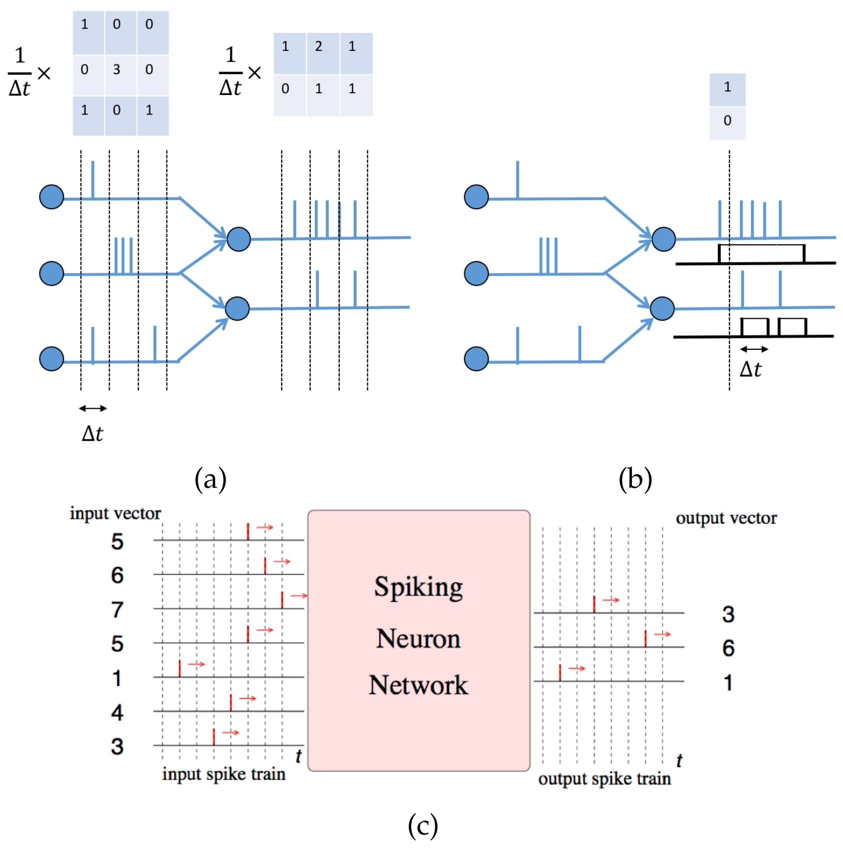


Figure 2.20.: Classical spike coding schemes. (a) Rate coding. (b) Binary coding. (c) Precise time coding (Paugam-Moisy et al. [167]).

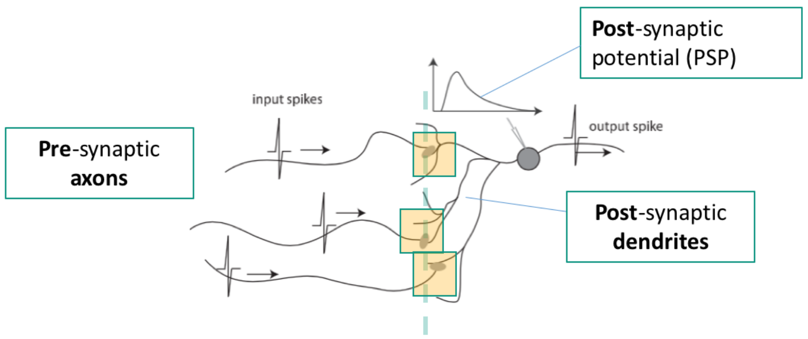
super-set to traditional networks. It is mainly used for image processing and cognitive operations. An interesting variant of rate coding is to use Poisson generators to represent the inputs, using the mean as the desired firing rate.

Binary coding works by sampling the spike trains and observing which neurons are active (see Figure 2.20b). When a neuron fires, a filter is applied to set it active for a period of time. A similar principle for values $\in \mathbb{R}$, using an exponential filter instead of binary (simulates PSPs). It is mainly used for stochastic inference.

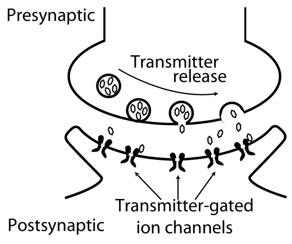
Spike time coding is a synchronous coding scheme (see Figure 2.20c). The problematic aspect is that a reference time has to be defined, and the information is coded with respect to it. There are different variations based on how time is measured — time to first spike, temporal coding, rank order coding, or correlation coding. This schema supports complex computations with few neurons, and communication is very efficient. There are analog computing principles that operate directly on the signal level implementing the circuits. They operate in parallel and are extremely fast, but they are difficult to program.

2.3.3. Plasticity and learning

What happens to neurons and synapses during learning? The most important elements are illustrated in Figure 2.21a. The strength of post-synaptic potential (PSP) depends on the number of neurotransmitters in the axon and the number of ion channels (receptors) in the dendrites (see Figure 2.21b). In simulators, this is represented by the synaptic strength — the weight of the connection. Plasticity is the change in one of these quantities. The details on the learning mechanisms used in this thesis are presented in Section 3.6.



(a)



(b)

Figure 2.21.: Learning and synapses. (a) Synapse elements (adapted from Grüning et al. [115]). (b) Schema of synaptic transmission (Gerstner et al. [109]).

Everything people know, everything people can do, everything people remember, is encoded in the synaptic strength —- according to neuroscience. Synaptic plasticity enables learning. With plasticity, learning is local and incremental; it is a form of Hebbian learning that can be understood as *"neurons that fire together are wired together"*. STDP depends on the precise timing of the spikes coming in and going out of a neuron. Different STDP curves are presented in Figure 2.22.

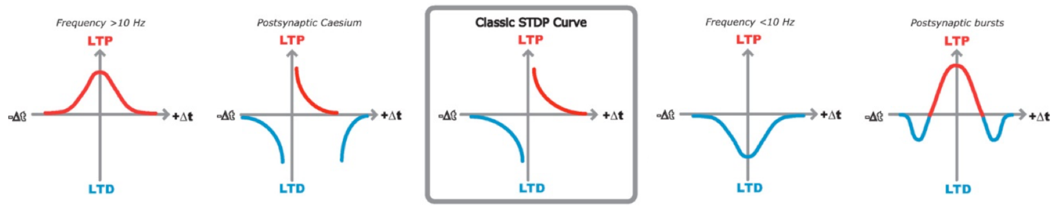


Figure 2.22.: Different STDP curves (adapted from Markram et al. [154]).

The question if backpropagation is biologically plausible or not is still open, and various hypotheses are under discussion. There are many papers, which train a traditional ANN with backpropagation and then convert it to a SNN. Nevertheless, there are still significant problems with the biological plausibility of backpropagation. The computation would need to be precisely clocked to alternate between feed-forward and backward phases. The information (error) does not travel backward in biological synapses. If the error is propagated in recurrent connections, these connections need to know about the feed-forward weights. Additionally, the gradient of spike activity is discontinuous.

However, there have been several promising papers on backpropagation-like learning rules to train SNNs. For example the works in Neftci et al. [162] using surrogate gradient, in Kaiser et al. [5] using the principles from feedback alignment, in Bellec et al. [53] and Subramoney et al. [209] using eligibility traces, in Schiess et al. [193] and Brea et al. [59] learning with multi-compartment neurons using dendritic spikes, or in Pozzi et al. [174] learning with reward-based using BrainProp.

2.4. Robot motor control using SNNs

The field of neurorobotics focuses on developing and analyzing robots using control principles of biological nervous systems. It follows the paradigm of embodied cognition that an intelligent agent needs a body to understand its environment to develop higher cognitive skills. Various computational models have been proposed to understand the sensorimotor system, which mimics the behavior of brain regions or replicate simple brain functionalities. The basis of those models is data from in vivo experiments and results from experiments with animals.

2.4.1. Coordinate transformation using STDP and learning of non-linear functions

Davison et al. [87] show that spatiotemporal data can be learned frame by frame with a two-layer network with plastic connections. The network is trained via association to perform a coordinate transformation from a reference frame to another. With this, a potential use of STDP was shown, together with an analysis about how different learning rules apply best to different training data.

Another example for a SNN learning spatiotemporal data is proposed in Srinivasa et al. [203]. They extend the idea of associative learning with a complex network. The model aims to associate joint configurations with spatial changes, which imply a feed-forward kinematic. The model is shown in Figure 2.23. The network consists of five layers connected in a feed-forward way using other emergent properties to improve learning, like lateral inhibition.

The first layer encodes the input data, and each encoded activation signal is sparsely projected on a second input layer. The second layer is connected to a Winner-Takes-All (WTA) circuit and a third layer. Each neuron in the second layer is laterally connected with its neighbors. This results in a fault-tolerance of the network towards missing input signals. The third layer is two-dimensional, and the neurons of the second layer are connected one-to-one so that the angular change θ_0 is connected to the rows, and angular change θ_1 is connected to the columns. The third layer is sparsely connected to the fourth layer, which represents the output layer. The output layer is connected with the teaching layer and the third layer.

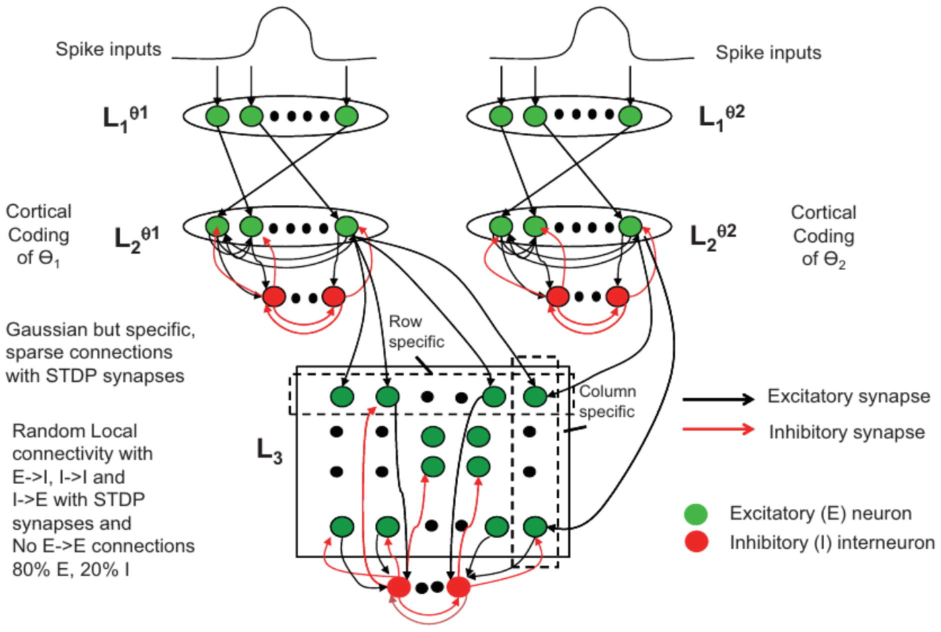

Spike inputs
Spike inputs
$L_1^{\theta 1}$
$L_1^{\theta 2}$
Cortical Coding of Θ_1
$L_2^{\theta 1}$
$L_2^{\theta 2}$
Cortical Coding of Θ_2
Gaussian but specific, sparse connections with STDP synapses
Row specific
Column specific
Random Local connectivity with E->I, I->I and I->E with STDP synapses and No E->E connections 80% E, 20% I
L_3
Excitatory synapse
Inhibitory synapse
Excitatory (E) neuron
Inhibitory (I) interneuron

Figure 2.23.: Associative learning model by Srinivasa et al. [203].

2.4.2. Learning arm configurations by associative learning and STDP

Bullock et al. [63] present the Direction-to-Rotation Effector Control Transform model DIRECT which trains a neural network to associate data of arm positions and joint configurations. The data is generated via motor babbling, which can be observed in the behavior of babies. The result is a network that learns to control the robot arm without knowing the underlying arm model. Such approaches are also called model-free learning.

Bouganis et al. [57] proposed a model inspired by DIRECT that autonomously learns to control a robotic arm through motor babbling. They use a feed-forward network with spike-time-dependent plasticity to learn an approximation of the inverse kinematic of a robot arm by association. The network is implemented with Izhicevich spiking neurons. The network architecture consists of seven input layers, four encode the actual joint configuration, and three encode a movement of the Tool Center Point (TCP) of the arm. The layers are connected all to all by inhibitory and excitatory connections, which provides a balanced weight development. The input layers are connected to four output layers, which encode angular changes in the joints. All connections are plastic and use a symmetric STDP learning rule. A gaussian population coding encodes input and teaching signals. There are two different phases for training and testing (see Figure 2.24). The training phase consists of random movements or motor babbling of the arm from a given start position. The changes in direction and the angular joint configuration, together with the joint configuration before the movement, build a single data sample. After 300 iterations, the network connections learn one sample. In the test phase, the network was presented with the actual joint configuration and a direction of the tool, which resulted in a movement of the arm.

2.4.3. Modeling the neural behavior of grasping

For grasping, Fagg et al. [100] propose a detailed computational model for visual grasping of primates with the FARS model. It describes the Ventral Pre-Motor Cortex(F5) models and the Anterior Intraparietal Sulcus (AIP) and simulates their interaction. The model uses a simple neuron model and combines several of them to a so-called p-unit that mimics a biological neuron's behavior. The main achievement of the model is the qualitative reproduction of cell behavior in the F5 and the AIP area. Oztop et al. [164] propose an extension to the FARS model to model mirror neurons. Motor neurons play an essential part in learning movements. They are active when actions are executed and fire if a primate observes another primate performing the same task.

Another model by Oztop et al. [165] describes the learning of reaching and grasping of infants. The model is called *Infant Grasp Learning* and it is illustrated in Figure 2.25. First, the affordance of an object is classified by an affordance input layer. Three kinds of layers receive the signals produced. They incorporate the concept of the virtual finger, a concept explained in the book of Chinellato et al. [68]. The concept of the virtual finger describes a physical finger acting together to apply an opposing force to an object or other fingers. This enables a reduction of the required DoFs for grasping. The virtual finger layers coordinate single virtual fingers The hand position layer calculates the movement of the arm. The wrist rotation layer computes the orientation of the hand. Suppose all layers have sent an output signal. In that case, the results are transmitted for further calculation in a simplified model of the motor cortex and spinal cord, resulting in the movement of the hand and fingers. The resulting grasp is evaluated, and a reward signal sends the updated connections of the model, leading to either synaptic potentiation or depression. The reward signal is interpreted as the joy that infants experience while interacting with the world.

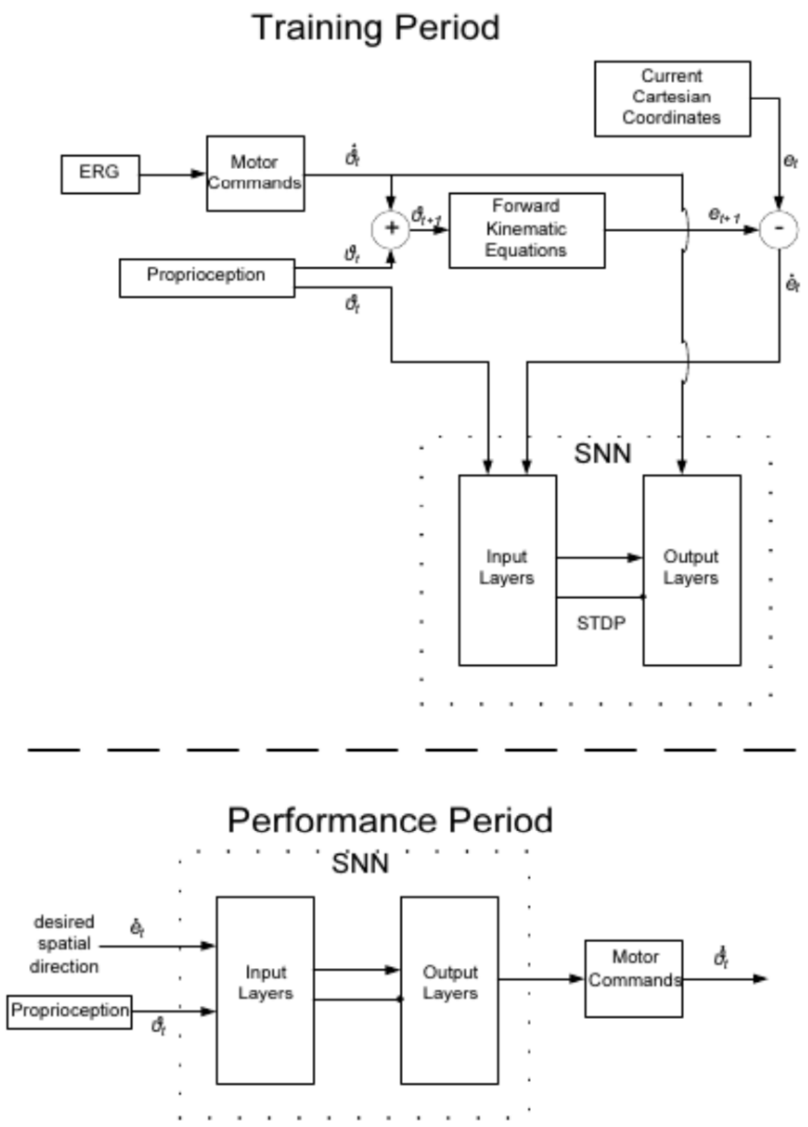


Figure 2.24.: Schematic of training and evaluation phases (Bouganis et al. [57]). For the training phase, a random movement generator was used to control the arm. Movements have been recorded and trained by the network. In the evaluation phase, only the current angle configuration and the movement intention is presented to the network.

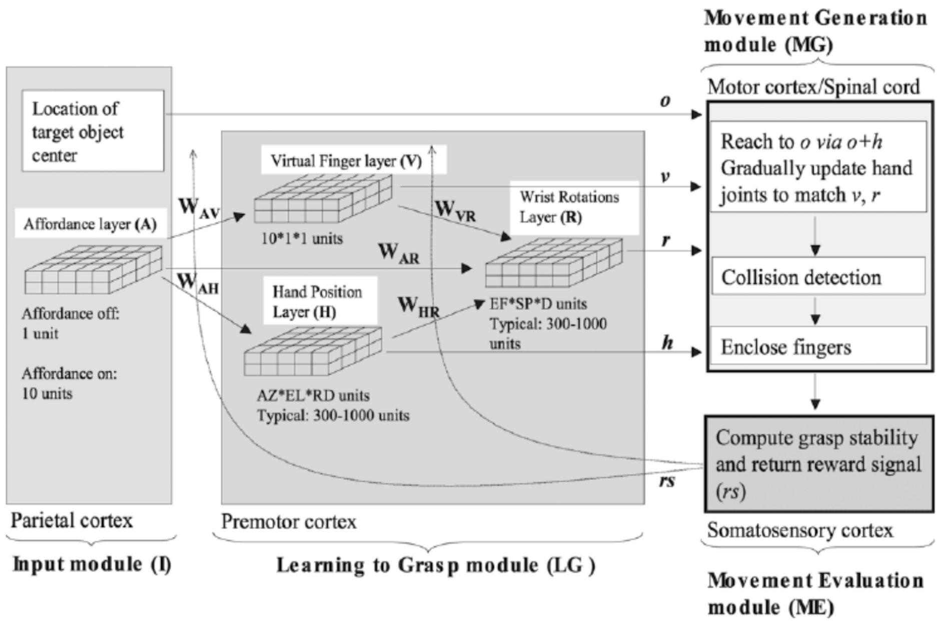

Movement Generation module (MG)
Motor cortex/Spinal cord
Location of target object center
Affordance layer (A)
Affordance off: 1 unit
Affordance on: 10 units
Virtual Finger layer (V)
10*1*1 units
Wrist Rotations Layer (R)
Hand Position Layer (H)
EF*SP*D units Typical: 300-1000 units
AZ*EL*RD units Typical: 300-1000 units
W_AV
W_VR
W_AR
W_AH
W_HR
o
v
r
h
rs
Reach to o via o+h Gradually update hand joints to match v, r
Collision detection
Enclose fingers
Compute grasp stability and return reward signal (rs)
Parietal cortex
Premotor cortex
Somatosensory cortex
Input module (I)
Learning to Grasp module (LG)
Movement Evaluation module (ME)

Figure 2.25.: Infant Grasp Learning model (Oztop et al. [165]). The model learns to grasp using reinforcement learning. The model is divided into an affordance layer, a layer for the virtual fingers, a hand position layer, a wrist rotation layer, and motor cortex and spinal cord model. With the division of the model, they could show that finger movement and hand movement can be modeled and trained independently.

2.4.4. Cerebellum models for Arm motor control

An overview is presented about approaches using functional models of the cerebellum applied to robotics in the following. Basic principles modeling cerebellar mechanisms for robots were first presented by Albus [44]. After that, several other models have been developed. The most relevant for this thesis are those implemented with SNN presented by D'Angelo (D'Angelo et al. [80] and Antonietti et al. [46]), Ros (Luque et al. [149] and Luque et al. [150]) and Tolu (Tolu et al. [215, 214]). The key characteristics of these models are the adaptation of the motion with regard to changes in the dynamics of the robot and the online learning capabilities. The cerebellum integrates the sensory input to fine-tune the motor activity, approximating its functionality and characteristics similar to the robust PID controller.

The work from Albus introduced the CMAC model to design a memory-driven control system capable of locally generalizing by associative mapping and weighing all of its inputs (Albus [44]). This means that similar inputs are mapped close to each other or even overlap in the conceptual memory since similar input values should produce similar outputs and distant input values produce independent outputs. They are then superpositioned by randomly connecting them over weights stored in the actual memory (see Figure 2.26a). CMACs can handle systems with a high number of inputs, that must not be linear nor free of hysteresis, but they need to be repeatable. It can learn a wide variety of functions with no explicit mathematical equations for the control. The local generalization offers less learning interference in comparison to the globally generalized multilayer neural networks. The model continuously improves by training incrementally and allows the acquisition and recall of new models. A life-long learning strategy is proposed for robots on this basis.

The work from D'Angelo et al. proposed various mechanisms for different functions and provided methods to consolidate the different models of the cerebellum (D'Angelo et al. [79] and Antonietti et al. [46]). Especially, this thesis explains how embodied neurorobotic models using spiking cerebellar networks can be used to explain the role and interplay of different learning rules and computation mechanisms in the brain. They propose a model implemented with a closed-loop robotic cerebellar controller from the point of neuroscience, which is illustrated (see Figure 2.26b). This control model shows how different brain regions responsible for motor control can be modeled and integrated. Three bidirectional long-term plasticity rules are implemented for different connections to approximate and compare learning behaviours similar to experimentally measured with humans. The controller was evaluated with a small humanoid NAO robot controlling three joints to execute different trajectories multiple times while the model was learning.

The work from Ros et al. proposed a feed-forward control loop with SNNs using synaptic plasticity (long-term potentiation and long-term depression) to adapt and cope with changes in dynamics and kinematics of a simulated robot (Luque et al. [149] and Luque et al. [150]). The desired trajectory in cartesian coordinates is generated to the target and transformed by the inverse kinematic module into joint coordinates (see Figure 2.27a). At each time step the target joint states ($Q_d, \dot{Q}_d, \ddot{Q}_d$) are processed to compute a crude torque command $\tau_{desired}$) and to update the predictive corrective command $\tau_{desired}$). A teaching signal is then computed by the IO neurons from the feedback error ϵ value, which is later forwarded to the PC over the CF. The controller was evaluated with a simulated light-weight robot (LWR) arm using three active DoF executing three-joint periodic trajectories with different dynamics and kinematics, providing corrective actions for more accurate movements.

Tolu et al. show two main differences regarding the three presented approaches (Tolu et al. [215, 214]). First, the main control structure includes feedback and feed-forward control loops. It can predict the next desired state or error correction and provides the possibility to implement additional motor controllers (see Figure 2.27b). Second, the functional cerebellum model has a hybrid architecture consisting of two modules: a locally weighted projection regression module (LWPR) and a cerebellum module (C). The LWPR represents the input module, substituting the following cell's connections: MF-GrC and MF-DNC. The LWPR module without any prior knowledge can cope with highly redundant and irrelevant data input by generalizing it locally. It increases its knowledge of the inverse internal model, allowing faster adaptation and prediction of the model's non-linearities. The LWPR shares a lot of functional similarities

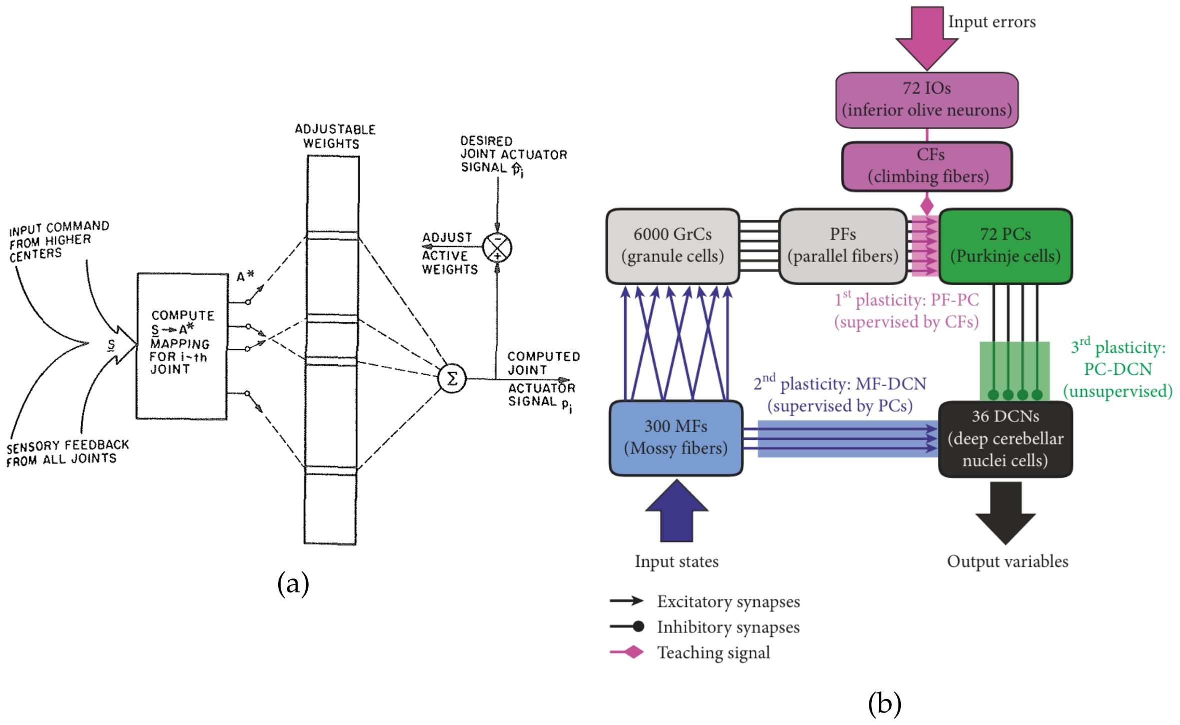

ADJUSTABLE WEIGHTS
DESIRED JOINT ACTUATOR SIGNAL $\hat{p}_i$
INPUT COMMAND FROM HIGHER CENTERS
ADJUST ACTIVE WEIGHTS
A*
COMPUTE S → A* MAPPING FOR i-th JOINT
S
Σ
COMPUTED JOINT ACTUATOR SIGNAL p_i
SENSORY FEEDBACK FROM ALL JOINTS
(a)
Input errors
72 IOs (inferior olive neurons)
CFs (climbing fibers)
6000 GrCs (granule cells)
PFs (parallel fibers)
72 PCs (Purkinje cells)
1st plasticity: PF-PC (supervised by CFs)
3rd plasticity: PC-DCN (unsupervised)
2nd plasticity: MF-DCN (supervised by PCs)
300 MFs (Mossy fibers)
36 DCNs (deep cerebellar nuclei cells)
Input states
Output variables
Excitatory synapses
Inhibitory synapses
Teaching signal
(b)

Figure 2.26.: (a) Model from Albus. A block diagram of the CMAC system for a single joint (Albus [44]). (b) Models from D'Angelo et al. The cerebellar model implemented with SNNs embedded into the controller of the NAO robot (Antonietti et al. [46]).

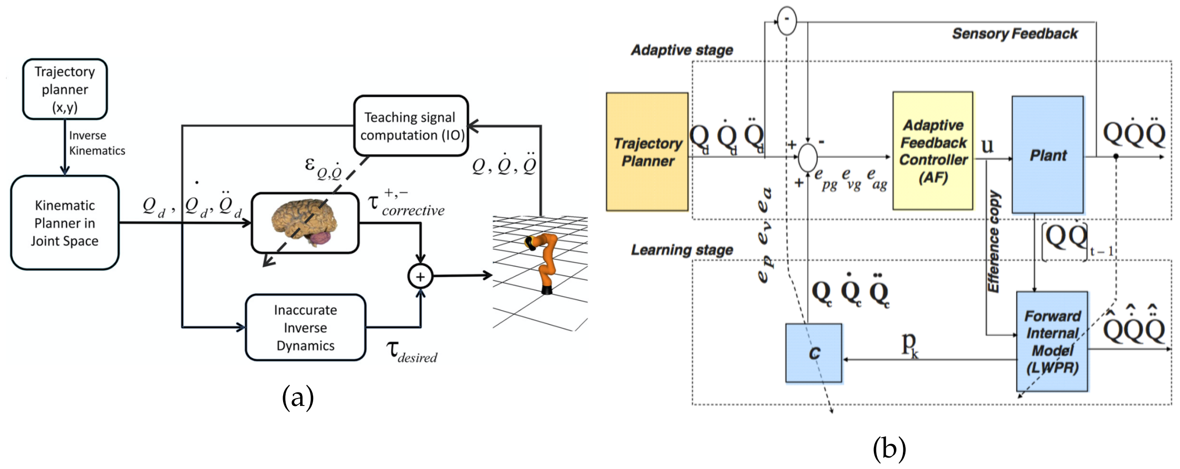

Trajectory planner (x,y)
Inverse Kinematics
Kinematic Planner in Joint Space
$Q_d, \dot{Q}_d, \ddot{Q}_d$
Teaching signal computation (IO)
$\varepsilon_{Q,\dot{Q}}$
$Q, \dot{Q}, \ddot{Q}$
$\tau^{+,-}_{corrective}$
Inaccurate Inverse Dynamics
$\tau_{desired}$
(a)
Adaptive stage
Sensory Feedback
Trajectory Planner
$Q_d \dot{Q}_d \ddot{Q}_d$
e_{pg} e_{vg} e_{ag}
Adaptive Feedback Controller (AF)
u
Plant
$Q\dot{Q}\ddot{Q}$
$[Q\dot{Q}]_{t-1}$
Learning stage
e_p e_v e_a
Efference copy
$Q_c \dot{Q}_c \ddot{Q}_c$
C
p_k
Forward Internal Model (LWPR)
$\hat{Q}\hat{\dot{Q}}\hat{\ddot{Q}}$
(b)

Figure 2.27.: (a) Model from Ros et al. The adaptive cerebellar module outputs corrective torque values ($\tau_{corrective}$) to compensate for deviations in the crude inverse dynamic module when manipulating an object of significant weight (Luque et al. [150]). (b) Model from Tolu et al. Block diagram model for the recurrent adaptive feedback error learning (RAFEL) architecture (Tolu et al. [214]).

with the CMAC model. This functional cerebellum model with hybrid architecture was implemented in two ways: in AFEL (Tolu et al. [215]) using the error between the current and target states, but this is difficult to determine; in RAFEL (Tolu et al. [214]) avoiding the distal error problem by using only sensory error data.

2.4.5. Multimodal activation, reflexes, and multi-layered multi-pattern CPG

Two studies provide the theoretical foundation for this approach. The first one, concerning the architecture of CPGs, states that pattern formation and rhythm generation are generated on different levels (see Figure 2.28). This is derived from discovering a neural circuitry, embodying a two-layered CPG that implements these processes disjunct. A biological motivation for this structure is that one-layered CPG lacks robustness since motor neurons and interneurons are connected directly. The second study is about a neural model with the ability to generate various patterns, even oscillations. This means that rhythmic and non-rhythmic patterns originate within one neural model. In this approach, the CPG of each joint is divided into three layers, rhythm generator, pattern-formation, and a motor neuron layer. The rhythm generator produces various sorts of patterns, like oscillation, almost oscillation, or plateau, depending on the tonic drive received from a locomotion control center. Proprioception, exteroception, and the output of the rhythm generator are the input for pattern formation. Here the generated patterns are shaped, and a balance between extension and flexion dominance is provided. An extension has been implemented for upper body motions, like reaching and writing (Debnath et al. [89]). They proposed a unique pattern generator controlling the upper as well as lower limbs of robots. According to regulations from Descending Control, the component Rhythm Generation produces different patterns that are subsequently shaped by the element Pattern Formation. The resulting output is forwarded to motor neurons, causing flexion or extension of muscles. A circuit is formed by Ascending Afferent, which provides higher control levels with feedback, influencing Descending Control.

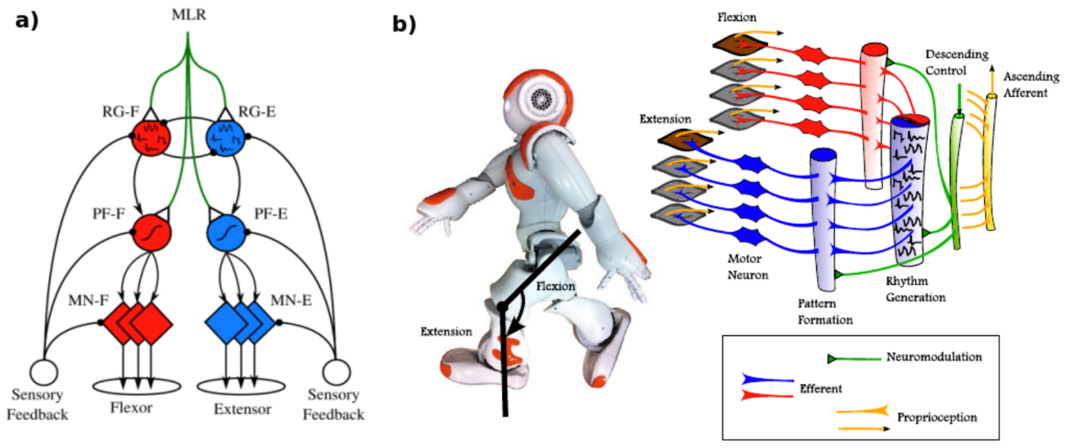


Figure 2.28.: Overview of a multi-layered multi-pattern CPG controlling a single joint (Nassour et al. [161]). The model embodies three layers, a rhythm generator, a pattern formation, and a motor neuron layer. Concept sketch illustrating how the integration of feedback and descending motor control influence the motion generation of the rhythm generator and pattern-formation layers.

2.5. Neurorobotics

Neurorobotics studies the application and validation of brain-inspired computational principles and neural control models for robots. Simulated functional brain models are connected to robotic embodiments to model closed-loop control systems. The ability of the robots to interact with the environment – either in the real world or in simulation – allows the study of brain models in a system where a brain model is directly coupled to an embodiment in real-time and where the actions generate sensor feedback. Such closed-loop experiments enable a bidirectional comparison and exchange of knowledge between robotics and neuroscience to understand the underlying principles better. In robotics, findings from neuroscience can be applied to overcome the limitations of standard control architectures and implement neuromorphic hardware for robot control tasks. In neuroscience, robots are applied as a tool for verifying hypotheses, as brain models can be fully observable during the robot's interaction with the environment. Thus, findings about the human brain can be applied to develop adaptive and learning robot controllers, and neuroscientists can test their theories about the brain and validate their models. There are global efforts to understand the human brain with initiatives such as the Allen Brain Atlas, the Human Brain Project in Europe, the Brain Research through Advancing Innovative Neurotechnologies (BRAIN) in the USA, the Brain Mapping by Integrated Neurotechnologies for Disease Studies (Brain/MINDS) in Japan, and the PRISM Brain Mapping in Australia.

2.5.1. The Human Brain Project

This work was developed under the research scope of HBP[2] as part of the neurorobotics sub-project. The HBP is part of the European Commission Future and Emerging Technologies (FET) flagship programme. It aims to accelerate the fundamental understanding of the human brain, make advances in defining and diagnosing brain disorders, develop and test new therapies for brain diseases, and develop innovative brain-inspired computing technologies. The HBP enables collaboration between scientists from different nations and research areas by creating a collaborative research infrastructure. The idea is to combine neuro-scientific data and interdisciplinary methods from neuroscience and medicine with information technologies and robotics.

Within the HBP, six platforms are developed as part of a shared digital brain research infrastructure called EBRAINS[3]. The neuroinformatics platform provides searchable atlases and analysis tools for brain data. The brain simulation platform allows building and simulating multi-level models of brain circuits and functions. The medical informatics platform allows analysis of clinical data to understand brain diseases better. The neuromorphic computing platform develops hardware implementing brain-like functions. The NRP is developed to test brain models and simulations with an embodiment in a virtual environment. High-performance platform computing provides the necessary infrastructure and computing power.

2.5.2. The Neurorobotics Platform

The *Neurorobotics Platform (NRP)*[4] (Falotico et al. [2]) is a research infrastructure for virtual neurorobotics developed as part of the HBP.The NRP is an open-source framework supporting neuroscientists and roboticists, offering a collection of different tools and technologies for defining and performing neurorobotics experiments. The

[2] `https://www.humanbrainproject.eu`
[3] `https://ebrains.eu/`
[4] `http://www.neurorobotics.net/`

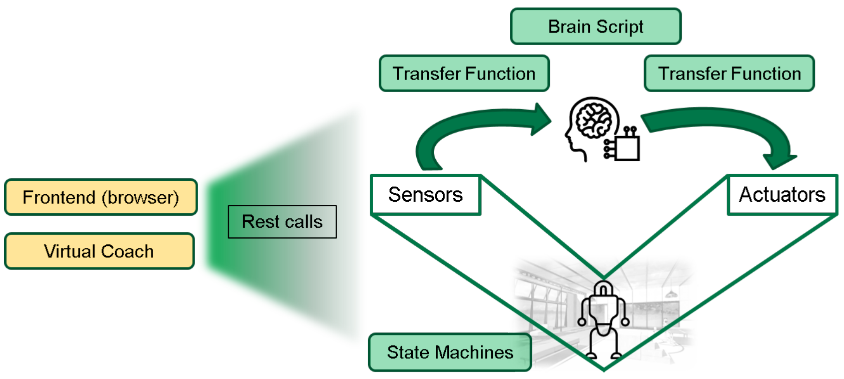


Figure 2.29.: General schema of the NRP with the main components illustrates how they interact in closed-loop (Tieck et al. [17]).

NRP integrates a neural simulator and a physics simulation by defining transfer functions to pass information between the brain model and the robot model within an experiment (see Figure 2.29). Sensor data from the robot, such as camera images or joint information, can be translated into neural stimuli for the brain simulation, and neural activity observed at a specified part of the brain can be used to control the actuators of a robot (Tieck et al. [17]). For more details about the NRP see Appendix A.

2.5.3. Neurosimulators

There are different ways of implementing and simulating SNNs depending on the neural models, the data available, and the desired functionality. In the following, the neurosimulators that were used for this thesis are presented, and other related simulators with relevant features for neurorobotics are discussed. Most of the experiments in this thesis were implemented using Nengo. Only the experiments in Sections 4.3 and 5.2 were implemented with NEST (NEST).

SNN simulators

Nengo[5] (Bekolay et al. [51]) is a tool based on the Neuro Engineering Framework (NEF) (Eliasmith et al. [99] and Stewart [207]) that is specialized for functional networks. It is implemented in Python to build and simulate large-scale neural models (e.g., SPAUN presented in Eliasmith et al. [97]) abstracts the definition of the network from the parameterization and from the backend it runs. It can use different backends such as CPU, GPU, neuromorphic hardware, or FPGA. It is fully scriptable and has an interactive GUI with live visualizations that are very useful for debugging and understanding how the network works. It is possible to define different neuron

[5] `https://www.nengo.ai/documentation/`

types, learning rules, optimization methods, reusable sub-networks, and use both spiking or traditional non-spiking models. It is also possible to process input directly from the hardware, build and run deep neural networks, drive robots, and integrate models running on other neural simulators or neuromorphic hardware. Nengo has libraries to accelerate and improve the development process, such as cognitive modeling, deep learning, adaptive control, and accurate dynamics. Nengo was used to implement most of the experiments in Chapter 4, Chapters 5 and 6.

NEST is a tool that focuses on the dynamics, size, and structure of neural systems. It is specialized for large-scale simulations on multi-core computers or clusters. It is possible to define and connect large networks using algorithmically determined connections or data-driven connectivity. The state of each neuron and each connection at any time during a simulation can be inspected or modified. NEST was first released in 1994, and it has one of the largest and most experienced developer communities of all neural simulators. NEST was used for the experiments in Sections 4.3 and 5.2.

PyNN (PyNN) offers a simulator-independent language to implement networks in a portable way for building neuronal network models. With PyNN it is possible to work with either NEST, Brian, NEURON, or specific neuromorphic hardware. The code for a model is written in Python using the PyNN API, and then it can be computed with few modifications on a different backend.

GeNN[6] (Knight et al. [139]) is a tool that focuses on performance using GPU enhanced neuronal network simulation. Models are defined in a C-style API, and the code for running them on either GPU or CPU hardware is auto-generated. There are interfaces for SpineML and SpineCreator, a Python interface (PyGeNN), and a Brian interface via Brian2GeNN.

Brian[7] (Goodman [113]) is a tool written in Python that is widely used in computational neuroscience focusing on single-compartment neurons. It is designed to be easy to learn and use, highly flexible, and extensible. It is possible to define models by writing arbitrary differential equations in ordinary mathematical notation, and it is especially valuable for working on non-standard neuron models. It is an alternative to using Matlab or C for simulations.

NEURON[8] (Hines et al. [124]) is a tool for computational neuroscience to create biologically realistic quantitative models of brain mechanisms. It is used for cross-validating data, estimating experimentally inaccessible parameters, testing hypotheses, and determining the smallest subset of anatomical and biophysical properties necessary to account for particular neural phenomena, and exploring the operation of brain mechanisms in a simplified form.

[6] `http://genn-team.github.io/genn/`
[7] `https://briansimulator.org/`
[8] `https://neuron.yale.edu/neuron/`

Deep learning tools for SNNs

Recent studies have modified and extended current state-of-the-art deep learning frameworks to simulate SNNs to take advantage of the tools and interfaces, especially for gradient calculations. The simulation of an ANN can be adapted to model spiking neurons using very short simulation times and a binary activation function. This is the case for SpyTorch (Neftci et al. [162]) using PyTorch (PyTorch) and e-prop (Bellec et al. [53]) using TensorFlow (TensorFlow). PyTorch and TensorFlow are deep learning tools that can be modified or adapted to simulate spikes.

The work from Neftci et al. [162] on surrogate gradient learning for SNNs is implemented with PyTorch. Surrogate gradient approaches can train SNNs to perform at unprecedented performance levels on a range of real-world problems. The implementation of SpyTorch is available online [9]. PyTorch was also used in combination with Nengo in Tieck et al. [18].

The work from Bellec et al. [53] on biologically plausible online network learning through gradient descent for SNNs is implemented with TensorFlow. The method is called e-prop, and it approaches the performance of Backpropagation Through Time (BPTT), which is the best-known method for training recurrent neural networks in machine learning. Parts of the implementation of e-prop are available online [10].

2.5.4. Neuromorphic hardware

Most advantages of SNNs can only be exploited with special hardware. Spike-based communication enables hardware optimizations that allow low energy consumption and fast local operations. Grapihcs Processing Units (GPUs) are appropriate for parallel computations but can not take advantage of sparse communication. Neuromorphic hardware (see Figure 2.30) is developed to take advantage of the sparse communication and efficient event-based computation of SNNs (Rhodes et al. [178]).

Research on neuromorphic hardware has been made mainly at universities. This is the case for the developments of SpiNNaker from the universities of Manchester and Dresden (Furber et al. [104] and Höppner et al. [127]), Brainscales from the university of Heidelberg (Pfeil et al. [171]), and DYNAPs from the University of Zurich (Moradi et al. [160]). Nevertheless, leading chip manufactures are entering this field, for example, Loihi from Intel (Davies et al. [85]) and Truenorth from IBM (Akopyan et al. [42]). Neurorobotics can benefit from the use of neuromorphic hardware directly on the robot, as SNNs can be scaled up to take advantage of the characteristics of event-based computation with great power efficiency (Zambrano et al. [228]).

[9] `https://github.com/fzenke/spytorch`
[10] `https://github.com/IGITUGraz/eligibility_propagation`

Neuromorphic hardware can be divided into two main categories: digital and analog. SNNs can either be simulated with digital processors or be implemented as analog circuits. Digital neuromorphic hardware uses traditional processors and memory, on which the execution of neurons is distributed. This type of hardware provides an abstraction layer between the SNN execution and the execution of instructions in the actual processor. All neurons are equal and can be parameterized as desired. SpiNNaker, Loihi, and Truenorth are examples of digital neuromorphic hardware. Analog neuromorphic hardware implement each neuron as an individual circuit, and the execution of the SNNs is made in parallel. This type of hardware can have high power efficiency and fast processing speeds, as it computes on the analog signal level. All the neurons are different, and there is intrinsic noise due to the electric signals. BrainScaleS and DYNAPs are examples of analog neuromorphic hardware.

Currently, there are significant synergies between neuroscience and neuromorphic engineering (Zenke et al. [229]). New systems and frameworks are being developed to benchmark and scale-up biologically plausible spiking neural models. There are performance comparisons between different systems, for example, between SpiNNaker and NEST (van Albada et al. [218]), and between Loihi and SpiNNaker 2 (Yan et al. [225]). A method to account for chip variations due to the manufacturing process is presented in (Büchel et al. [61]). A survey of applications and results with Loihi is presented in (Davies et al. [86]).

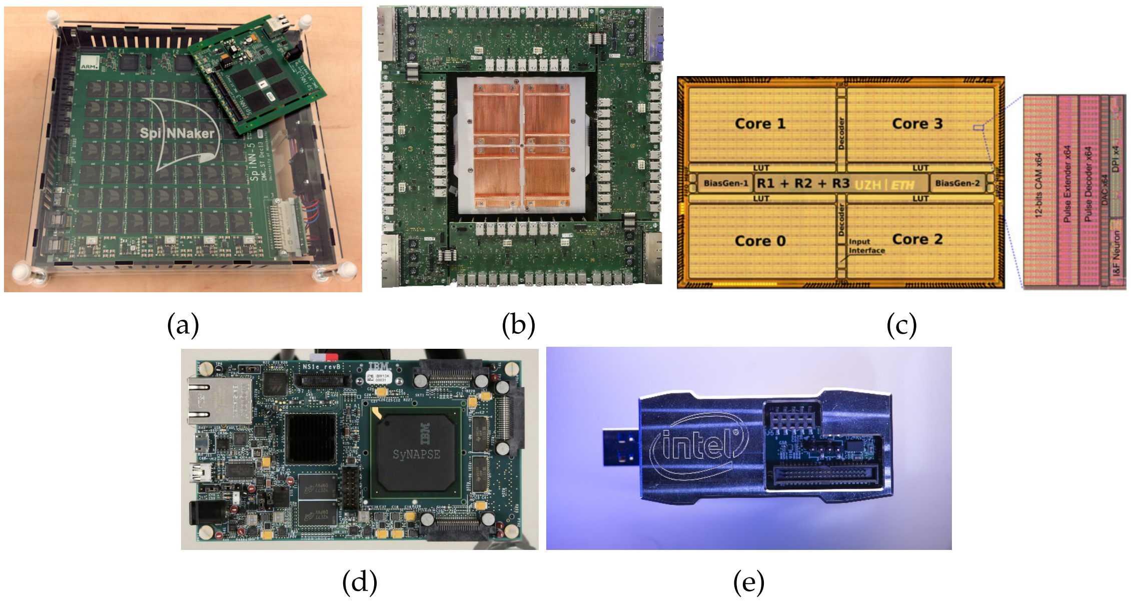


Figure 2.30.: Different neuromorphic hardware systems. (a) SpiNNaker from the university of Manchester (Furber et al. [104]). (b) BrainScales from the university of Heidelberg (Pfeil et al. [171]). (c) DYNAPs from the university of Zurich (Moradi et al. [160]) . (d) TrueNorth from IBM (Akopyan et al. [42]). (e) Loihi from Intel (Davies et al. [85]).

2.6. Summary

This chapter presented the link between biological motor control principles, computational neuroscience, and robotics on the subject of motor control.

The mechanism of how movement is represented and executed in biology is an active field of research. A widely accepted hypothesis is that the CNS uses a small number of motor building blocks that are combined to produce motion. These building blocks are formed by *muscle synergies* and are called *motor primitives*. Neuroscientists have found that motor primitives are organized hierarchically and are combined by the CNS to compose complex motions. There are studies on the evidence of muscle synergies for reaching and grasping. These insights from neuroscience have been successfully applied in robotics, for instance, with the concepts of the *dynamic movement primitives* and the *eigengrasps*. Nevertheless, robotics still relies mainly on classical methods. The existing models from neuroscience are not designed with functionality in mind and can not be used to control a robot. They require a lot of computational power and focus mainly on reproducing biological data or replicating the statistics of brain areas' neural activity and connectivity.

In classical robotics, the problem of motion control is solved by calculating the IK for the target point, then validating the configuration, and finally planning the trajectory. The complexity of the problem increases with the number of joints and DoF. Which makes computational expensive the development of closed mathematical models and extensive exploration methods. These solutions are suitable and affordable for production applications but lack adaptability for humanoid and service robots operating in an environment shaped for humans and interacting with objects designed for human ergonomics. DL has the potential to solve most of these problems and limitations. Nevertheless, despite recent success, DL also has some drawbacks which can not be overlooked and are impractical for real robotics applications. To train a system based on DL a lot of training data and simulation time is required. Depending on the learning method also complex fitness, as well as reward functions, are needed.

Neuroscience principles can be used to control complex biological inspired robots, and robots can be used to validate and understand brain mechanisms and learning processes. This opens a gap to use different paradigms to validate brain mechanisms and learning principles and develop functional models to control robots. One such paradigm is event-based computation using SNNs. SNNs are modeled closer to the biological aspects of real neurons, replicating the way real neurons work. With the use of neuromorphic hardware, SNNs can be scaled up to take advantage of the characteristics of event-based computation with great power efficiency.

3. Modeling and generating motion with motor primitives using SNNs

This chapter presents the motion representation principles based on motor primitives implemented with SNNs. The hypothesis that SNNs are a promising technology for controlling robot motion is based on the fact that nature has examples of sophisticated motor control mechanisms with great potential for adaptation and flexibility. Generating motor commands to solve a robotic task is complex, and therefore it requires an abstraction model. The approach proposed in this thesis is based on motor primitives as a parametric modeling approach that simplifies control, and it is biologically plausible as shown in Section 2.1. Such motion models can produce a variety of movements and be activated in different ways.

The concepts of muscle synergies, motor primitives, reflexes, circular activation in the motor cortex, central pattern generators, inter-neurons in the spinal cord, and motion adaptation motivate this thesis and are applied to model motions with SNNs. The details of the underlying principles from biology for motor control were presented in Section 2.1. The parametric modeling for motor primitives, its formalization, and its mapping to robot kinematics is presented in Section 3.2. This model allows a dimension reduction of the control parameters. It provides a simplification of the search space for valid configurations as the primitives provide a set of "correct" or "optimized" motions. Different activation modalities for motor primitives are presented in Section 3.3. The activation mechanisms for voluntary (intentional), rhythmic (repetitive), and reflex motion activations are described and a mechanism for contact detection. The way motion primitives are organized in a hierarchical structure and how they are combined to generate more complex motions are presented in Section 3.4. The parameterization of motor primitives is possible by changing the way the activation parameter is generated and changing the mapping to the robot kinematics. The encoding and decoding schemas are presented with the closed-loop control schema with the robot control interfaces in Section 3.5. The methods to learn new primitives and adapt the network are presented in Section 3.6. Motions can be learned using examples from demonstration, from a mathematical function, or minimizing an error signal. Motions can also be adapted with online learning.

3.1. Concept overview and methodology

A general view of the closed-loop architecture with all the components of the SNNs is presented in Figure 3.1. From right to left, the layers are arranged in increasing order of abstraction, going from *motor control* of the joints and base motor primitives, followed by *low-level* and *high-level* control representations.

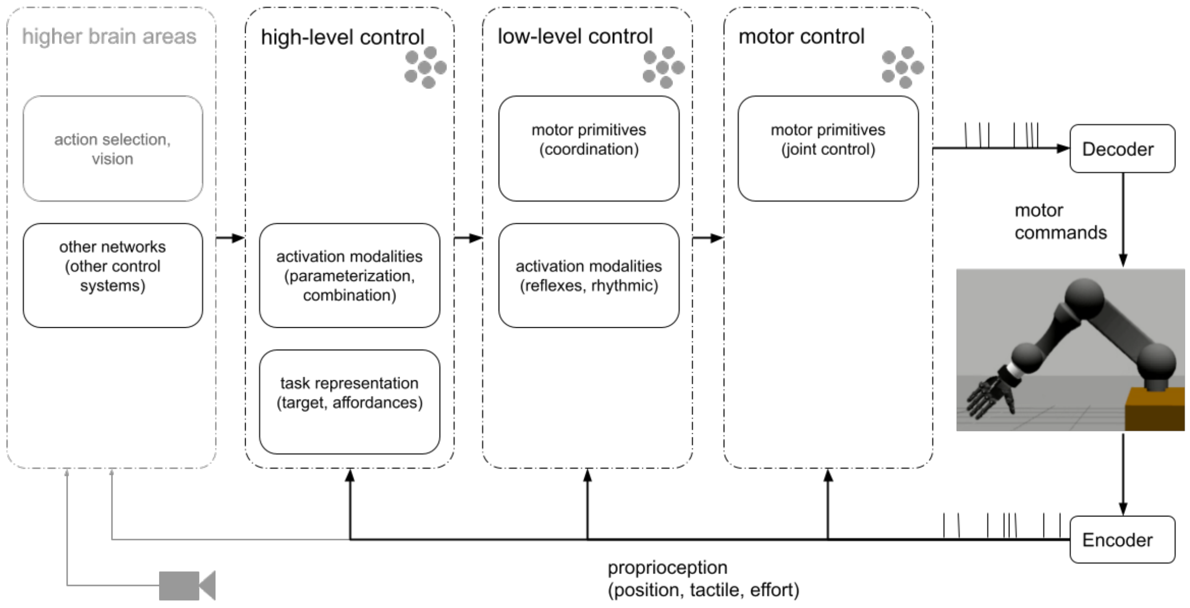

higher brain areas
action selection, vision
other networks (other control systems)
high-level control
activation modalities (parameterization, combination)
task representation (target, affordances)
low-level control
motor primitives (coordination)
activation modalities (reflexes, rhythmic)
motor control
motor primitives (joint control)
Decoder
motor commands
Encoder
proprioception (position, tactile, effort)

Figure 3.1.: General architecture for motion representation using SNNs. From right to left, the layers are arranged in increasing order of abstraction, going from *motor control* of the joints and base motor primitives, followed by *low-level* and *high-level* control representations.

In the *motor control* layer, there are motor primitives that control the joints directly. Join positions are used to control the robot. Each joint has a corresponding output population. In the *low-level control* layer, there are motor primitives and activation modalities. These motor primitives coordinate other motor primitives from the *motor control* layer, for example, for grasping or locomotion. The activation modalities in this layer provide the mechanisms for rhythmic or repetitive activations and reflexes. The reflexes provide mechanisms to activate or inhibit motions and the adaptation of motions according to sensor feedback. In the *high-level control* layer, there are activation modalities and task representation mechanisms. These activation modalities parameterize the activation modalities in the *low-level control* layer and allow the combination of motions. In the *higher brain areas* there are interfaces and mechanisms for signals coming from other networks or other control systems. It is possible to integrate non-spiking components or integrate SNNs for other brain functions such as action selection or vision. Processing the information for input and output of the

SNNs requires schemas for encoding and decoding. The proprioception provides feedback about the joint positions, tactile sensors, and the efforts of the motors.

This work is based on the principles of the NEF (Eliasmith et al. [99]) to define and generate the spiking neuron models. The software package Nengo (Bekolay et al. [51]) implements NEF and allows the creation of large-scale SNNs by breaking the networks down into smaller sub-networks. To define a SNN model using NEF, the control algorithm has to be broken down into vectors, functions, and differential equations. The activity of a population (group of neurons) is considered to be a distributed representation of a vector, for example, 100 spiking neurons representing a 2-dimensional vector. The connections between populations compute functions on those vectors. The connection weights ensure that if the first population represents x, then the second population will represent $y = f(x)$. The connection weights for each sub-network are optimized separately, and then they are combined into one large neural network. Performing this optimization (i.e., finding connection weights) locally means that large SNN systems can be generated without using the traditional neural network approach of optimizing over vast amounts of training data. However, the trade-off is that explicit claims must be made about the function of each sub-network of the model. By changing the connection weights, the function being computed is modified. The functionality of each sub-network can be defined as a mathematical function or as a set of input and target output data. Finally, recurrent connections can be added to implement differential equations. These principles are applied in this thesis to implement motor primitives.

3.2. Motor primitive formalization using SNNs

The modeling and formalization of motor primitives with SNNs is presented in this section. The concepts of motor primitives as building blocks (Bernstein [54], Flash et al. [103], and Hart et al. [118]) and the hierarchical organization of the motor system combining motor primitives (Bizzi et al. [55] and d'Avella et al. [81]) are applied.

Considerations about the modeling of coordinated motions are formulated in the following. A motion is independent of the task it can perform in terms of its execution and coordination of the actuators involved and their relation to each other (Tieck et al. [16]). For example, take the swing motion of an arm, the motion of the arm to point at something, or the flexion of individual fingers. In this sense, a motion is a sequence of joint activations, representing the synergies between the joints. A motion is defined by its start and end positions, and it is executed within a finite period. A motor primitive represents the joint activations during the execution of a motion. This can be described by a function defining a time sequence of the state of all active joints (Tieck et al. [24, 25]). Nevertheless, a model depending on time is inconvenient because this requires the system to provide a time variable to evaluate the function. Instead, a model of a motor primitive is proposed with an activation parameter $u \in [0, 1]$, an activation function $f : [0, 1] \rightarrow [0, 1]$ and a mapping to the robot kinematics $g : [0, 1] \rightarrow \mathbb{R}^n$. This modeling is parametric, normalized time independent (see Figure 3.2).

Therefore, by modeling motions with motor primitives, the problem of generating motor commands for specific motions is mapped to the problem of how to activate the primitives (Tieck et al. [24]). This method provides a dimension reduction of the control parameters because the activation parameter is controlled to generate a specific motion instead of controlling all the joints individually. The resulting motions are flexible and adaptive instead of being rigid.

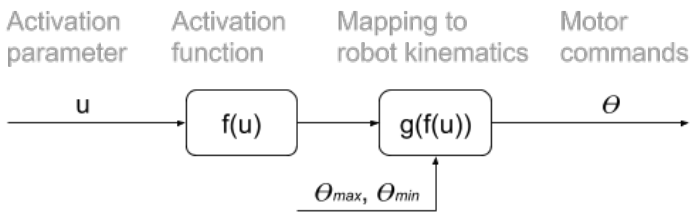


Figure 3.2.: Block diagram of the modeling of a motor primitive. The functions $f(u)$ and $g(f(u))$ are defined in Equations (3.1) and (3.2) respectively.

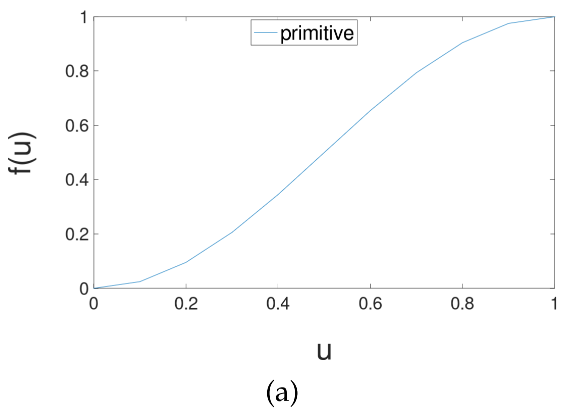
3.2.1. Modeling the activation function

A motor primitive is modeled as a mapping of an activation parameter u to a sequence of joint activations (Tieck et al. [24, 25]). The activation signal is normalized $u \in [0, 1]$, and it can be mapped to one or multiple joints. The start and end of a motion are represented with 0 and 1 respectively. This modeling provides a reduction of the control parameters of multiple joints to one control parameter (see Figure 3.3).

The activation parameter u does not represent any particular motion by itself; it is the input for the joint activation function. In theory, the activation function can be any arbitrary function as long as it is continuous. A parametric and normalized activation function $f(u)$ is proposed to represent a motor primitive with the parameter $u \in [0, 1]$, so that it is time-independent. Then, $f : [0, 1] \rightarrow [0, 1]$ can be described as

$$f(u) = \frac{sin(u \cdot \pi - \frac{\pi}{2})}{2} + \frac{1}{2}, \tag{3.1}$$

a sinusoidal function with smooth initial and final phases, as shown in Figure 3.3a. With these properties, consecutive activations of the same function generate a continuous curve with smooth transitions. Additionally, the derivative is continuous and has smooth initial and final phases. This is appropriate to control real robots in both position and speed while preventing unnecessary wear in the motors and transmissions. This type of function is commonly used in robotics for interpolation. Sinusoidal activations yield to better energy characteristics and suppress structural vibrations.



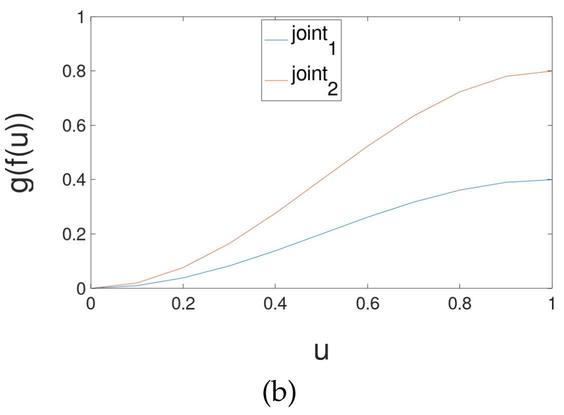


Figure 3.3.: Motor primitive modeling (Tieck et al. [25]). (a) Mapping of u to the activation function $f(u)$, Equation (3.1). (b) Complete mapping of u to the robot kinematics $g(f(u))$, Equation (3.2).

Joint name	**Primitive**	θ_{min}	θ_{max}
$joint_1$	1	0	0.4
$joint_2$	1	0	0.8

Figure 3.4.: Table for the joint mapping schema. The joints are defined with a name, the associated primitive, and the interval of the joint θ_{min} and θ_{max}. A primitive can be mapped to one or more joints.

3.2.2. Modeling the mapping to the robot kinematics

Finally, to control a robot, the primitives have to be mapped to its kinematics (Tieck et al. [24, 25]). The activation function $f(u)$ needs to be translated to motor commands to generate motions. For this purpose $g : [0, 1] \to \mathbb{R}^n$ is defined as

$$g(f(u)) = \vec{\theta}, \tag{3.2}$$

where $\vec{\theta} \in \mathbb{R}^n$ is the vector of joint values for the robot. The function g maps $f(u)$ to a set of joints by scaling it to the respective joint angle interval and using an offset to the minimum of each joint. Thus, g is robot-specific, but f is general and can be reused. In other words, this mapping scales the activation function to the motion interval $(\theta_{max} - \theta_{min})$ with an offset θ_{min} of the joint θ. For this purpose $g : [0, 1] \to \mathbb{R}^n$ is defined as a function for each joint as

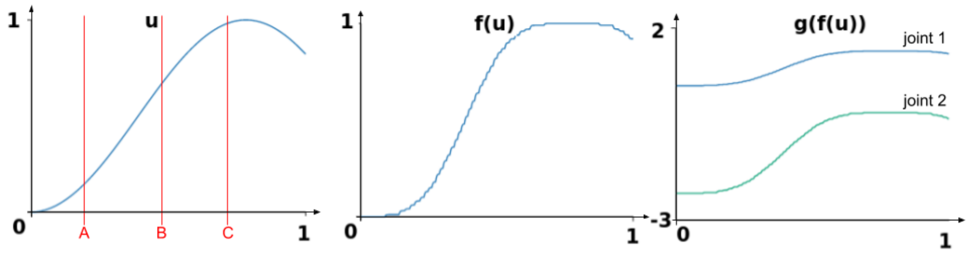
$$g(f(u)) = f(u) \cdot (\theta_{max} - \theta_{min}) + \theta_{min} \tag{3.3}$$

to generate appropriate motor commands. A schema for the mapping g is illustrated in Figure 3.3b and it is defined in the table in Figure 3.4.

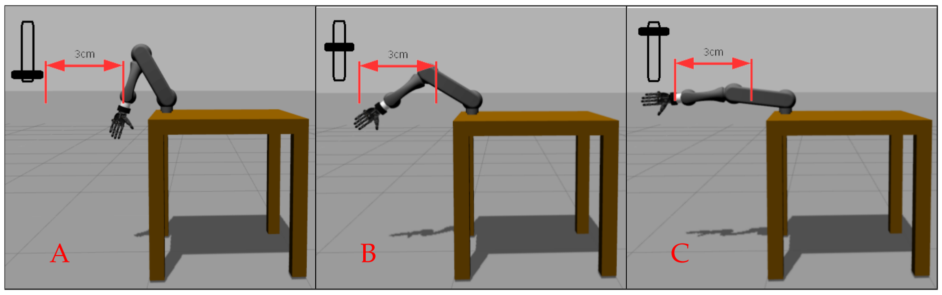
3.2.3. Generation of robot motion with motor primitives

To illustrate how all components work together, Figure 3.5 shows how a motor primitive is used to control the shoulder and elbow joints of a two-link robotic arm. Changes in the activation parameter u generate different values of the activation function, which are mapped to the joints involved in the motion (Tieck et al. [24]).

Motor primitives can be parameterized by changing the way u is generated and can be combined by using the same u to activate different primitives at the same time. There is no inverse coordinate transformation nor a trajectory planner in cartesian space. The generation of the trajectories is done in joint space using the motor primitives. Thus, the problem of joint motor control can be reduced to the definition and generation of the parameter u to activate a primitive.



(a)



(b)

Figure 3.5.: Generating motor commands using motor primitives (Tieck et al. [24]). (a) From left to right, the activation parameter u, the joint activation function *f(u)*, and the mapping *g(f(u))* for two different joints. (b) Motion frame sequence of a robotic arm controlled by a motor primitive. The frames correspond to the three red vertical lines in (a).

3.3. Activation of motor primitives and contact detection

Different mechanisms to activate motor primitives as well as a contact detection mechanism are presented in the following. A voluntary or intentional activation controls the motion constantly and can start or stop at any point of it, for example, moving the arm to point to an object. A rhythmic or repetitive activation goes from the beginning to the end of the motion and repeats it multiple times, for example, moving the legs for walking. A reflex activation is a complete one-time execution of a motion, for example, the retract reflex of the arm. The different activation modalities (voluntary, rhythmic and reflex) are used to activate, combine and parameterize motions in Section 4.2. There are different ways of modeling a motion generation layer that produces rotational neural activity similar to the motor cortex. This layer generates a relatively constant or normalized amount of spikes in a time interval. The activity of the motion generation layer is used to drive either one or multiple complete activations of a motor primitive. The first mechanism is to model a function that produces this type of activity and then having a network learn the function, e.g., a neural oscillator. Another mechanism to achieve this is by gradually tuning the network by locally adjusting the synaptic weights to match the desired activity (see Section 4.3.1). Additionally, a mechanism based on reflex circuits of interneurons in the spinal cord is proposed. This mechanism considers the joint efforts and proprioception to detect contact and trigger a reflex.

3.3.1. Voluntary motion activation

A voluntary or intentional motion activation controls the motor primitive continuously (Tieck et al. [24]). The execution of a voluntary motion is represented as a one-dimensional input. It defines the value of u, and thus, the position in the trajectory of the given primitive. The process of generating motor commands based on motor primitives is illustrated in Figure 3.5. In Figure 3.5a, the relation between u, $f(u)$ and $g(f(u))$ is illustrated. The vertical lines in the first plot labeled as "A, B, C" correspond to the frames of Figure 3.5b showing the robot motion for different values of u. Notice that a motor primitive can be mapped to one or multiple joints.

This activation mechanism is applied in the experiments presented in Sections 4.2 and 4.4. In general, there is no principal difference between voluntary and reflex motions regarding the coordination synergies. The difference is the activation mode. For voluntary motions, it is discrete over time, and for reflexes, it is immediate.

3.3.2. Motion generation layer by modeling an oscillator for rhythmic activation

The first mechanism to implement a motion generation layer requires modeling a function that generates the required activity and have a SNN learn it (Tieck et al. [25, 24]). In this case, the function is modeled as an oscillator. This modeling allows further parameterization of the motion, because the frequency and amplitude of the oscillator can be adjusted continuously to modify the generated activity. This activation mechanism is used in the experiments discussed in Sections 4.2 and 6.2 to generate rhythmic motions.

In contrast to voluntary motions, rhythmic or repetitive motions (e.g., waving with one arm) consist of consecutive executions of the same set of primitives. Pattern generators control rhythmic motions in the spinal cord of vertebrates. They are formed by an interneuron connected to a set of nerves and motor neurons, generating oscillatory patterns. In Churchland et al. [70] evidence is provided, rhythmic muscle contractions come along with neurons displaying firing rate oscillations at a similar frequency. A hierarchy of pattern generators was proposed in Nassour et al. [161] to generate joint activations for locomotion.

The generation of u has to be continuous and repetitive until interrupted by higher control layers to achieve a rhythmic activation. It is modeled as an oscillator

$$h(\omega) = a \cdot \sin(b\omega\frac{\pi}{2}), \tag{3.4}$$

with ω a recurrent connection and a and b the parameters for the amplitude and frequency, respectively. To change the parameters of rhythmic motions, a set of populations in a higher layer of the network was added to control the parameters a and b. By changing a, the amplitude of the movement is modified, and by changing b, the frequency of the movement is modified.

The oscillator is implemented with a population of spiking neurons organized in a grid (see Figure 3.6a). By indexing the neurons in the oscillator population, the activity can be mapped to a $2D$ plane. To extract u from this circular activity, the mean activation of this population of neurons is decomposed in the X and Y components. This components represent the activation of the population it the XY plane as shown in Figure 3.6b. The X and Y components can be represented as a vector (x, y) rotating around the origin, as illustrated in Figure 3.6c. From this representation a continuous and normalized signal $u \in [-1, 1]$ is extracted as

$$u = \sin(\arctan(\frac{y}{x})), \tag{3.5}$$

where $\arctan(\frac{y}{x})$ is the angle of the vector represented by (x, y). The function sin is applied to smooth and normalize the trajectory between $[-1, 1]$. One oscillation activates a whole primitive once, as both the oscillation and the primitive are normalized. Consequently, multiple oscillations activate the motor primitive multiple times.

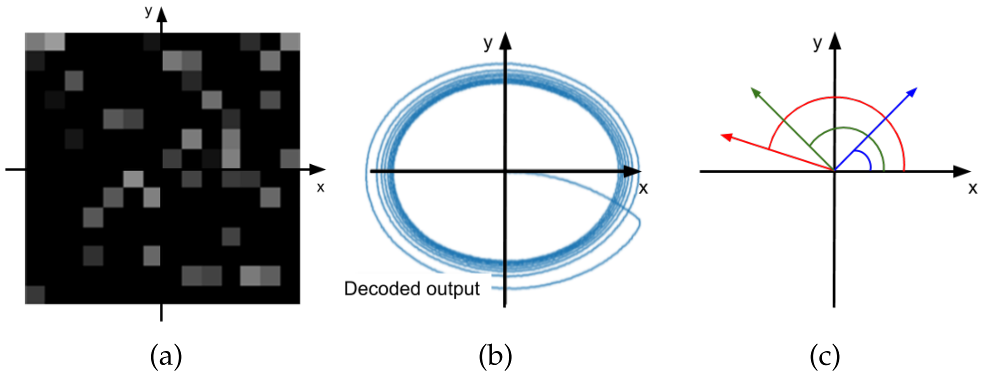


Figure 3.6.: Circular activation is modeled as an oscillator by a population of spiking neurons (Tieck et al. [24]). (a) The population is represented as a 2D grid, and the intensity represents the amount of spikes. (b) The activity is decomposed in the X and Y components. (c) These components represent a vector rotating. The angle of the vector (*different colors*) is used to generate the activation parameter u.

3.3.3. Reflex motion activation

In biology, reflexes are neural circuits that mediate specific actions depending on the sensor input and the desired motor activation. There are different types of reflexes: e.g., pupil dilatation, pain reactions, and inhibitory (Byrne et al. [64]). In this work, two different reflexes are modeled — the withdrawal reflex and the inhibitory reflex. The first one activates the retraction of the arm, e.g., when there is a collision or when touching a hot surface. The second one stops the motion in case of contact, e.g., when grasping an object. In nature, the activation of the reflexes is implemented with interneurons in the spinal cord. They receive the current motor activation and the sensory input from the body at the same time. The *Reflexxes Motion Libraries* (Kröger [143]), a reactive framework based on reflexes, is a successful example of applying reflexes for robotic motor control.

The reflex activation mechanism was introduced in Tieck et al. [25] and later refined in Tieck et al. [24]. The weights of the synapses for the activation of a reflex have to be very strong. This way, it provides either a single immediate execution of one or more primitives or the inhibition of the current motion. Both reflexes affect the motor output for the current executed motion. This activation mechanism is applied in the experiments presented in Sections 4.2, 5.3 and 5.4.

To achieve this activation, the generation of u has to be continuous and have a finite duration. The reflex activation mechanism is similar to the rhythmic motion activation (see Section 3.3.2), but with two recurrent connections. One excitatory connection prompts the reflex, starting the oscillator in the same way. An additional inhibitory connection forces the oscillator back to zero after one oscillation.

3.3.4. Contact detection intercircuit

The following mechanism to detect contact and trigger the reflex is inspired by the circuits of spinal interneurons combining inhibitory and excitatory connections. The contact detection circuit is modeled as an alternative selection mechanism of the networks with interneurons in the spinal cord (Jankowska [131]). This activation mechanism is used in the experiments presented in Sections 4.2, 5.4 and 6.2.

The basic idea is to use only the motor currents to calculate the joint efforts and use the signal to detect contact (Tieck et al. [16]). The problem with this is that both the movement of the arm and an external force cause changes in the joint efforts. This means that high efforts do not always indicate a contact. In addition to external forces, high efforts can be generated from a fast movement due to the robot's dynamics and inertia. This is inconvenient because reflexes should only be triggered when there is contact and not whenever a motion is performed. To overcome this problem, a neural circuit is proposed to provide the interneurons with proprioception to detect if the joint is moving (Tieck et al. [24, 21]).

The contact detection mechanism is illustrated in Figure 3.7. The effort in the joints is connected excitatory to the corresponding reflex interneuron. The proprioception (joint angle θ) signal is used to get the motion (joint speed Δ_θ), and the interneuron is inhibited proportionally to it. Accordingly, if the joint moves (meaning $\Delta_\theta > 0$), the population excited by the effort is also inhibited by the motion. This ensures that the interneuron can only detect contact if the effort in the joint is increasing, but the joint is not moving. Thus, solving the problem that the moving arm also generates efforts because of the dynamics and the inertia.

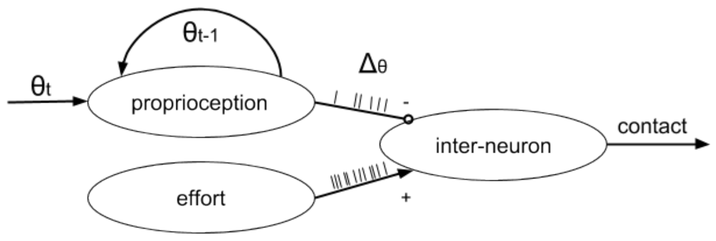


Figure 3.7.: Contact detection mechanism (Tieck et al. [21]). The *proprioception* determines the velocity of the joint Δ_θ. The current angle position θ_t is compared with the previous $\theta_{\text{t-1}}$, provided by a delayed, recurrent connection. The *interneuron*, is excited by the effort feedback from the motors and inhibited by the current Δ_θ. This ensures that the interneuron only detects contact if the effort increases and the corresponding joint is not moving.

3.4. Motor primitive combination, parameterization and hierarchy

This section presents the principles to combine multiple motor primitives, parameterize them, and build a hierarchy of motor primitives. These three principles are fundamental to reuse motor primitives to represent complex motions and behaviours.

Two biological concepts inspire the mechanisms to combine motor primitives. The first one is the evidence of motor primitives as motion building blocks (Bernstein [54]). The second one is how motion is represented, combining these fundamental building blocks to generate different movements (Bizzi et al. [55]). The parameterization of motion is based on the works from Ayaso [49] and Nassour et al. [161]. Ayaso [49] proposes an architecture to generate motor commands for arm motions and describes how learning and adaptation can be achieved by changing the gain. Nassour et al. [161] presents a hierarchy of pattern generators that are dynamically parameterized to generate joint activations for locomotion. The hierarchical architecture is based on the research from d'Avella et al. [81], which shows how the different parts of the CNS and the body form a motion hierarchy based on motor primitives.

3.4.1. Combination of motor primitives

Motor primitives are combined by activating multiple primitives at the same time. The output of the different primitives is merged in one single motor command for each actuated joint (Tieck et al. [16, 24, 20, 23]). It is clear that if the primitives activate different joints from each other, this combination is trivial. If different primitives activate the same joints, then various combination scenarios are possible, as shown in Figure 3.8. For each case, a specific combination mechanism is proposed.

In Figure 3.8a two primitives activate one common joint — *joint 2*. The mechanism averages all activations for that joint as a weighted mean. The influence of one single primitive depends on the number of active primitives. This combination mechanism is applied in the experiments presented in Sections 4.2 and 4.4.

In Figure 3.8b two primitives activate the same set of joints. Imagine two primitives controlling different motions of the same arm. For this case, the activation of both primitives is proportional, and it is considered to have different weights. The influence of one single primitive depends on a feedback signal or a signal coming from higher brain areas. This happens when there are different control loops in the network with different priorities. This happens especially in two situations. One, when there is a base motion being performed, and other motions are adjusting it. The experiments in Section 4.3 make use of this mechanism. The other one is when another motion interrupts or overtakes the control with a strong activation, as a reflex, for example. This combination mechanism is applied in the experiments presented in Sections 4.2, 5.2, 5.4 and 6.2.

In Figure 3.8c, two primitives activate the same set of joints, but their activation is exclusive. In this case, the activation of one of the primitives inhibits the other one. Thus, only one is active at a given time. This combination mechanism is applied in the experiments presented in Sections 5.4 and 6.2.

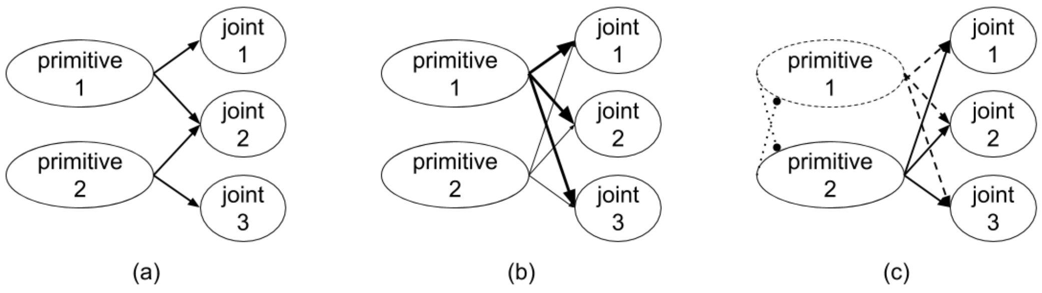


Figure 3.8.: Combination of motor primitives. Different combination cases for activation of the same joints by different motor primitives.

3.4.2. Parameterization of motor primitives

The way to parameterize a motor primitive, is by controlling the generation of the activation parameter u (see Section 3.2). There are different mechanisms to achieve this. The speed of a motion can be changed by changing the activation function $f(u)$. The range of movement of a primitive can be controlled by changing the *min* and *max* values of the mapping function $g(f(u))$. For the motion generation generation (circular activity) or for the oscillators, the amplitude and frequency can be controlled (see Section 3.3). For the combination of primitives, the weighting factor of the active primitives can be controlled. In Section 6.2, the leg primitives were controlled by changing the behavior parameters.

3.4.3. Hierarchical architecture to generate complex motions

To model complex motions based on SNN motor primitives, a control hierarchy is proposed. Motor primitives can be combined and organized in different layers, starting with simple joint movements, forming complex arm motions, and integrating signals from higher brain areas. A detailed view of the hierarchical control architecture with all layers and populations of the SNN is presented in Figure 3.9. The layers are organized in increasing order of abstraction from right to left, starting with the *motor control* of the joints and base motor primitives, followed by *low-level* and *high-level* control representations. There are many functional layers in the brain, but these four conceptual layers can represent the main motoric functions in the human motor system. These hierarchical architecture principles are used in the experiments presented in Chapters 4 to 6.

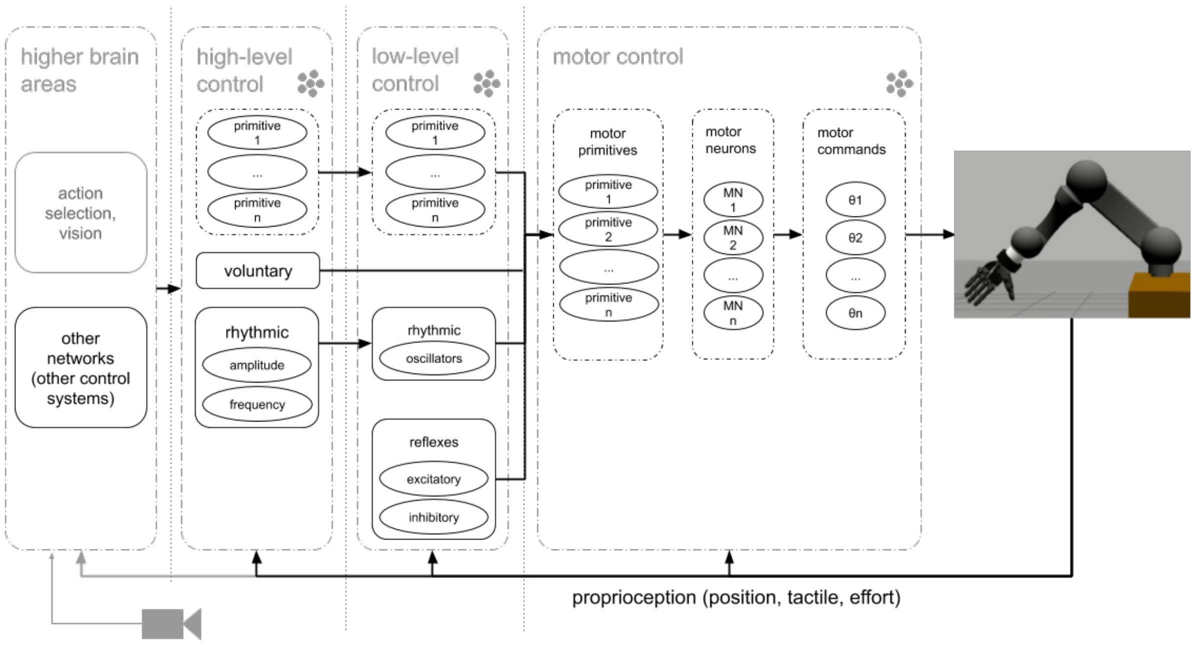


Figure 3.9.: Detailed view of the hierarchical control architecture with all layers and populations of the SNN. The layers are organized in increasing order of abstraction from right to left.

The feedback signals represent proprioception, which provides joint states, tactile information, and joint motor efforts. The feedback data is available to all the layers.

The motor control layer consists of motor primitives, motor neurons, and the mapping to motor commands. The robot receives commands from the motor control layer, where the primitives and output populations for each joint are processed. This layer is sub-divided into three sub-layers. The first sub-layer represents the joints and the mapping to the robot kinematics. The second sub-layer represents motor neurons that in biological systems activate the muscles, but with physical robots, the motor commands have to control motors. In this case, the robot requires joint position commands. The third sub-layer defines base or elementary primitives that activate multiple joints and mirrors the topology of the kinematics of an arm, finger, or leg.

For example, there is a hand coordination layer representing grasping affordances (see Chapter 5). The low-level control layer has the oscillators for rhythmic motion activation, interneurons, neural circuits for reflexes, pattern generators, and motor primitives. For the representation and generation of complex motions, motor primitives can be defined in higher layers to control other primitives instead of controlling joints directly. Additionally, other functional components can be modeled within the hierarchy. For example, to control the motor primitives as behaviours for synchronization and generate activation patterns (see Chapter 6).

The high-level control layer represents voluntary or intentional activations, the parameters for the oscillators, pattern activations, other pattern generators, and other motor primitives. The error signals are calculated using the TCP position and the desired target in the high-level control layer. The target representation can be done in the configuration space or polar coordinates.

A high-level *control interface* controls the activations patterns, providing an interface to other networks or control systems. This is necessary to integrate other SNNs and to perform experiment control. With higher-brain areas and vision, different functions coming from other networks are represented. These are included greyed out for completeness, but they are out of the scope of this thesis. SNN-based closed-loop control systems can be seen in analogy to PFM controls (Pulse Frequency Modulated controls), which has been successfully applied for stepper motor drivers and pulsed satellite control systems (Dillmann [95] and Dillmann et al. [94]).

3.5. Encoding and decoding spike activity for closed-loop robot control

Integrating SNNs into robot control requires the translation of spike trains into real-time-robot control commands and vice versa. Two aspects have to be considered, robots operate in closed-loop and in real-time. This imposes restrictions and limitations on how the network operates and how it can be trained. The encoding and decoding schemas need to be compatible with the standard robot control interfaces (Kaiser et al. [9]). The encoding schema allows the conversion from a vector of input data of sensor information into neural activity as spike trains. The decoding schema allows the conversion in the other direction, from neural activity as spike trains into robot motor commands. The closed-loop control schema integrating SNNs with the control interfaces for robots or simulation platforms is shown in Figure 3.10. For the experiments, the neurosimulators Nengo and NEST, have been connected with robot simulators and real robots with the help of Robot Operating System (ROS). ROS offers a standardized interface to most of today's robots and simulators.

The proposed encoding and decoding mechanisms are based on population representations. This means that a single value is represented by a set of neurons and not by individual neurons. Two mechanisms for encoding and decoding are presented. First, a mechanism for distributed encoding and decoding information using the principles from NEF is introduced. Second, a mechanism to encode and decode spike activity using Gaussian curves is proposed. An essential aspect for representing data with ANNs and especially SNNs is the normalization of the data.

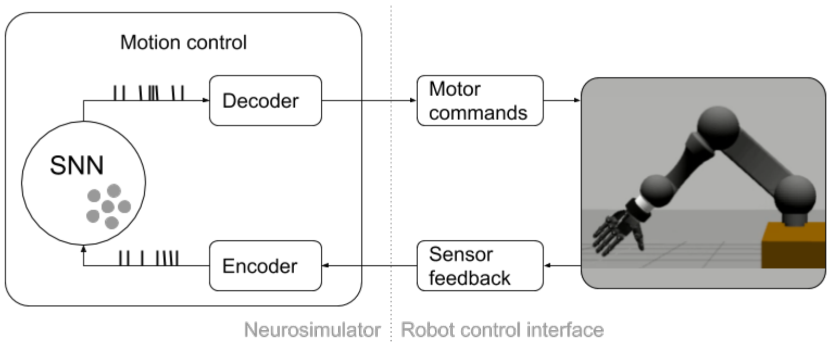


Figure 3.10.: Closed-loop control schema with SNNs. Schemas for encoding and decoding spike trains are incorporated for communication between the neurosimulator and the robot control interface.

3.5.1. Distributed representation for encoding and decoding

One mechanism to represent sensor or control values with spikes is to use the activity of a population of spiking neurons as a distributed representation (Tieck et al. [25, 24]). Thus, different values correspond to different activity patterns of the whole population and not of individual neurons. This method is based on the principles of the NEF (Eliasmith et al. [99] and Stewart [207]).

For encoding, the parameters of neurons in a population — for example, the bias currents — are initialized with a random distribution. Neurons can be driven by individual spikes or by the input current (see Figure 2.19b). In biology and neuromorphic hardware, all neurons have different physical properties; thus, this initialization resembles this process. Once the neurons are initialized, an encoder vector is generated for the population. There are many methods of doing this initialization. The encoder is calculated once when the population is initialized. A general approach to generate the encoder vector is to use a uniform random distribution with regular intercepts in the input signal interval. The input signal is translated to current, and it is multiplied by the encoder vector to activate the neurons. The relation between changes of the input signal with the neuron's firing rate can be seen with the tuning curves (see Figure 3.11a). The tuning curves are generated by changing the input current of each neuron and recording its firing rate. Changes in the neuron parameters, e.g., bias current or refractory period, and changes in the encoder vector affect the tuning curves. Finally, the spike activity of the population encodes the input signal. The spike plot in Figure 3.11b shows spike trains for different input signal values.

For decoding, the spike activity of the neurons is filtered with an exponential decay temporal filter (see Figure 3.11c). A constant linear decoder vector weights the response of each neuron. The estimated decoded value is calculated by multiplying the filtered spike trains with the decoder vector and adding them (see Figure 3.11d). The decoder vector is optimized to find a set of weights that minimize the difference between the function to be represented and its estimate. In this case, the function is sinusoidal. The decoder vector is calculated when the network is initialized, but it can be modified online using learning rules like Prescribed Error Sensitivity (PES) (see Section 3.6). A detailed explanation about the encoding and decoding mechanisms of the NEF are provided by Eliasmith et al. [99] and Stewart [207].

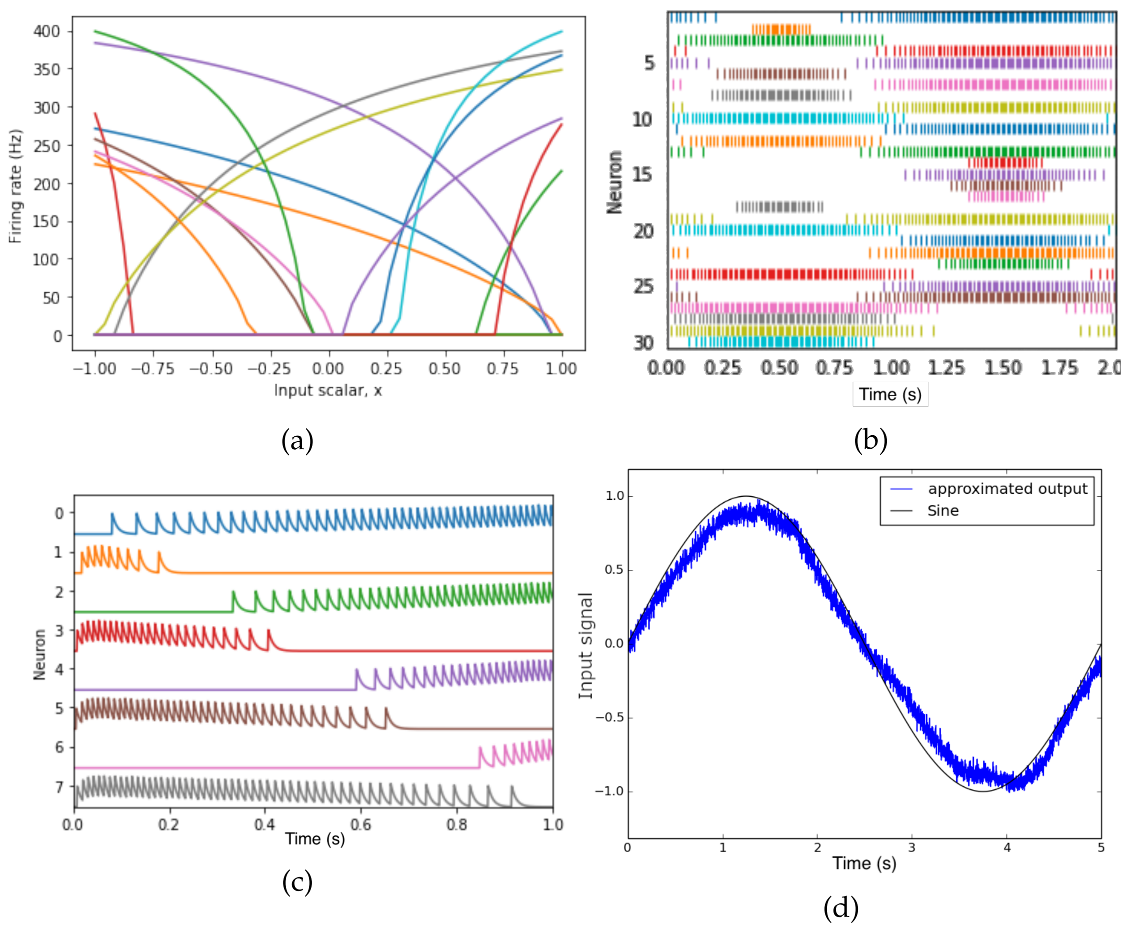
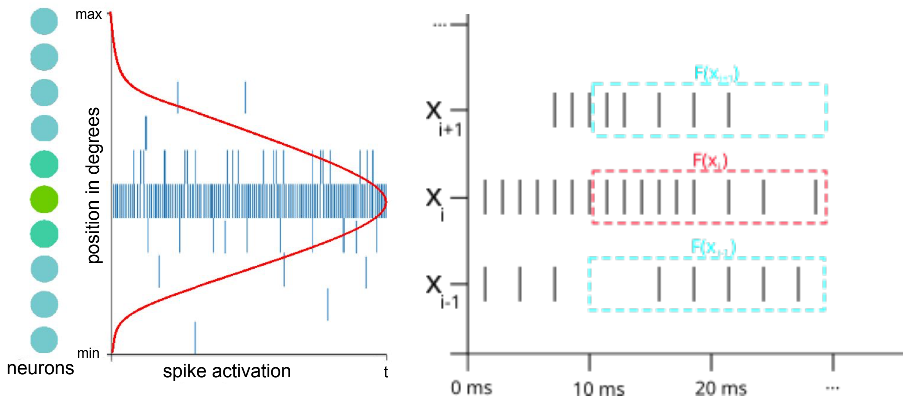


Figure 3.11.: Encoding and decoding with the NEF, different colors indicate different populations' neurons. (a) Tuning curves define the firing rate with respect to the input signal (Stewart [207]). (b) Spike trains for different values of the input signal. (c) The spikes trains are filtered with a temporal filter (exponential decay) (Nengo [163]). (d) Approximated output is calculated by multiplying the filtered spike trains with the decoder vector and adding them together.

3.5.2. Stochastic Gaussian population position encoding

An alternative mechanism is proposed to represent a value with spikes applying stochastic Gaussian curves for encoding and decoding (Tieck et al. [16]). This mechanism is based on the proprioception of muscles with receptive fields (Arbib [47] and Wu et al. [224]). A stochastic population encoding with Gaussian receptive fields encodes the robot's joint positions into neural input. Indeed, it is possible to apply this method for both encoding and decoding with the same schema. However, this is reasonable and efficient because the robot kinematics are represented in joint space. In contrast to this, biological muscles proprioceptors react to local forces, and in the brain, different representations can be observed.

This mechanism is called stochastic population position Gaussian coding. For the encoding, values are converted into spike trains using a mixture of Gaussian kernels to tune the firing rate of Poisson generators for the neurons in a population. A population of neurons maps the joint interval to a Gaussian activation curve centered on the actual joint angle. This Gaussian receptive field is applied to define the activation frequency of the neurons. An illustration for encoding an arbitrary angle is presented in Figure 3.12a. This scheme is mapping values of consecutive states by sharing part of the activation. Different receptive fields encode values in a normalized interval. The populations are sub-divided to quantize the interval with an overlap of the Gaussian curves. For decoding, a voting schema is applied to determine the central activation of the Gaussian spike distribution (Figure 3.12b).



(a) (b)

Figure 3.12.: Neural encoding and decoding schemas. (a) For encoding, joint positions are converted into spikes by having each of the neurons on the left represent an equal share of the possible joint interval, and the Gaussian curve defines the activation frequencies (Tieck et al. [24]). (b) For decoding, a voting schema $F(x)$ determines the central neuron x_i of the activation to center the Gaussian curve (Tieck et al. [16]).

3.5.3. Data normalization

An important consideration for representing information with ANNs and especially SNNs is to normalize the data. Normalization ensures that the values are bounded to an interval represented by the network, which makes learning more flexible and that the network's learning considers all input features to a similar extent (LeCun et al. [146]). A spiking neuron has a limited firing rate, which limits the values it can represent. This is critical for multi-layer networks as consecutive layers will saturate the firing rate. Additionally, if the inputs have different scales, the weights associated with some inputs will be updated much faster than others (Eliasmith [98]). A standard approach is to scale the inputs to have a mean 0 and a variance of 1 into the interval $[-1, 1]$. Another approach is to scale the inputs with a scaling factor and transforms all the values into the interval $[0, 1]$.

3.6. Mechanisms to learn motor primitives with SNNs

In the following, the different learning mechanisms to learn motor primitives with the concepts from Section 2.3 are presented. A method to learn function representations offline by optimizing the decoders using NEF is presented in Section 3.6.1. A method for learning human demonstrated motions offline by using associative learning and a teaching signal is proposed in Section 3.6.2. Finally, a method for learning online by modifying previously learned motor primitives is proposed in Section 3.6.3.

3.6.1. Learning functions by optimizing the decoding weights

A mechanism to represent motions with SNNs, is to learn a motion offline optimizing the weights of the network as described in NEF (Eliasmith et al. [99] and Stewart [207]). The motion can be defined as a function or as a set of waypoints in the trajectory (Tieck et al. [25]). A complex function has to be broken down into vectors, functions, and differential equations, which are learned in different sub-networks. The connection weights for each sub-network are optimized separately, and then they are combined into one large neural network. This approach has two advantages. First, the definition of what the sub-network does is explicit and understandable. Second, finding the connection weights locally using optimization enables the generation of large complex systems without using the traditional neural network approach of optimizing over huge amounts of training data. However, the trade-off is that explicit claims must be made about what each sub-part of the model is doing.

For any non-linear neuron model G, assuming that the input current x of a neuron is proportional to the value it represents, the activity a can be defined as

$$a_i = G(\alpha_i e_i \cdot x + b_i) \tag{3.6}$$

with α_i a gain parameter and b_i is a constant bias current for the neuron (Stewart [207]). Based on the principles described in Section 3.5.1 for encoding and decoding information into a SNN, the learning problem is to find a linear decoder $\vec{d_i}$ that represents the desired output $\vec{x}$. This decoder maps the activity of the network to an estimate $\hat{x}$ of the desired output x as:

$$\hat{X} = \sum a_i \vec{d_i} \tag{3.7}$$

Here, $\vec{d}$ can be solved as a least-squares minimization problem. According to Stewart [207], the solution for $\vec{d}$ is defined as

$$\vec{d} = \Gamma^{-1}\Upsilon, \quad \Gamma_{ij} = \sum_x a_i a_j, \quad \Upsilon_j = \sum_x a_j \vec{x}. \tag{3.8}$$

Now, with Equation (3.8) the activity of a sub-network or group of neurons can be decoded to represent the desired output. To build complex systems, the output of a sub-network is connected to the input of another sub-network with a different function, and this process can be repeated multiple times. In order to create this connections, an intermediate layer of ideal linear neurons is defined (see Figure 3.13a). The first part $\vec{d}$ is what is needed to compute $\vec{x}$ given the activity of a sub-network A. Then the second part $\vec{e}$ is what is needed to compute $f(x)$ given $\vec{x}$. This abstraction is the ideal representation, and there is no intermediate layer. However, they can be removed by connecting A and B directly with $\omega_{ij} = \vec{d_i} \cdot \vec{e_j}$ as shown in Figure 3.13.

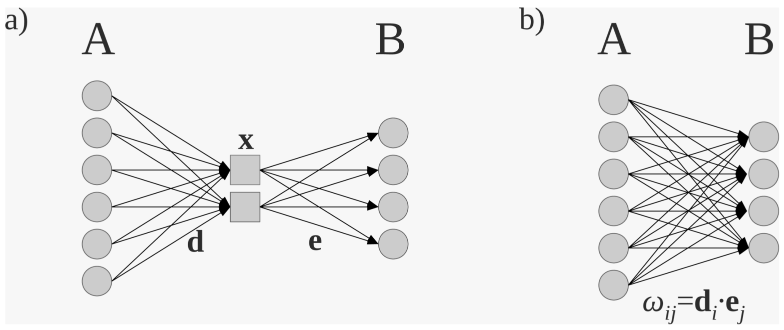


Figure 3.13.: Learning with NEF (Stewart [207]). Circles are any neuron model G with gain α_i and bias bi. Squares are idealized perfectly linear components. a) Shows how Equation (3.7) computes x from a_i using weights d and how Equation (3.6) combines x with e to compute the input current to the next layer of neurons. b) Eliminates these idealized components, giving a realistic neuron model functionally identical to a).

Equation (3.8) can be generalized for any function (Eliasmith et al. [99] and Stewart [207]). The precision of the function's representation depends on the number of neurons and the characteristics of the function. This method is used to represent motions by modeling the function that describes a specific trajectory or by a sampling of the trajectory in joint space.

3.6.2. Learning sequences with associative supervised learning

A supervised associative learning mechanism based on STDP is proposed to learn sequences (Tieck et al. [16, 20]). During learning, a teaching signal is connected one-to-one to the output layer. For noise reduction in the output layer, a winner-takes-all circuit is incorporated. Sub-networks like the ones for the arm, hand, fingers, or legs can be independently trained. The plastic synapses are initialized with random weights $\in (0, 1]$. This method is inspired by the work of Bouganis et al. [57].

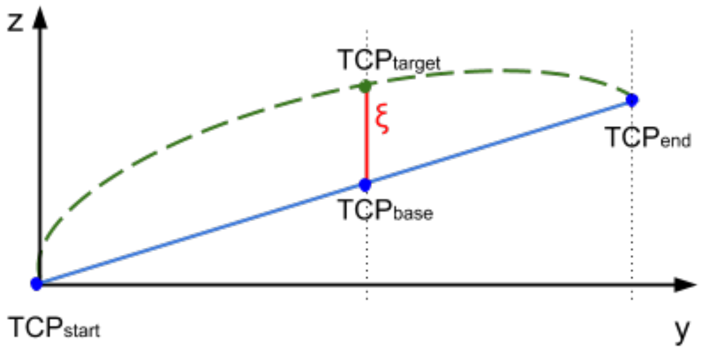
With the training data, the direction vector of the motion is calculated for each fingertip and the desired motor commands to feed the hand and finger networks, respectively. The teaching signal encodes the desired activation. It raises the membrane potential of specific output neurons just below the firing threshold and inhibits all others. Thus, the activation of input neurons triggers STDP for the desired activation. This approach is called *selective disinhibition* (Sridharan et al. [202]) and is used to prevent input and teaching signals from competing. After learning, the SNN associates the input with the desired output and reproduces the teaching signal.

To regulate STDP, each neuron in the output layer has incoming synapses from two corresponding neurons in the input layer, one excitatory and inhibitory. These two neurons work as one having complementary activations (the sum of their firing rates is constant). For excitatory synapses, an anti-symmetric STDP learning rule and a weight-dependence rule are used (Gütig et al. [116]). The inhibitory synapses use an anti-symmetric anti-Hebbian STDP rule using additive weight-dependence. Jointly tuned excitatory and inhibitory STDP in feed-forward SNNs lead to a local synaptic balance to make the network sensitive for learning new synapses and prevent fast increase of the weights (Kleberg et al. [138]).

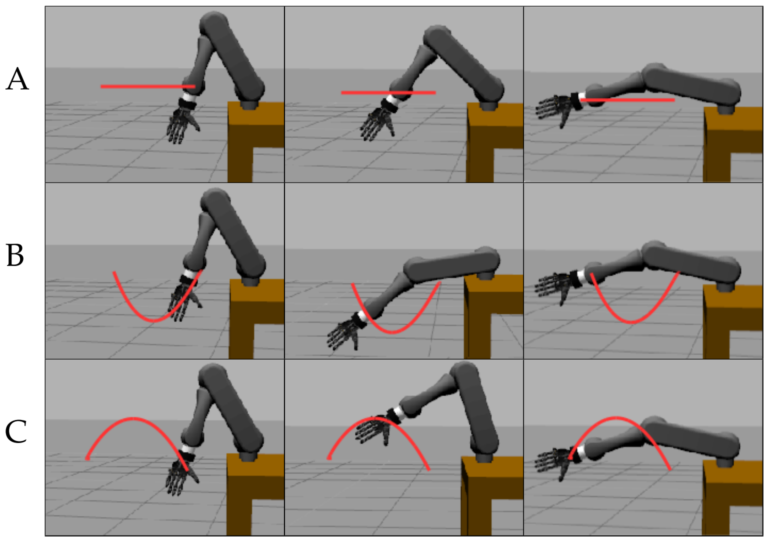
3.6.3. Adapting motions with online learning

Motions are represented as primitives modeled as $g(f(u))$ (see Section 3.2). The different activation modalities were modeled as different mechanisms to generate u (Section 3.3). The joint activation function f is always the same for all joints. In this section, a mechanism is presented to learn new motor primitives by learning the function g based on existing primitives, thus reusing previous knowledge (Tieck et al. [24]).

The principle is illustrated in Figure 3.14a. A motor primitive (*continuous line*) is used as base, and a target trajectory (*dashed line*) represents the desired motor primitive to learn. The target trajectory is generated in another brain area but is modeled as an arbitrary function. The network learns to move the TCP along the target trajectory using the distance from the TCP to the base trajectory as the error signal. Two examples illustrate the process in Figure 3.14b. A linear motion is selected as the base (see *first row* of Figure 3.14b). The error signal ξ is calculated in spherical coordinates with reference to the shoulder. The desired motion (see second and third row in Figure 3.14b) is implicitly encoded in the error signal. Thus, the initial motor primitive is reshaped by learning, according to the target trajectory. Finally, the network learns to move the TCP along that trajectory.



(a)



(b)

Figure 3.14.: Learning new motor primitives (Tieck et al. [24]). (a) The *continuous line* is the base primitive. The *dashed line* is the target primitive. The error signal ξ is defined as the distance between $\text{TCP}_{\text{target}}$ and TCP_{base}, in spherical coordinates to the shoulder. (b) Using an already learned motor primitive *A* as a base motion to learn two different target trajectories *B* and *C* as new motor primitives.

For learning, the Prescribed Error Sensitivity (PES) rule is used. PES was first introduced by MacNeil et al. [153]. The weight updates performed by PES during learning, resemble skipped Back-propagation. PES modifies the decoders of a connection to minimize an error signal. The change in weights w_{ij}, is given by

$$\Delta w_{\mathrm{ij}} = \kappa \cdot \alpha_j \cdot e_j \cdot \xi \cdot a_i, \tag{3.9}$$

where α is the gain, κ the learning rate, ξ the error, *e* the encoder of the neuron and *a* the presynaptic activity. The pre-synaptic population is indexed by *i* and *j* indexes the post-synaptic population.

3.7. Summary

The taking points of this chapter are the following. It has been shown that there is a parametric way of modeling a motor primitive with an activation parameter and a mapping function. This modeling reduces the control parameters of the controlled joints to one activation parameter for each motor primitive. The mapping to the robot kinematics is the only part that is robot specific, and it can be modified to another target system. There are different activation mechanisms for the parameter that controls a motor primitive. It can be activated with different modalities — voluntary or intentional, rhythmic or repetitive, or reflex. There are also mechanisms for contact detection and selective disinhibition. In order to model complex motions and behaviours, motor primitives can be organized in different layers in a hierarchy. This structure allows the combination and parameterization of primitives. This allows the reuse of low-level primitives for different motions. The schemas for encoding and decoding spike information are presented with a closed-loop control schema. There are different mechanisms to learn new motor primitives, either online or offline, and either modeling the motion or using human demonstrated motions.

In the following chapters, SNNs are systematically studied to develop building blocks for different control problems. After modeling and formalizing the building blocks to work with motor primitives using SNN, the next step is to apply these principles with applications in different problems. With the help of robot experiments, different SNN based control modes are discussed and evaluated regarding the underlying learning strategies, complexity, and robustness. The next chapters present three types of applications: arm (Chapter 4), hand motion (Chapter 5), and multi-legged locomotion (Chapter 6)). In all cases, biological plausibility is discussed.

4. Combination and activation of motor primitives to control robotic arms

In robotics, manipulation is a fundamental task that is performed before and after grasping. Recent studies provide insights into the neural mechanisms for motion generation in the motor cortex. During reaching, neural activity in the motor cortex shows a short but strong rotational component (Churchland et al. [70] and Russo et al. [184]) and a strong and amplified but stable response to initial activation (Hennequin et al. [119]). Additionally, neural correlates of many different types of parameters of arm movements have been found in the motor cortex (Kalaska [134]). Studies have shown that the human brain incorporates feedback information from vision and proprioception to execute reaching movements (Saunders et al. [188] and Filimon et al. [102]). The link between these two systems implies that other important components are involved in the generation of motion. Humans can quickly determine which object is in front/back or on the left/right of another one, which of two angles is wider, and then move the arm accordingly (Pfeifer et al. [169]).

This chapter presents methods and experiments to discuss and evaluate control principles for different motions of a robotic arm using motor primitives and SNNs. The arm motion is modeled by defining primitives to perform simple motions — moving left-right, for example — and combining them with different mechanisms. In the following, three scenarios are considered: multimodal activation, target reaching, and motion adaption.

Multimodal activation represents the mechanisms to activate, combine, and parameterize motor primitives directly. Three mechanisms for motion activation are modeled: voluntary, rhythmic, and reflex. A voluntary or intentional activation controls the motion continuously and can start or stop at any point in it — for example, moving the arm to point to an object. A rhythmic activation goes from the beginning to the end of the motion and repeats it multiple times — for example, moving the legs for walking. A reflex activation is a complete one-time execution of the motion — for example, the retract reflex of the arm. Additionally to the different activation modalities, combining different motor primitives and parameterizing them is a challenge.

Motion adaption is modeled by combining a base motion with a set of correction motor primitives offline using a proportional target representation. It is defined as the pre-shaping of pointing motions in a plane. Offline means that the motion is generated in predictive feed-forward mode before the movement. The purpose is to generate new motions offline by adapting a given set of motions to point at different targets in a plane.

Target reaching is modeled with three correction primitives — left-right, up-down, far-near — and the target is represented as discrete negative feedback to activate the motion. It is defined by an initial state of the robot arm and a target position in space, move the TCP of the arm to the target. The idea is to control the arm to reach specific targets combining the motions online using sensor feedback. The challenges are how to model this control problem with motor primitives and how to incorporate target information into the network.

4.1. Modeling the motion of a robot arm

The experiments are performed with commercially available robotic arms and a humanoid robot with two arms. The control interface is based on ROS, which allows communication between the SNNs and both the simulation and the real robots. With all robotic systems, the same abstraction of the arm kinematics is applied, and the control is made with the joint position. For the experiments, the arm is considered to have three active DoF — two in the shoulder and one in the elbow (see Figure 4.1).

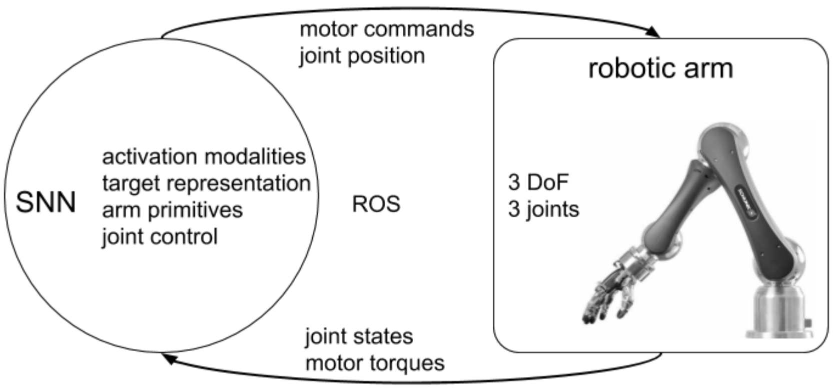
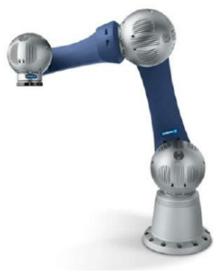


Figure 4.1.: Arm motion modeling. The robotic arm is controlled with the joint position interface. It has three active DoF, and it has no force or contact sensors. Primitives are defined for different groups of joints. There are different representations of the target. Different modalities are integrated to activate the primitives.

An output population is defined for each of the joints to control the arm with SNNs. Each primitive represents the synergies between the joints during a simple motion of the arm. A primitive can activate one, two, or all three joints. The base motions are represented by sampling the joint states of the robot while moving it manually in a similar way to the corresponding human motion. Then, the primitive sub-networks are trained to generate the sequences. The targets are represented in different ways to create different behaviours. Different primitives are activated with different modalities and combined to generate complex motions. There is a contact detection mechanism to trigger reflexes.

The robotic arm is an industrial Schunk Powerball Lightweight Arm LWA 4P (see Figure 4.2a). It is developed by Schunk for industrial and research tasks, such as manipulation and human-robot collaboration. All the electronics are integrated into the joints and the base. This robot offers a standard platform that resembles many industrial arms. To control the robot arm, the official ROS CANopen driver is used (Heppner [120]). The arm has two DoF per joint for a total of 6-DoF.

The humanoid robot HoLLiE (Hermann et al. [123]) is a mobile service robot with two industrial arms and two humanoid hands (see Figure 4.2b). The robot is developed at FZI Research Center for Information Technology (FZI) for service tasks, such as guiding visitors and mobile manipulation (FZI [105]). With a wide range of sensors and an articulated body, HoLLiE can manipulate everyday objects, interact with humans, and be employed in service robotic scenarios. For these characteristics, HoLLiE was chosen to achieve human-like pointing motions, as the arms are mounted on an upper body with similar kinematics to a human arm. The control interface is based on ROS control. Each arm has 6 DoF, each hand has 9 DoF, and the head and torso are also articulated.


(a)

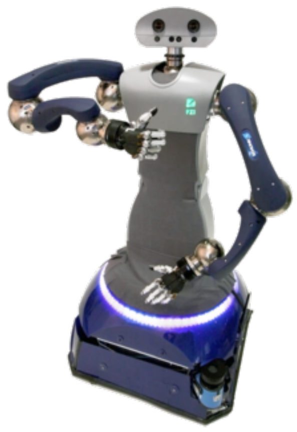

(b)

Figure 4.2.: Robots used for the arm experiments. (a) Robotic arm Schunk Powerball Lightweight Arm LWA 4P (Memar et al. [157]) (b) Humanoid robot HoLLiE (FZI [105]).

4.2. Activation modalities and combination of motor primitives

By modeling motion with motor primitives, the problem of generating a specific motion is mapped to the problem of how to activate the primitives. Humans and animals can change their motions efficiently and intelligently to adapt to changes in the environment, react to unforeseen events, or learn new skills. In nature, there are three main activation modalities: voluntary, rhythmic, and reflex (Churchland et al. [70] and Byrne et al. [64]).

A voluntary or intentional activation controls the motion constantly and can start or stop at any point of it, for example, moving the arm to point to an object. A rhythmic activation goes from the beginning to the end of the motion and repeats it multiple times, for example, waving the arm. A reflex activation is an immediate and complete one-time execution of a motion, for example, the retract reflex of the arm. Additionally to the different activation modalities, there is the challenge of how to combine different motor primitives and how to parameterize them. In this sense, multimodal activation represents the mechanisms to activate, combine and parameterize motor primitives directly. An illustration of the problem is presented in Figure 4.3.

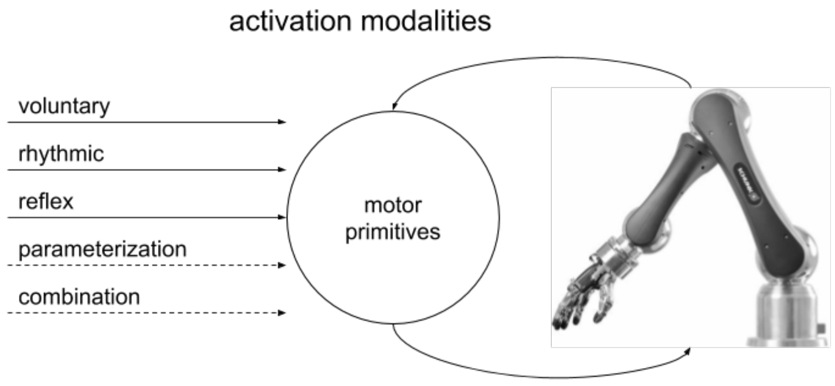


Figure 4.3.: Multimodal activation problem definition. Given a set of primitives, activate them in a voluntary or intentional way, or with many repetitions in a rhythmic way, or with a one-time execution as a reflex. Additionally, parameterize the activation and combine different activation modalities.

Biology provides two specific aspects that can be exploited for robotic systems. First, motion is represented in a hierarchical and distributed way in the body and the nervous system (Pfeifer et al. [169]). Several areas are involved in performing different parts of the motor control — e.g., the brain, the cerebellum, the spinal cord, and the muscles. Second, motions can be activated in different modalities. Multiple control processes are operating in parallel that alternate operation with time-variant priorities and switch the motion activation (Churchland et al. [70] and Bernstein [54]).

In Ijspeert et al. [129] a mathematical model was proposed to combine and parameterize different primitives. An extended mathematical model for central pattern generators in the spinal cord, generating several different motion patterns by changing a small set of parameters, was introduced in Nassour et al. [161]. The architecture has separate layers for pattern formation, rhythm generation, and motor neurons.

In the following, a framework to perform multimodal motion activation is presented, integrating a mechanism to learn new motions using the principles described in Chapter 3. The general architecture of this approach is presented in Figure 4.4. This section incorporates findings and methods from Tieck et al. [24].

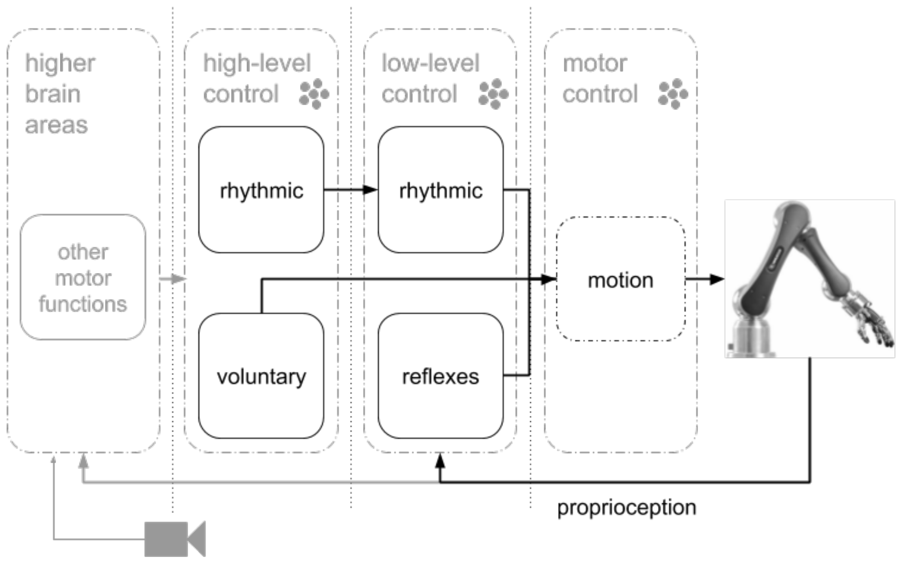


Figure 4.4.: General view of the closed-loop architecture with SNNs for multimodal motion activation (Tieck et al. [24]). From right to left, the layers have an increasing level of abstraction. In *motor control* there are the base motions and joint control, followed by *low-level* and *high-level* control representations with three activation mechanisms — voluntary, rhythmic and reflex. Higher brain areas and vision are included greyed out for completeness.

Low-level motor primitives are combined as base building blocks to represent motions. The neural output of a primitive is translated to robot motor commands. Complex motions are represented as combinations of primitives. Three types of modalities to activate the motions are modeled: voluntary (also called intentional), rhythmic, and reflexes. The method was successfully tested in several scenarios in simulation. Motions can be reused, combined, and parameterized. SNN are especially interesting for multimodal motion activation because modeling the problem with spikes provides an intuitive way of combining different control signals, switching between activation modalities, and parameterizing motions.

4.2.1. Methods

To illustrate the idea, imagine a person in front of a table stretching the arm above the table parallel to the surface without touching it and then retracting it back. These two motions represent simple primitives for the arm. The brain will reuse motions for other similar tasks, as they represent common patterns of motor activation. They can be used to point at an object or reach a target as a voluntary or intentional activation. They can also be alternated and repeated to clean a table or waive as a rhythmic activation. The retract motion is also used if the arm hits an obstacle while moving, activated as the withdrawal reflex.

In Figure 4.4 the different layers, the main components, and the information flow are shown. Complex motions are represented in a hierarchy combining motor primitives. The motor primitives are modeled as in Section 3.2. Three different motion activation modalities are modeled: voluntary, rhythmic, and reflexes. The individual activation modalities are presented in Section 3.3. The layers are arranged in increasingly abstraction, starting with motor control of the joints and base motions, followed by low-level and high-level representations. The encoding and decoding schemas used to get information in and out of the SNNs are presented in Section 3.5.

A detailed view of the architecture showing the neural populations in all layers is presented in Figure 4.5. The populations are defined individually with specific functions. The motor control layer provides the representation including motor primitives, motor activation through motor neurons, and the mapping to the target robot system. This layer receives excitatory and inhibitory input from other layers. Depending on this input, the different motor primitives are parameterized and combined.

The low-level control layer takes input from higher layers and provides two activation mechanisms. First, rhythmic activation (Section 3.3.2) to generate a continuous and repetitive activation of the primitives. Second, the activation with reflexes (Section 3.3.3) incorporates proprioception and either inhibits a currently executed primitive or triggers an immediate one-time execution of another primitive. To detect contact and trigger the retract reflex, the mechanism described in Section 3.3.4 is incorporated using the joint efforts and joint states.

The high-level control layer provides the voluntary activation (Section 3.3.1) for the primitives and another layer of rhythmic activation to provide parametrization for the previous layer. On top of that, different activation signals are provided by higher layers to combine, parameterize and select the motions. The higher brain areas layer represents action selection, motion planning, visual input processing, and other sensory stimuli processing tasks required for motor control.

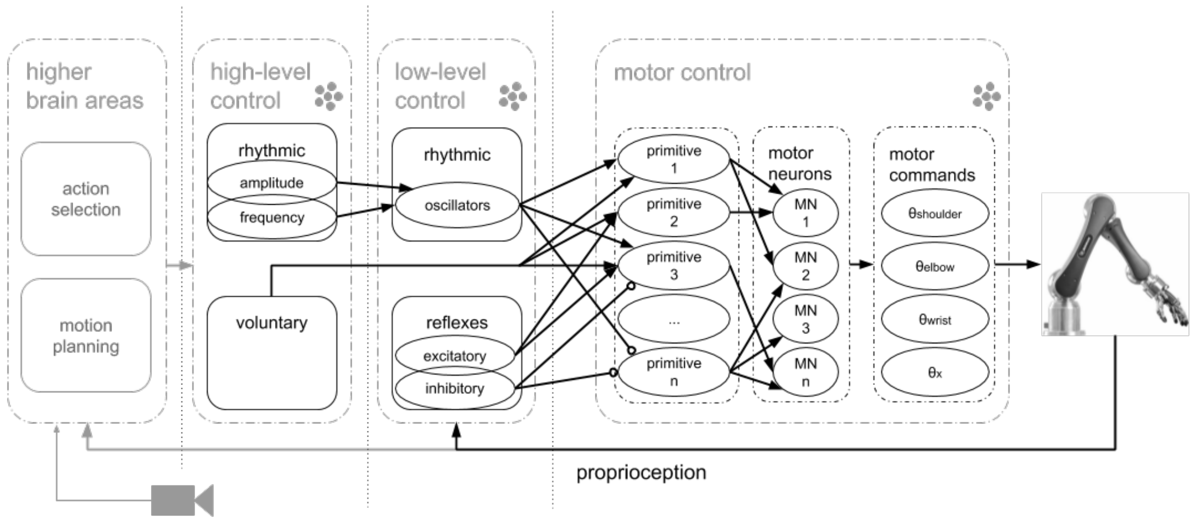


Figure 4.5.: Detailed view of the SNN in closed-loop showing the motor control hierarchy (Tieck et al. [24]). The *motor neurons* generate *motor commands*. The *motor primitives* are activated by three activation modalities — voluntary, rhythmic, and reflexes. For the *rhythmic* activation, there are oscillators and their parameters — amplitude and frequency. For the excitatory and inhibitory *reflexes* there are interneurons and neural circuits. The *proprioception* includes joint states and joint efforts. Higher brain areas and vision are included greyed out for completeness.

4.2.2. Experiments

Different scenarios were defined to combine and test the different activation modalities to evaluate the SNN architecture. First, the three modalities were evaluated individually. After that, several combinations were evaluated: voluntary and reflex, voluntary and rhythmic, rhythmic and reflex, and finally, a combination of all three modalities. The SNN is implemented in the Nengo[1]. A *Schunk LWA 4P* robot arm is simulated with the *Gazebo Simulator*[2]. The communication between the components and both simulators is implemented using ROS.

Voluntary motion activation

A voluntary or intentional motion is used to extend the robot arm and retract it. This type of motion is used to point at given targets. The motion is performed on-demand controlling a parameter to change the value of u manually. The activity recorded during the execution is presented in Figure 4.6. In segment A the changes in the shoulder $\theta_{shoulder}$ and elbow θ_{elbow} joints correspond to changes in u and $f(u)$. After

[1] http://nengo.ca/
[2] http://gazebosim.org/

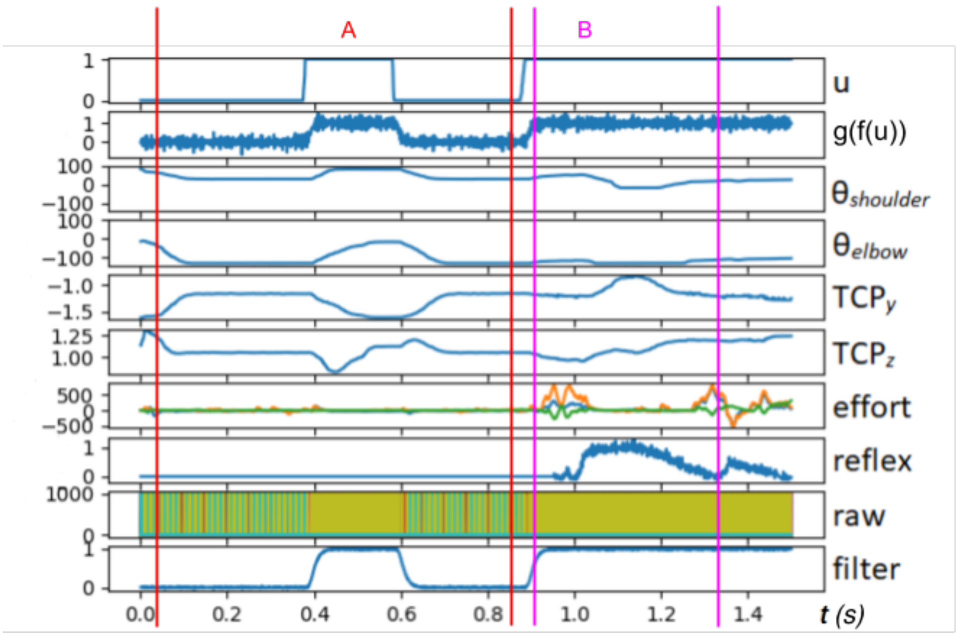


Figure 4.6.: Voluntary activated motion, interrupted by a reflex (Tieck et al. [24]). Rows 1 and 2 show the activation u and the motor primitive $g(f(u))$. Rows 3 and 4 show the angles of the shoulder $\theta_{shoulder}$ and elbow θ_{elbow} joints in degrees. Rows 4 and 5 display the position of the TCP in the YZ plane. In row 7, the efforts on the wrist, elbow, and shoulder joint are plotted. In row 8, the activity of the population that activates the *reflex*. In rows 9 and 10, the *raw* spike output, and the *filter* motor output.

that, the arm is fully extended, and it collides with an object. The contact triggers the a reflex activation to retract the arm as shown in segment B, notice how the *effort* increases and a *reflex* is triggered.

Rhythmic motion activation

A rhythmic or cyclic motion is used to move the arm back and forth to generate waving motions.For the rhythmic motion activation, instead of activating the motor primitive through a discrete signal, an oscillatory input is used to generate a continuous activation (see Section 3.3.2). In Figure 4.7 a rhythmic activation is used to control the shoulder joint of the arm. The motion is parameterized by having populations control the amplitude a and the frequency b of the oscillation. Observe how the activation of u changes with different values of a and b, which generates corresponding motor commands for the $\theta_{shoulder}$ joint. The rhythmic behavior is also observed in the TCP positions in the YZ plane.

So far, the experiments have shown that both rhythmic and voluntary modalities can be activated independently. Another experiment combined a voluntary activation

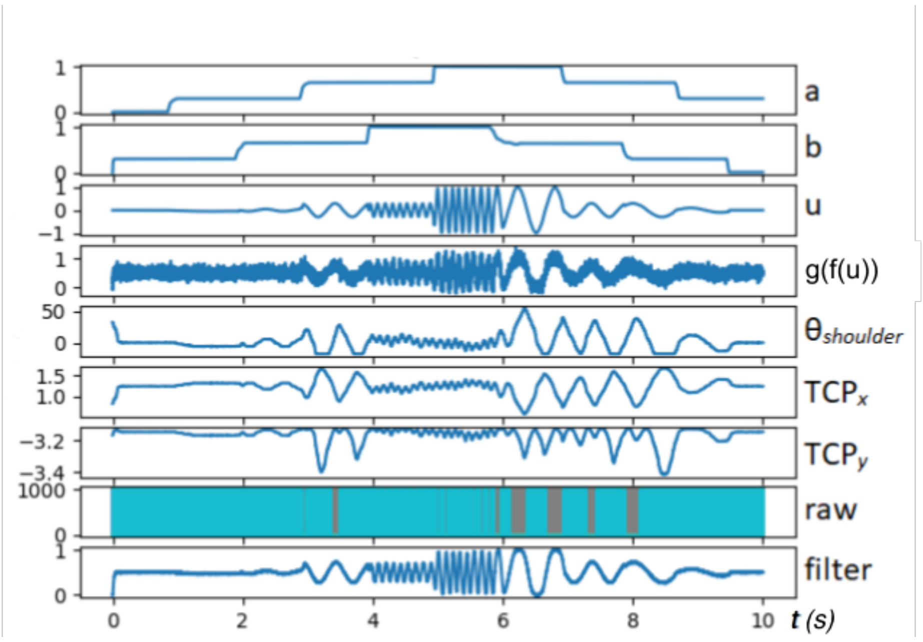


Figure 4.7.: Rhythmic activated motion with different parameters (Tieck et al. [24]). Rows 1 and 2 show the parameters a and b that control the amplitude and frequency of the motion. Rows 3 and 4 show the activation u and the motor primitive $g(f(u))$. Rows 5 shows the angles of the shoulder $\theta_{shoulder}$ joint in degrees. Rows 6 and 7 display the position of the TCP in the YZ plane. In rows 9 and 10, the *raw* spike output, and the *filter* motor output.

with a rhythmic activated motion for a two-fold activated movement. An actual application for such a combination is a task like wiping a table. The rhythmic part performs the basic wiping motion, and the voluntary part makes sure that the desired table area is covered.

Activation of reflexes

Two different reflexes were implemented to initiate an alternative withdrawal motion and inhibit the motion being executed. The contact detection intercicuit described in Section 3.3.3 that incorporates dynamic proprioception (motion) and the effort in the joint is used to detect contact. When a threshold is reached in the interneuron, the oscillator is triggered, and a reflex is activated.

The withdrawal reflex is evaluated while performing voluntary or rhythmic motions. In Figure 4.6 in segment *B* a voluntary motion is interrupted because the arm hits an obstacle. In Figure 4.9 the reflex activation can be seen multiple times in the sectors marked with *C*.

The inhibitory reflex is evaluated during a rhythmic motion. In Figure 4.8 an inhibitory reflex is triggered (see segment *B* row 3)) during a rhythmically activated motion when an obstacle is hit. The input of the motor primitives is inhibited, and the motion is stopped (see segment *B* rows 4 and 5). The spike plot ($raw_{rhythmic}$, the decoded value $filter_{rhythmic}$) and the joint angles (θ_E and θ_W) show how the motion being executed is inhibited. The output spike train stops, and the joint angles maintain their values after inhibition.

Combination of activation modalities

An experiment was made with complex motions activated by all three modalities to show the interplay of the activation modalities with a hierarchy of motor primitives. A rhythmic motion controlling the wrist and the elbow joints performs a waving motion. A voluntary motion is activated after a while, controlling the shoulder joint to move the arm right and left. An obstacle is placed in the motion trajectory, and when the arm hits it, a reflex is triggered. Plots of this scenario are presented in Figure 4.9. During the segment *A* only voluntary activated motions are performed. During the segment *B* a rhythmic motion with varying parameters is performed. The segment marked with C represent a motion interrupted by an excitatory reflex.

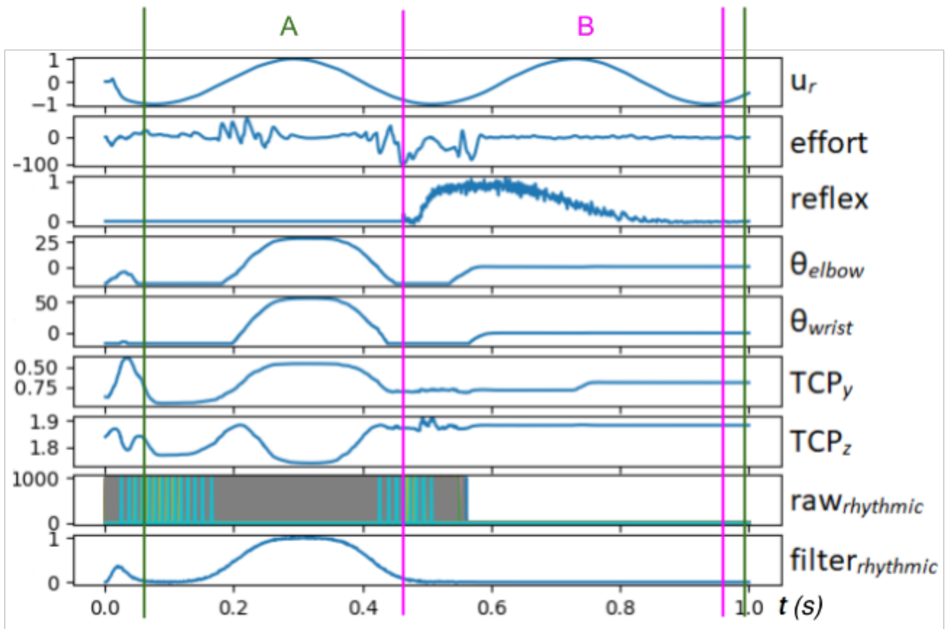


Figure 4.8.: Inhibition reflex during rhythmic motion (Tieck et al. [24]). Row 1 shows the rhythmic activation u_r. Row 2 displays the effort on the elbow joint, which is used to trigger the *reflex*. Row 3 shows the activity of the population that activates the *reflex*. Rows 4 and 5 show the angles of the elbow θ_{elbow} and wrist θ_{wrist} joints in degrees. Rows 6 and 7 display the position of the TCP in the YZ plane. In rows 8 and 9, the *raw* spike output, and the *filter* motor output.

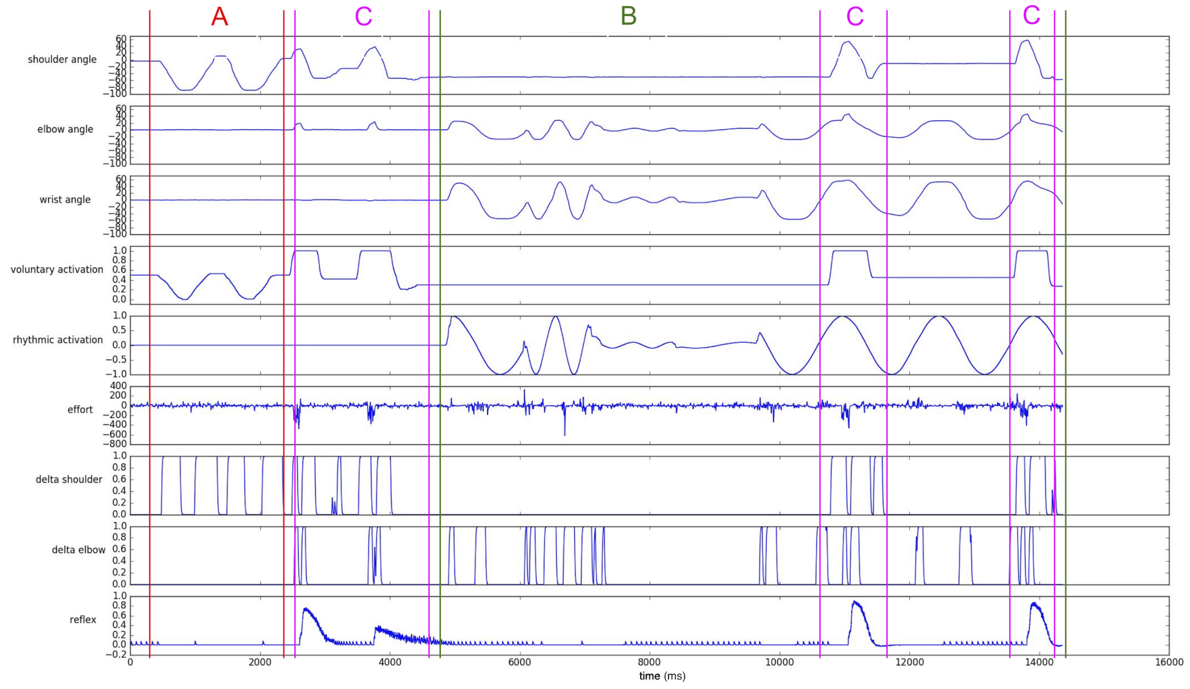


Figure 4.9.: Combination of all activation modalities (Tieck et al. [24]). Rows 1,2 and 3 show the joint angles of the shoulder, elbow, and wrist joints. In rows 4 and 5, the activation of the primitive for the voluntary (controlling the shoulder) and the rhythmic (controlling elbow and wrist) parts are plotted. Row 6 displays the effort on the joints, which is used to trigger the *reflex*. Rows 7 and 8 show the delta (motion) of the shoulder and the elbow joints; if the delta value is *1*, the corresponding joint is moving. Row 9 shows the activity of the population that activates the *reflex*.

4.2.3. Discussion

The proposed framework for motor control with SNNs allows activation, combination, and parameterization of motor primitives to control a robot arm. The representation of the motions is hierarchical, going from simple joint movements to complex arm motions and integrating signals from other higher brain areas. Motions are represented with SNNs using motor primitives as fundamental building blocks and can be activated by different modalities. This representation allows the combination and parameterization of motions. A mechanism using the efforts in the joints and the joint position changes was integrated to detect contact and trigger a retract reflex.

Multimodal activation was implemented and used to perform different complex movements and control a robot arm in simulation. The three modalities — voluntary (intentional), rhythmic, and reflexes — were implemented and evaluated individually and combined in different scenarios. An experiment combining all the components was presented (see Figure 4.9). A single motor primitive activated as a rhythmic motion was combined with a voluntary motion, and contact was generated with a collision that triggered a retract reflex on the arm. A framework with these capabilities can be extended to perform complex behaviors as in Nassour et al. [161].

For these experiments, the raw output of the network was used to control the robot. A muscle model could be integrated to smooth the control signals by damping irregular spike trains from the output populations. Although a mechanism for learning new primitives was integrated (see Section 3.6.3), there are open questions on extending the network for the new motions and also on how and when to enable and disable learning. One alternative to modulate learning could be implementing the hPES-rule in Nengo, or alternatively, PES could be combined with the triplet-based STDP rule as introduced in Pfister et al. [172].

4.3. Generation of pointing motions for targets on a plane in 3D space

In Section 4.4 a method to combine motions online was presented. In contrast, this section presents a method for offline motion adaption that combines the motions as a whole before their execution. The human motor system is a topic of multidisciplinary research for a considerable period. However, robots lack robust, flexible, and adaptive controllers comparable to the human sensorimotor system (Pfeifer et al. [169]). One specific example is the capability to generate and pre-shape motions before its execution (Shenoy et al. [198]).

Motion adaption in this context is defined as the pre-shaping of pointing motions on a plane in 3D space. Offline means that the motion is generated before its execution. The challenge is to generate a new motion offline by adapting a set of already learned motions to reach different points on the plane. In this sense, motion adaption is modeled by combining a base motion with a set of correction primitives using a proportional target representation. The problem is presented in Figure 4.10.

Recent studies provide insights into the mechanisms for motion generation in the human motor cortex. During reaching, activity in the motor cortex as a whole shows a brief but strong rotational component (Churchland et al. [70] and Russo et al. [184]). Instead of encoding parameters of movement in single neurons, the motor cortex as a whole can be understood as a dynamical system that drives the motion. An initial state is produced externally, and the system naturally relaxes while producing motor activity. This activity is then projected down the spinal cord to inter-neurons and motor-neurons (Churchland et al. [70] and Russo et al. [184]). Neural activity in the motor cortex shows a strong amplified but stable response to initial activation (Hennequin et al. [119]). There is no broad consensus on the role of the motor cortex in voluntary movement. Nevertheless, neural correlates of many different types of parameters of arm movements have been found in the motor cortex (Kalaska [134]).

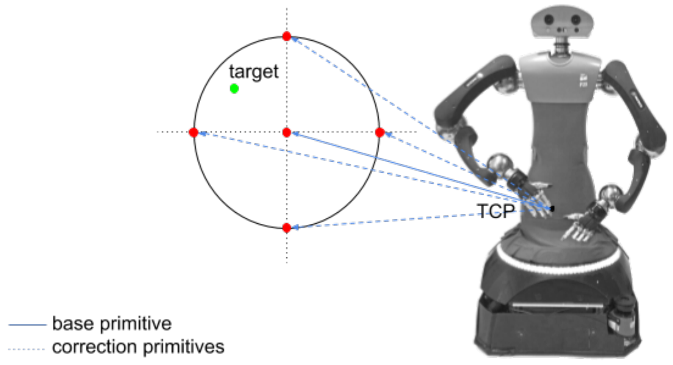


Figure 4.10.: Motion adaption problem definition. Given a fixed initial state of the robot, move the TCP to a specific target point in a plane in 3D space. A base primitive (*continuous line*) is defined to reach the center of the considered working space (*circle*), and four correction primitives (*dotted lines*) are defined to reach the boundaries.

Artificial neurons can replicate this behavior with strong recurrent connections balanced by strong inhibitory connections (Hennequin et al. [119]). Activity in the resulting network closely resembles activity in the motor cortex and can be used as an engine for complex transient motions (Hennequin et al. [119]). For example, in Ayaso [49] an architecture is proposed detailing how to generate motor commands for arm motions, which also models how learning and adaptation can be achieved by changing the gain.

In the following, a model for pre-shaping of pointing motions for a humanoid robot is presented combining motor primitives with the mechanisms for motion generation of the motor cortex (Hennequin et al. [119] and Ayaso [49]) using SNNs. The general architecture of this approach is presented in Figure 4.11. This section incorporates findings and methods from Tieck et al. [20].

A simplified model of the motor cortex is presented to generate pointing motions combining motor primitives to control a humanoid robot arm. The motion generation layer produces circular activity that creates the activation patterns for the primitives. The motor control layer has one base primitive for the pointing motion and four correction primitives that point to targets — left, right, up, and down — from the base motion target point. The target representation layer takes the target position, and based on the relative distance to the base motion target point, uses selective disinhibition to activate the correction primitives. The approach was tested with a humanoid robot using three active joints by defining different targets on a plane in 3D and control the robot to point at them. An extension of this approach, from pointing to a given target to performing a grasping or tool manipulation task, has many applications for engineering and industry involving real robots.

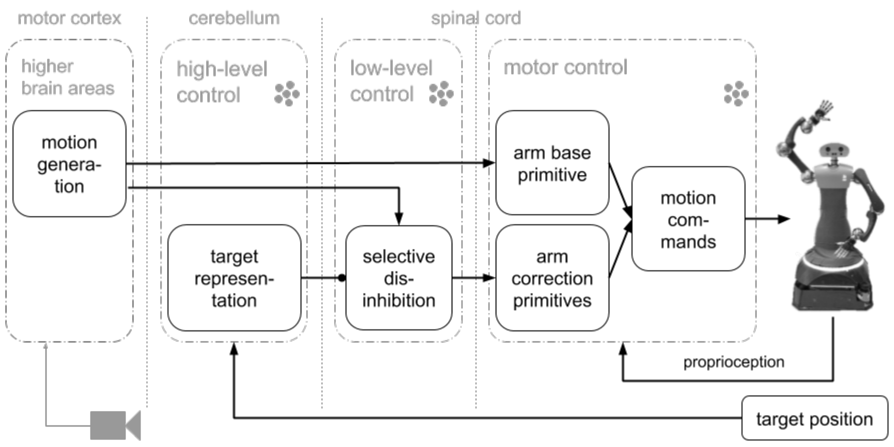


Figure 4.11.: General view of the closed-loop architecture with SNNs using predictive feed-forward control for motion adaption (Tieck et al. [20]). From right to left the layers have an increasing level of abstraction. In *motor control* a base and four correction motor primitives, in *low-level control* selective disinhibition as activation mechanism, in *high-level control* the target representation, and in *higher brain areas* a motion generation mechanism that generates the activation patterns.

4.3.1. Methods

The problem of motion adaption is formalized as follows: given an initial state of the robot and a set of primitives, move its TCP to a target point on a plane in 3D space. In classical robotics, a system calculates the IK and then validates the configuration to generate a motion trajectory. In contrast, this approach can solve this without calculating the IK and without validating the resulting configurations. The motor primitives for the arm are defined as valid possible motions in the working space. The robot arm is considered to have three active joints. The way new motions are generated is by using a base primitive combined with full or partial activations of the correction primitives. By using motor primitives to represent motions, the trajectory generation is solved in the "motor primitive space". The resulting motions are combinations of the primitives, which have only valid configurations.

A go-cue in one neuron initiates circular activity in the motor generation layer representing the motor cortex (Russo et al. [184], Kalaska [134], and Ayaso [49]). The activity of this layer is used to activate the base and correction motor primitives. Based on an error signal representing the target, the correction primitives are disinhibited and combined with the base (Richter et al. [179] and Sridharan et al. [202]). The resulting spike activation is decoded into motor commands for the robot joints. The learned weights are the distance-based inhibitory connections in the motion generation layer, the connections to the base motor primitive, and the connections to the correction primitives. The general architecture with the main components is presented in Figure 4.11. It has three main components: a motion generation layer, a motor control layer with motor primitives, and a target representation layer.

The motion generation layer produces circular activity that creates the activation patterns for the primitives. A population generates neural activity over a certain period. The first step is to normalize spike activation by changing the weights of active neurons to obtain a similar number of spikes from the whole population. Then, to obtain heterogeneity, an inhibitory population with random connections is added. In the motion generation layer MG, there is a group of two recurrent populations representing the motor cortex; one is a 2D grid MG^G and the other is an inhibitory MG^I to obtain heterogeneity. This layer generates circular neural activity over a period of time (Russo et al. [184] and Churchland et al. [70]).

The motor control layer provides the low-level motor representation based on motor primitives. There is one base motion primitive for pointing to the center and four correction primitives that point to targets left, right, up, and down from the center target point. The base primitive is activated, and, depending on the target representation signal, the correction primitives are disinhibited.

The target representation layer takes the actual target position and, based on the relative distance to the base motion, uses selective disinhibition to activate the correction primitives. The target signal is the relative position to the base primitive final position, and it is used to control the activation of the correction primitives.

Motion generation layer by adjusting local connectivity

Another mechanism to implement a motion generation layer consists of defining the connectivity of a population of spiking neurons and locally changing the synaptic weights to match the desired neural activity. In analogy to the motor cortex, it requires a relatively constant (normalized) amount of spikes being generated at a given time (see Section 2.1.1). The output of this motion generation layer is used as an activation parameter for the motor primitives.

The motion generation layer MG consists of a group of two recurrent populations of spiking neurons representing the functionality of the motor cortex. One is a 2D grid MG^G and the other is an inhibitory MG^I to obtain heterogeneity (see Figure 4.12). This layer generates circular neural activity over a period of time (Russo et al. [184] and Churchland et al. [70]).

There are two steps to initialize a motion generation layer. First, the spike activity in MG^G is stabilized, and second, the inhibitory connections from and to MG^I are added. Then, a process goes over all neurons with a "go-cue" to activate each one and record how long the activity propagates. The "go-cue" is a continuous input of spikes to one neuron for $10ms$. For each motion, the "go-neuron" is selected as the neuron that produces activity with a similar time to the desired motion.

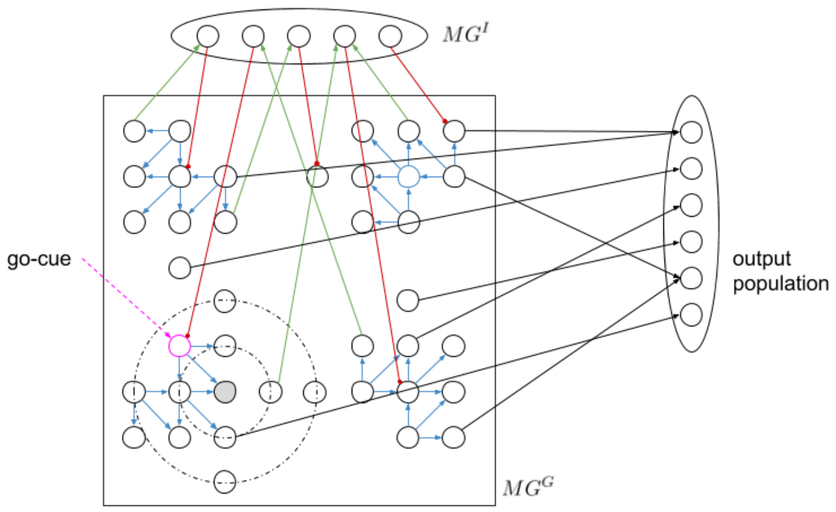


Figure 4.12.: Motion generation layer with circular activation (Tieck et al. [20]). A grid MG^G of neurons (*circles*) is connected with directed static excitatory connections depending on the quadrant. There are inhibitory plastic connections based on the distance. A hidden layer MG^I is connected with input excitatory and output inhibitory connections to disturb the regular activity of the MG^G. The generated activity from is connected to an output population.

MG^G is a square grid of 20x20 spiking neurons with recurrent connections (see Figure 4.12). There are two types of connections, the directed excitatory to generate the circular activity and the local inhibitory to stabilize the activity. The excitatory connections (blue connections in Figure 4.12) are static and have specific directed connectivity depending on the quadrant the neurons are. This amplifies the activity and forces the rotational activation. The distance based local inhibitory connections (black dotted circular lines in Figure 4.12) stabilize the activity.

To normalize the spike activity of MG^G, the inhibitory weights are increased or decreased to achieve a specific total activity MG^G_{norm} with the following learning procedure. A spike recorder is connected to all MG^G neurons. A "go-cue" (pink dotted arrow in in Figure 4.12) is given as short burst of $10ms$ of spikes into one single neuron. This initial neuron is selected randomly every time so that there are no "dark" spots in MG^G without spike activity. Every 100ms Δt (nest.sim(100ms)) the simulation is stopped. The total spikes of MG^G in Δt are counted as MG^G_{spikes}. If $MG^G_{spikes} > MG^G_{norm}$, then the weights of the inhibitory connections coming out from all active neurons are increased by Δw. Else, if $MG^G_{spikes} < MG^G_{norm}$, then the weights are decreased. Δw must be small so that a weight update does not stop the activity. This normalizes the total global activity of the MG^G population.

After training, once the propagation of circular activity of MG^G is stable, a small population MG^I is added with input and output connections with the 2D grid MG^G to obtain heterogeneity. Both input and output connections are static and random. The output connections — from MG^I to MG^G — are strong inhibitory (red connections in Figure 4.12). The input connections — from MG^G to MG^I — are excitatory (green connections in Figure 4.12). A fixed amount of input and output connections are defined to set the connections, then random neurons are sampled from both populations and are connected.

With MG^I and MG^G connected, a process goes over all neurons in MG^G to record the resulting activity. A "go-cue" is given again as a short burst of spikes for $10ms$ into each single neuron "go-neuron" (pink circle in Figure 4.12), and the duration of how long the activity propagates is measured. The time is measured until no more spikes occur or the execution is interrupted after a maximum time limit. The activity duration for each "go-neuron" is stored in a lookup table. Then, one with a similar time to the desired motion is selected to be the "go-neuron" for that primitive.

Base and correction motor primitives

The motor primitive layer MP is a layer for low level motor representation using motor primitives (see Figure 4.13). The primitives are combined to generate a specific motion based on the activation by the motion generation layer MG (see Section 4.3.1). In MP, there are populations, one for the base primitive and one for each of the correction primitives. During the execution of a motion, the base primitive is activated, and depending on the actual target representation signal (error), the correction primitives are activated respectively.

The following motion representation is proposed to generate pointing motions in the considered working space. First, a base primitive MP^B is defined (see Figure 4.13), which is a motion to point at the center of the working space. Then, four correction primitives MP^C are defined to point at points to the left, right, up and down of the center (see Figure 4.13). These four points define a circle on a plane in 3D as the boundary of the working space.

For each primitive, a different population is connected to MG. Each primitive has two motor neurons per joint in the robot. Each output spike causes a small change in the corresponding robot joint, defined as a fixed gain factor regulating the speed. A detailed view of the primitive population for the base motion is shown in Figure 4.13. The training is done one primitive at a time using an exemplary motion. Supervised learning is applied to minimize the error and adapt the weights to learn a specific motion (see Section 3.6).

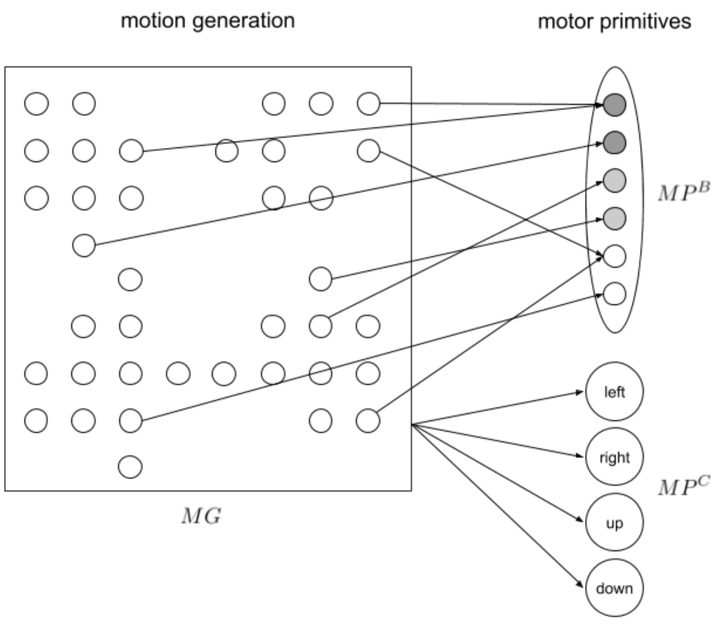


Figure 4.13.: Motor primitives layer (Tieck et al. [20]). The *small circles* represent individual neurons. The base primitive MP^B is detailed for three joints, and the same structure applies to each of the correction primitives. There are four correction primitives MP^C — left, right, up, and down. All primitives receive activation input from the motion generation layer MG. The motion generation layer is detailed in Section 4.3.1.

Target representation

The target representation layer is connected to the correction primitives with inhibitory synapses as shown in Figure 4.14. The correction primitives are inhibited by default, and they are disinhibited according to the signal provided by this layer. This mechanism is called selective disinhibition, and it is used for attention mechanisms, decisions, and mechanisms for target selection (Richter et al. [179] and Sridharan et al. [202]). For example, if no correction to the right is necessary, then the right primitive remains fully inhibited. In Kawato [135] and Wolpert et al. [223], they see the cerebellum as an internal model that can predict how the end outcome of a known motion will be. This prediction can be compared to the desired target to make the respective corrections before execution.

In this approach, a relative representation of the target is used with reference to the final position of the base primitive. This signal is applied to regulate the activation of the neurons in this layer. By decreasing the input current proportionally, this layer activates the correction primitives using selective disinhibition. This signal translates to the amount or percentage of activation, between 0 and 1, of the respective correction primitives, with 1 being total inhibition and 0 full activation. This adaptation or pre-shaping happens before executing the motion.

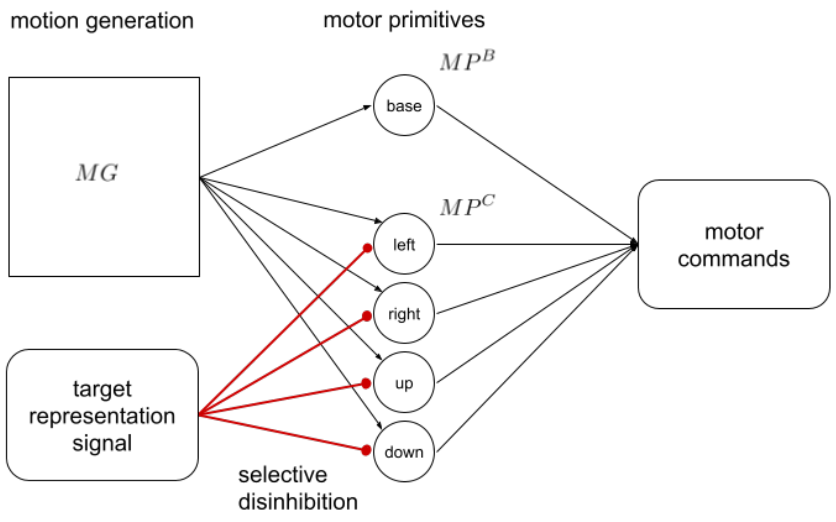


Figure 4.14.: Target representation (Tieck et al. [20]). This layer projects with strong inhibitory connections (*red*) to the correction primitives MP^C. The target is represented as a relative error signal to the target from the base primitive. This signal is used to disinhibit the primitives and adapt the base motion. The output from the base primitive and the active correction primitives are combined to generate the motor commands for the robot.

4.3.2. Experiments

In complex robotic applications, motions have to be dynamically generated according to variable targets or constraints. A major component of many robot tasks is reaching specific dynamic targets, usually followed by manipulating an object. Reaching of goal state with a robot manipulator can be understood as a pointing motion. For this reason, pointing towards different goal-points on a plane is 3D used as a benchmark task to evaluate how well the robot generates adaptive motions.

Experiment setup

Initially, the base motor primitive has to be learned. A base motion pointing towards a central target is either manually defined, generated with a motion capture system, or teach-in. The robot arm is considered to have three active joints — two in the shoulder and one in the elbow. The network is then trained to generate this specific pointing motion with all correction primitives fully inhibited. Afterward, the base motion is manually adapted towards 4 points in the boundary of the working space, each with a distance of 25 cm from the center to the left, right, top, and bottom (red points in Figure 4.15). The correction primitives are trained to produce the difference from the base motion towards these adapted motions. The network generates them when their corresponding correction primitive is disinhibited. In order to cover a different workspace, new primitives have to be defined.

This modeling allows the network to create different motions by partially inhibiting the corrections primitives and combining them. The quality of the resulting pointing motions is measured by having the SNN point at different targets in the planar working space. The reference points are used as a coordinate system, with the positive x-axis representing the inverse inhibition of the right primitive and the negative axis the left primitive, respectively. In the same way, the y-axis represents the up and down primitives. This allows a mapping from every point on the board to specific inhibitions of the correction neurons. A pointing motion is generated by setting these inhibitions manually and comparing the final position of the end-effector of the robot with the intended goal. The distance between actual and target position is used to measure the error in the following experiments.

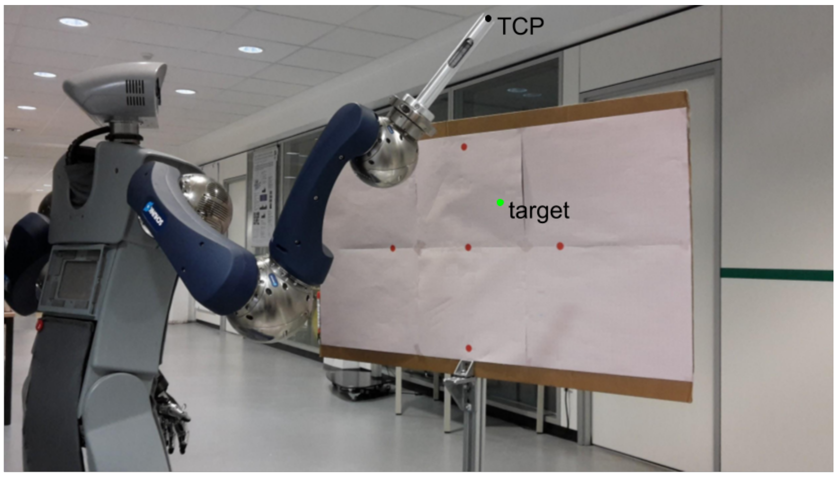


Figure 4.15.: Basic experiment setup (Tieck et al. [20]). The starting position of the robot is in front of a board representing the plane in 3D. The SNN generates a motion to reach a target point (*green dot*) with the TCP. The red dots show the end points for the base and correction primitives that are already learned.

Implementation details

The pointing motions are generated by a SNN implemented in NEST using the PyNN API. ROS is used as a communication layer to connect NEST with the robot driver. The SNN was simulated in steps of 100 milliseconds, and the spikes in this time frame were accumulated before being sent to the robot. This frequency allows the generation of smooth real-time robot movements; a complete pointing motion takes about 10 seconds. The generated spikes from the motor neurons were directly decoded into changes of joint values for the robot. The neuron activity is decoded by changing joint position by a fixed value for each spike. The resulting joint values are then used as goals for the joint trajectory controller in ROS.

During the training of MG^G, the weights of one iteration are stored in a dictionary data structure where all the required weight updates are performed. Only after all updates have been calculated, the "set weights" function in NEST is called, as constant weight changes can significantly reduce the simulation time. Thus, the total training time is reduced to about one hour on a single processor.

The network is implemented with basic LIF neurons. The layer MG is a population organized in a grid of 20x20 neurons MG^G and an inhibitory population of 20 neurons MG^I . For each of the five motor primitives MP (one base and four correction), two neurons are used per joint. The robot arm is considered to have three active joints for a total of 30 neurons. The SNN has a total of 450 neurons and about 20000 synapses.

Learning in the motion generation layer

The first evaluation was about how the learning in the SNN network work, especially in the motion generation layer. In Figure 4.16 the spike activity of all the neurons was recorded before and after learning. Without learning (*left*), the go-cue propagates in the neurons and then saturates, producing chaotic activation. After learning (*right*), the activation of the population is periodic (circular) and stable.

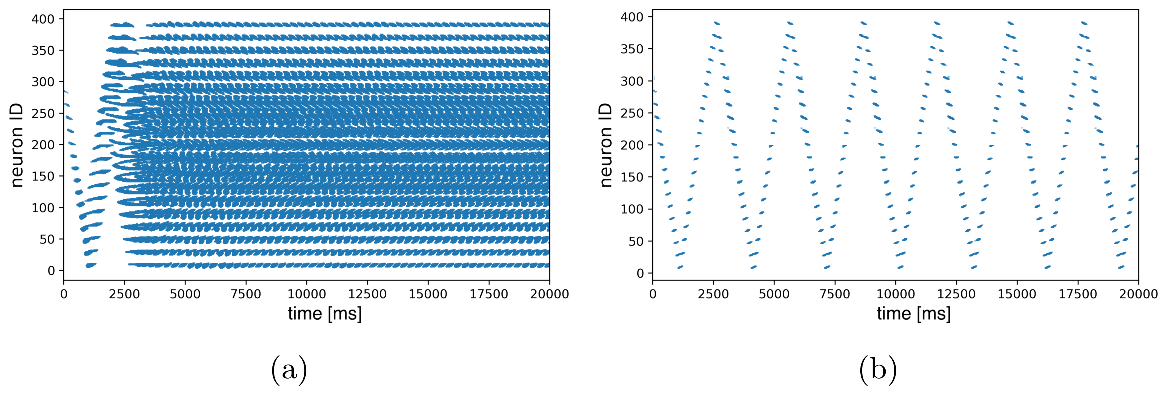


Figure 4.16.: Spike plots for the motion generation population, time is in milliseconds (Tieck et al. [20]). A: Before learning. B: After learning.

Pointing at targets on a plane in 3D space

The experiments consisted of reaching different targets on the board with the robot and evaluate the sensibility of the TCP over the working space. The distance from the

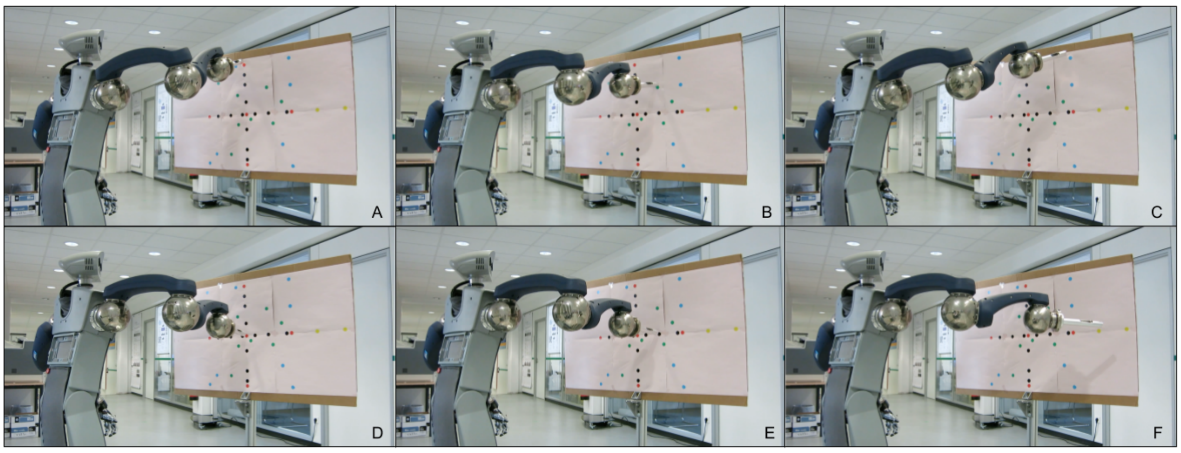


Figure 4.17.: Samples of different pointing motions (Tieck et al. [20]). The robot is pointing at different types of targets on the board (Figure 4.18A).

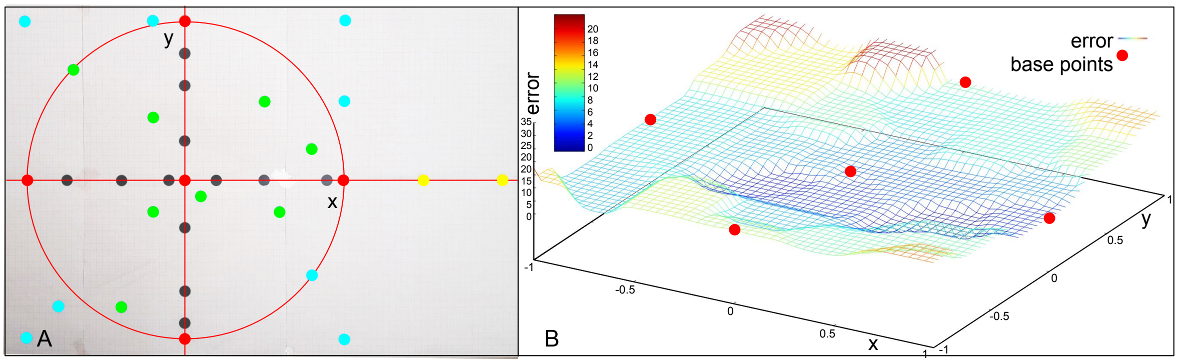


Figure 4.18.: Target points and error visualization (Tieck et al. [20]). A: Different target points are evaluated in a planar working space in 3D. B: Visualization of the error values over the working space with the error values for primitive base points (red) as reference.

target in millimeters to the final TCP position is used as an error for the evaluation. Sample images of the robot pointing at different targets are presented in Figure 4.17.

Figure 4.18 shows different targets with different colors. The target point of the base motion is the red dot in the center. The targets for the correction primitives are the red dots around the center. The black dots are obtained by activating only one correction primitive at a time. The green and blue dots are a combination of multiple correction primitives. The blue dots are outside of the working space but still in the primitive space. The yellow dots on the right are extrapolations of one primitive. A visualization of the error values over the working space in the board is shown in Figure 4.18B. The error values for primitives (red dos) are included as reference.

First, the performance of single primitives is evaluated. The red dots (in Figure 4.18) are the base points, and represent the targets for the base and the correction prim-

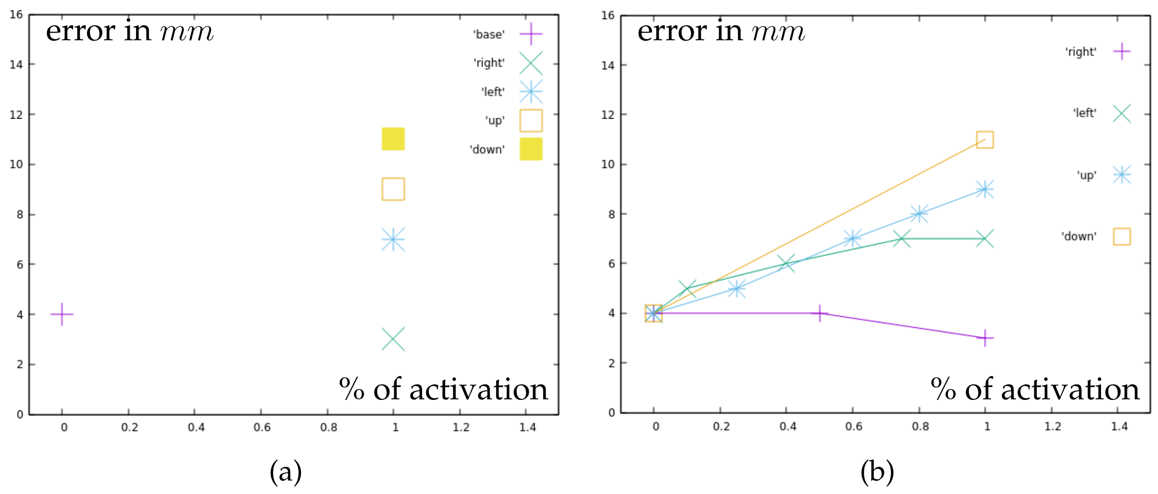


Figure 4.19.: Evaluation of single primitives (Tieck et al. [20]). (a) Error by distance to center for reaching the base points (red in Figure 4.18A). (b) Error by distance to center for different correction primitives (black in Figure 4.18A). Right is the most accurately primitive, down is the least accurate.

itives. They are reached by fully inhibiting all the correction primitives or all but one correction primitive. The plot in Figure 4.19a shows the errors for the different primitives. It can be seen that they are not reached perfectly. This results from the relatively high impact that single spikes have on the joint position. The black dots (in Figure 4.18) represent motions using only a single but partially inhibited correction primitive. The plot in Figure 4.19b shows that there is no additional error created by partially inhibiting the primitives.

Second, the performance of the combination of primitives is evaluated. The green dots (in Figure 4.18) are motions combining two correction primitives, but with a distance from the base motion not greater than one correction primitive. The plot in Figure 4.20a shows the error of the green dots. Except for one point directly on the circular test area, all motions produced a smaller error than the most inaccurate base motion. This suggests that no additional error is added through the combination of two correction primitives. The blue dots (in Figure 4.18) are motions combining two correction primitives, but with a target at a distance greater than one correction primitive. The plot in (Figure 4.20b) shows the error of the blue dots. In some cases, there is a more significant error than the combination of the errors of the primitives. The upper right point, using two fully activated primitives, has an error of 14 millimeters, while the sum of the errors of the primitives is only 12 millimeters.

Finally, the performance of the extrapolation of primitives is evaluated. The yellow dots (in Figure 4.18) are extrapolations of the right primitive. The plot in (Figure 4.20b) shows the error of the yellow dots. They are not reachable with the defined primitive, meaning that an activation of 1 (100%) is not enough to go outside of the bounds defined by the primitives (red dots). The right primitive is not only disin-

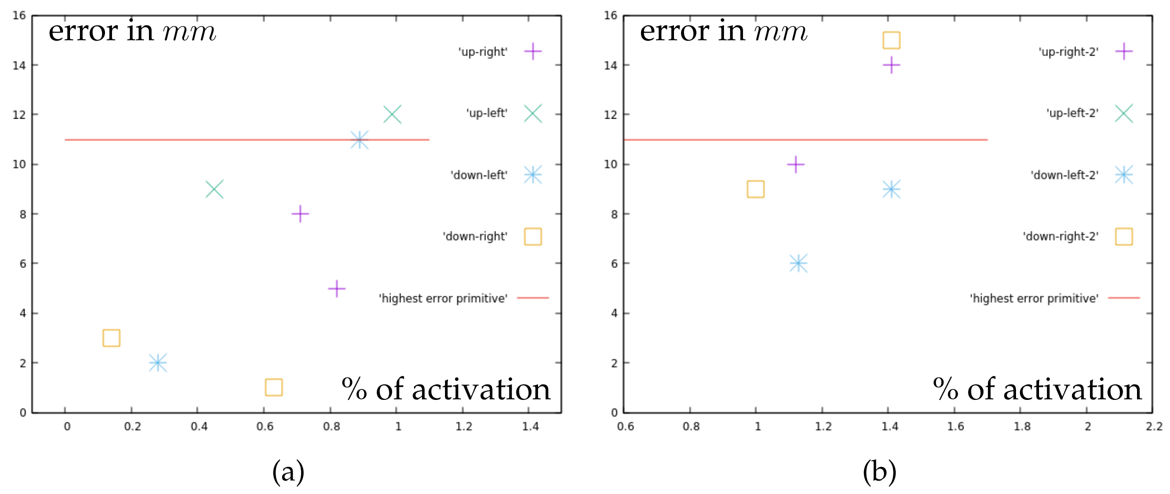


Figure 4.20.: Evaluation of primitive combinations (Tieck et al. [20]). For comparison, the error of the most inaccurate correction primitive is noted. (a) Error by distance to center for points in the different quarters (green in Figure 4.18A), they are inside the space defined by the primitives (red circle). (b) Error by distance to center for points in the different quarters (blue in Figure 4.18A), they are outside the space defined by the primitives (red circle).

hibited o accomplish the extrapolation but additional spikes are added to generate more activity. The error increases proportionally to the percentage of extra activation, which increases the extrapolation distance.

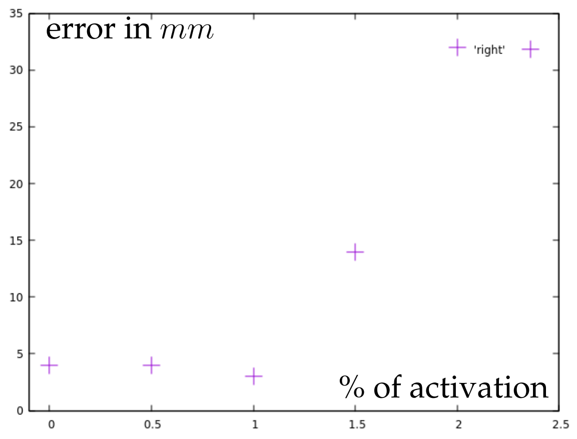


Figure 4.21.: Evaluation of primitive extrapolation (Tieck et al. [20]). Error by distance to center for motions using more than one full primitive (yellow in Figure 4.18A), these are extrapolation of the primitives. Points after 1.5% of activation show a significant increase in the error.

4.3.3. Discussion

A SNN was implemented and evaluated to generate adaptive motions using an architecture based on the motion generation mechanisms in the motor cortex. The network could pre-shape motions and generate new trajectories before its execution by combining primitives using selective disinhibition. The SNN was able to control the arm of a real humanoid robot in real-time in a predictive feed-forward scenario. The approach can be used with different robot arms and is not dependent on a specific kinematic structure.

Based on the results and the evaluation of the experiments, the following conclusions can be made. If the target distance is of one correction primitive or less (inside the circular working space), no significant error is added through adaptation. If there is a greater distance the error increases. Single spikes have a significant impact on the precision of the motions. This could be overcomed by using larger populations to increase precision and using a population encoding technique (as in Section 3.5.1) to reduce the impact of single spikes. This issue is a low-level control problem, and current work focuses on a spike-based controller for ROS to achieve smooth control.

With the recent advances in backpropagation-like learning rules for SNN, as in Kaiser et al. [5], different motion types for different tasks could be learned in the same network and start them with different go-cues. Another important aspect is integrating event-based vision to this system to get the target and drive the adaptation as in Kaiser et al. [9] or as in DeWolf et al. [91]. The association of this information with predictive motion selection yields to learning sensorimotor representations from human demonstration as in Kaiser et al. [8].

4.4. Perception driven target reaching in 3D space combining motor primitives

After experimenting with different activation modalities and the parameterization of motor primitives, the idea is to take advantage of these mechanisms to control the arm to reach targets combining the motor primitives online using perception information. Target reaching is one of the most important problems in robotics. Object interaction, manipulation, and grasping tasks require reaching a specific target (Latombe [145]).

Target reaching is defined as given an initial state of the robot arm and a target position in space, move the TCP of the arm to the target. The challenges are how to model this control problem with motor primitives and how to incorporate target information into the network. In this sense, target reaching is modeled with three correction primitives — left-right, up-down, far-near. The target is represented as a discrete negative feedback signal. An illustration of the problem is presented in Figure 4.22.

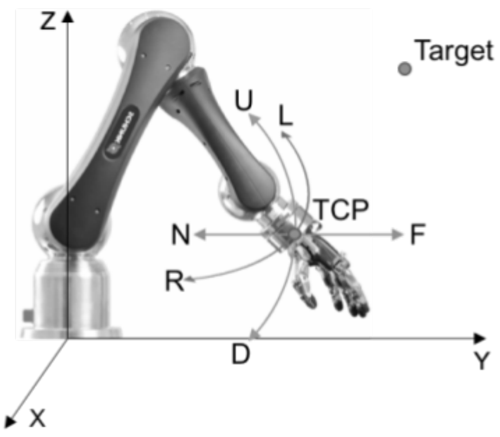


Figure 4.22.: Target reaching problem definition (Tieck et al. [23]). Given an initial state of the robot arm, move the tool center point (TCP) to a specific target point in space. Three motor primitives — $left - right$, $up - down$ and $far - near$ — are defined to move the robot TCP in different directions. They are denoted with the letters $[L, R, U, D, F, N]$ and double head arrows, respectively.

This approach is motivated by the hypothesis that human beings estimate positions and distances relative to the head. Humans can quickly determine which object is in front/back or on the left/right of another one, and which of two angles is wider (Pfeifer et al. [169]). Studies have shown that the human brain processes feedback from vision and proprioception to execute reaching movements (Saunders et al. [188] and Filimon et al. [102]). A coupling between these two systems suggests that other important components are involved in the generation of motion.

Although there have been advances in biologically inspired mechanisms for motor control, robotics still depends mainly on the classical methods. In classical model-driven robotics, the problem of reaching a target is solved by calculating the inverse kinematics (IK) for the target point, then validating the configuration, and finally planning the trajectory. These steps are computational expensive (see Section 2.2).

In the following, a model to perform target reaching with a robot arm without planning is presented, building on previous work using motor primitives for grasping and manipulation. The general architecture of this approach is presented in Figure 4.23. This section incorporates findings and methods from Tieck et al. [23, 22].

Different correction primitives are combined using an error signal to control a robot arm in a closed-loop scenario. The network can combine motions online and provide continuous control for the robot. Experiments with a simulated robot arm are presented to extensively cover the working space by reaching different target points returning to the start point, and test boundary targets and random points in sequence. The complexity of calculating the inverse kinematics and model-driven motion plan-

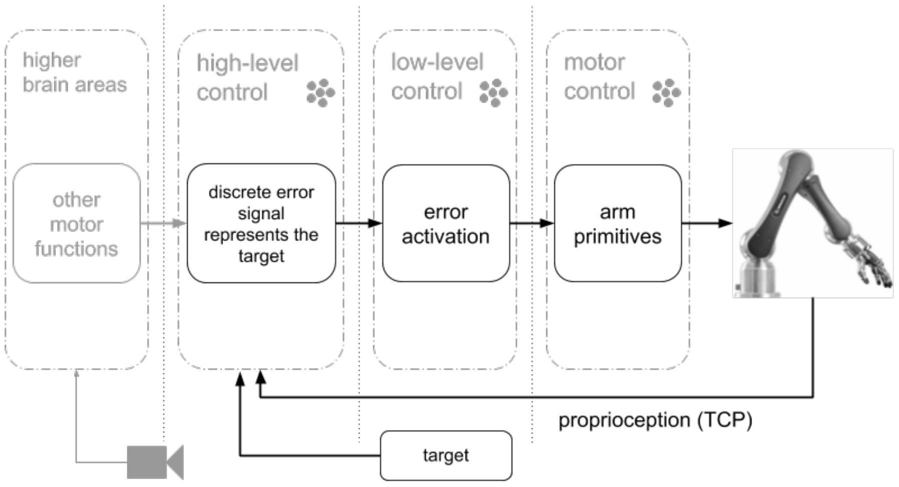


Figure 4.23.: General view of the closed-loop architecture with SNNs for perception-driven target reaching. From right to left, the layers have an increasing level of abstraction. In *motor control* there are the arm motor primitives, in *low-level control* the error activation, in *high-level control* a discrete error signal that represents the target relative to the robot, and higher brain areas and vision are included greyed out for completeness.

ning is avoided, and instead, a combination of motor primitives is used. The target is represented as a discrete error signal used as a negative signal to drive the correction primitives. Robotics applications, like target reaching, can provide benchmarking tasks and realistic scenarios for the validation of neuroscience models.

4.4.1. Methods

The problem definition is that given an initial state of the robot arm, move the tool center point (TCP) to a specific target point in space. The problem is split into three main parts. The SNN that represents the motions and controls the robot arm. The error calculation and discrete error signal that represents the target. Finally, the complete architecture connecting the error signal to drive the motor primitives.

Motion representation

Three motor primitives are defined to move the robot arm in three different directions $left - right$, $up - down$ and $far - near$, as illustrated in Figure 4.22. For a simple motion, a motor primitive represents the synergies between the active joints. The principles to represent motor primitives with SNN and to convert the spike activity to motor commands were introduced in Sections 3.2 and 3.5.1.

Target representation

Humans do not calculate a precise model and the error for moving the arm around. Instead, the brain perceives a predictive decision or signal on how the motion should be (Filimon et al. [102] and Saunders et al. [188]). Humans have multiple control loops that overlap and change activation depending on the current action and the sensory feedback. The human sensori-motor system is highly adaptive in terms of parameter variance. Thus, a precise target and error representation is not required. The target is represented as an error signal in a discrete scale to determine the direction of motion. An error ξ between the current TCP position and the target is calculated in polar coordinates $[\varphi, \theta, r]$ (see Figure 4.24) as

$$\xi_\varphi = \varphi_B - \varphi_A, \tag{4.1}$$

$$\xi_\theta = \theta_B - \theta_A, \tag{4.2}$$

$$\xi_r = r_B - r_A. \tag{4.3}$$

The error signal ϵ is generated based on the error ξ. It is defined as $\{\epsilon_\alpha, \epsilon_\beta, \epsilon_r\} \in [-1, -0.5, 0, 0.5, 1]$. In other words, it is a signal to indicate the direction and distance — if the target is far away $[1, -1]$, close $[0.5, -0.5]$, or on target $[0]$. This error signal represents the visual feedback about the target.

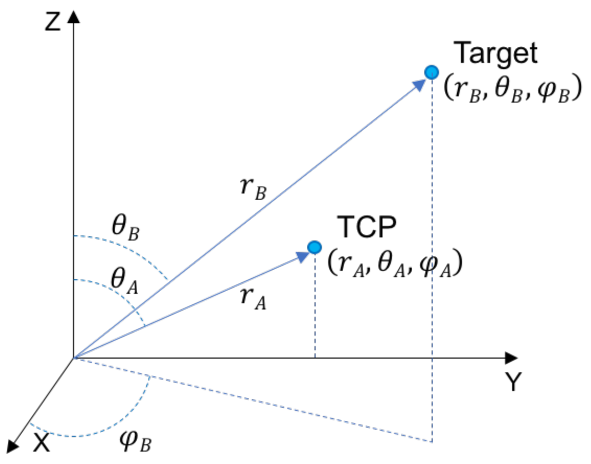


Figure 4.24.: Target representation as an error signal (Tieck et al. [23]). To represent the target as a discrete error signal, the distance between the TCP and the target is determined in polar coordinates (φ, θ, r). The sub-indexes $_A$ and $_B$ represent the TCP and the target, respectively. The error in each component is discretised with values $[-1, -0.5, 0, 0.5, 1]$.

Connect primitives with error signal

Three primitives are defined, one for each direction. The different primitives are combined using the error signal. The motion representation with SNN and a hierarchical architecture is used as presented in Chapter 3. The error signal is encoded into spike trains by three different input populations. Instead of planing a trajectory, the error components are used to activate and drive the different primitives. The primitive $left - right$ is connected to ϵ_α, $up - down$ to ϵ_β, and ϵ_r to $far - near$. The complete SNN architecture in closed-loop is presented in Figure 4.25.

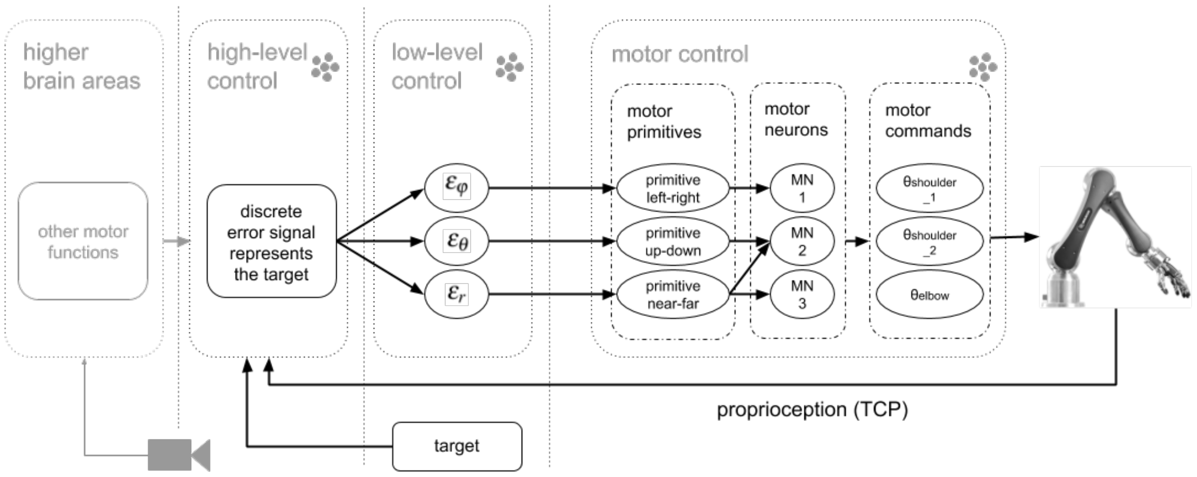


Figure 4.25.: Detailed view of the SNN in closed-loop showing the motor control hierarchy (Tieck et al. [23]). The *motor neurons* generate *motor commands*. The *motor primitives* — $left-right$, $up-down$ and $far-near$ — are activated by the *discrete error signals*. The error signals are calculated using the TCP proprioception and the target representation in polar coordinates. Higher brain areas and vision are included greyed out for completeness.

To illustrate how the system works a sample run is shown in Figure 4.26. From top to bottom, the TCP positions, the error encoded in spikes and the spike activation of the output population for the primitives.

4.4.2. Experiments

The experiment setup to evaluate the approach is shown in Figure 4.27. A simulated robot arm is fixed on a base. For the experiments, the robot has three active degrees of freedom — two in the shoulder and one in the elbow. A target is always a point in space, and the center of a sphere represents it. The radius of the sphere represents the threshold for the desired precision/error while reaching the target. The simulation provides the position of the TCP of the robot arm and the center of the sphere. The SNN has to control the arm to reach the target with the TCP. A motion is considered successful if the TCP is stable positioned inside the sphere. Figure 4.27 presents an illustration of different targets and the robot motion.

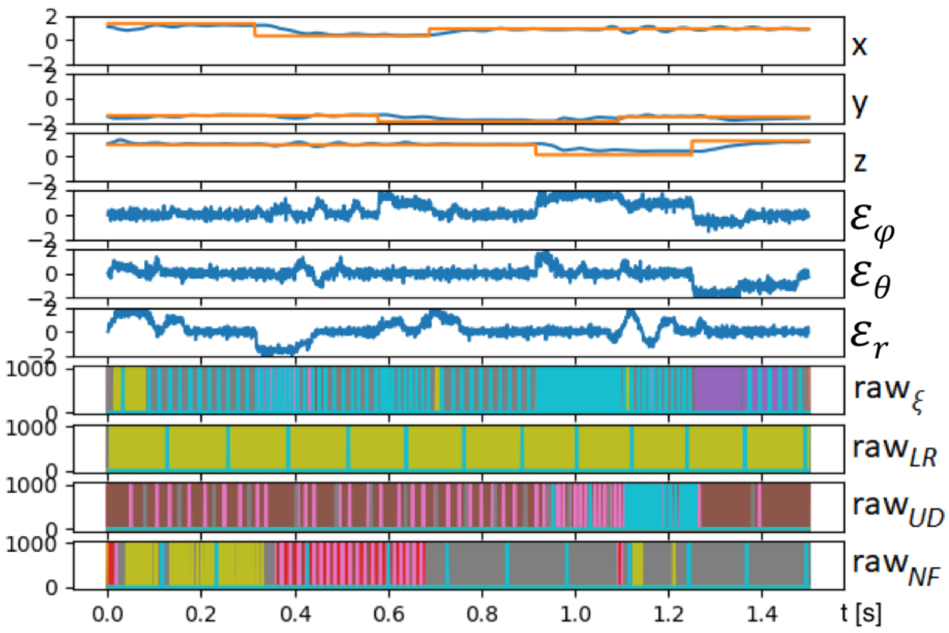


Figure 4.26.: Spike activation, error signals and TCP position (Tieck et al. [23]). To illustrate how the system works, a sample run of the system is presented. The first three rows show the current TCP position x, y, and z in blue vs. the target's location in orange. The next thee rows show the spike activation of the error signals ϵ_φ, ϵ_θ and ϵ_r. In row seven raw_ϵ shows the spike pattern of the error-related population. The spike patterns of the neuron populations, representing the motor primitives, are respectively plotted by raw_{LR}, raw_{UD}, and raw_{NF}.

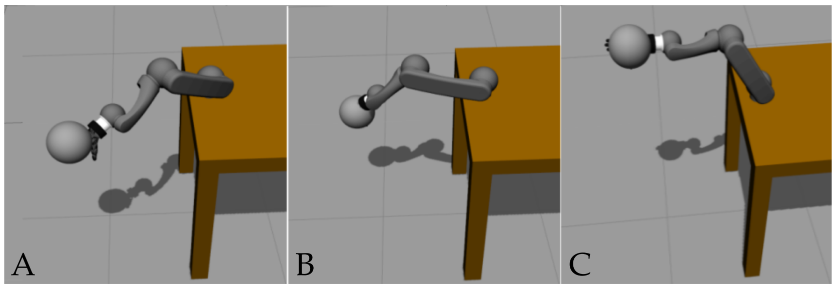


Figure 4.27.: Experiment setup. A frame sequence of one of the experiments (Tieck et al. [23]). The images show the robotic arm in simulation following a ball in space, controlled by the SNN. The robot arm is fixed on a table. It has three active joints. The center of the sphere represents the target point, and the radius represents the threshold for the allowed precision.

Implementation details

ROS is used as a communication framework to integrate the neurosimulator, the physics simulator, the robot controller, and the experiment scripts. A robot model of a Schunk LWA 4P robotic arm with three active joints is used for the experiments. The environment and the robot were simulated in Gazebo.

The SNN is implemented in Nengo. For each joint, an output population of 200 spiking neurons is used. All neurons are LIF. There are three primitives — $left - right$, $up - down$ and $far - near$ — implemented as sub-networks of about 1000 neurons. The primitives are trained offline. For the inputs, populations of 200 neurons encode the signals for the TCP proprioception, and the target is represented as a discrete error signal. The implemented SNN is presented in Figure 4.28

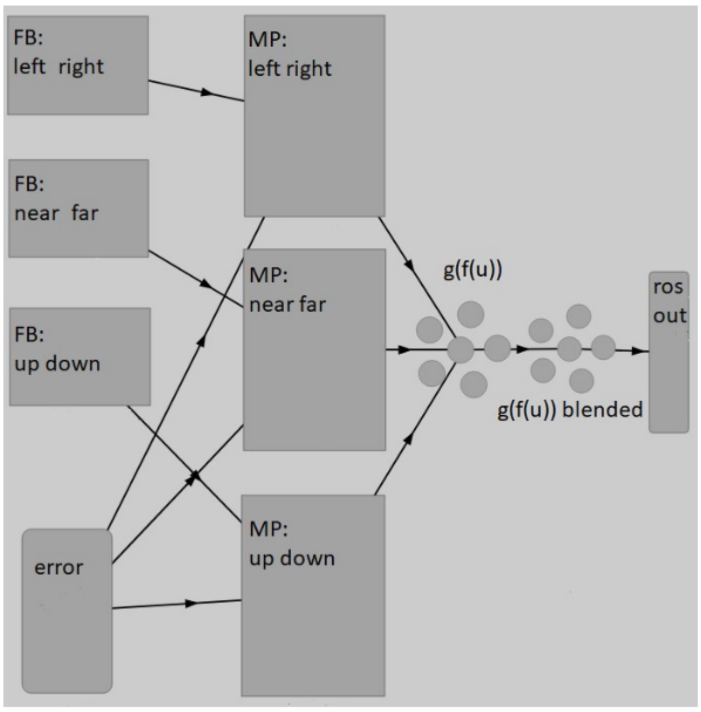


Figure 4.28.: SNN implemented in Nengo for target reaching driven by the error signal (Tieck et al. [23]). The three sub-networks *left* represent the proprioception of the TCP. The node *lower left* represents the target error signal. There are three sub-networks, each responsible for one motor primitive. Two ensembles, $g(f(u))$ and $g(f(u))blended$ generate motor control output for the robot simulation, via the "ros out" node.

Cover the working space returning to the start

An interesting aspect is the activation of motion primitives to move the TCP to a target point in space and then return to the starting position. This task can be executed in two ways: in systematic sequence and random.

First, different targets are selected in sequence by systematically changing in fixed increments one of the coordinates at a time. This is repeated until the working space is covered, see Figure 4.30a. In total, there were 6426 target points. The intervals for each primitive and the step size for the sampling are summarized in table in Figure 4.29.

Primitive	**Min**	**Max**	**Step size**
$left - right\ (\varphi)$	3.0	6.3	0.1
$up - down\ (\theta)$	0.5	2.5	0.1
$far - near\ (r)$	0.2	0.6	0.05

Figure 4.29.: Table with primitive parameters (Tieck et al. [23]). Each "Primitive" has a minimum "Min" and maximum "Max" value to move the TCP with respect to the origin shown in Figure 4.22. To cover the working space of the robot arm, the intervals were sampled with "Step size".

The SNN was able to reach all points. The threshold used was $0.1m$. See that the error is lower than the threshold on average, which indicates a successful motion. This is an exhaustive evaluation of the approach in the working space.

After that, random target points are selected within the working space, see Figure 4.30b. A set of random points are selected from the point cloud in Figure 4.30a. Notice that for most of the points, the robot reached the target, but for some points, there are oscillations. After reaching a target, the robot returns to the initial position so that all trajectories start from there. These two experiments show that the network can drive a robot arm to any point in the spherical sector representing the working space. This shows that the method is robust and does not depend on previous motions.

Reach boundary targets and random points in sequence

For the evaluation of the network, target points are defined in the boundary points that define the working space as shown in Figure 4.31a. The robot has to move sequentially through all the points. Random points are selected within the working space. The movements are in sequence without returning to the base point. The process can be observed in Figure 4.31b. This shows the reaching of different points in an arbitrary order and that the error does not accumulate.

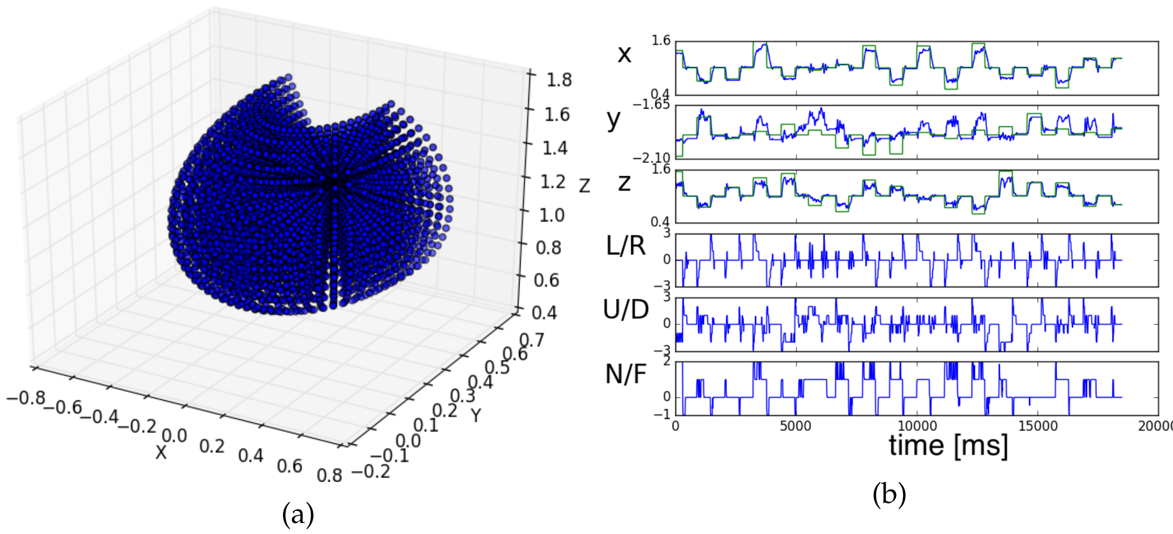


Figure 4.30.: Experiments: covering the working space returning to start (Tieck et al. [23]). (a) Point cloud of targets to sample the working space of the robot arm according to the parameter table in Figure 4.29. (b) Performance plot and primitive activation. In x, y and z the TCP position in blue and the target position in green. The last three rows show the activation of the primitives — "L/R", "U/D", and "N/F" — based on the error signal.

4.4.3. Discussion

Using a combination of motor primitives and a discrete error signal, the network was able to perform online target reaching without model-driven planning and avoiding the complexity of calculating the IK. Motions can be modeled in a parametric way with motor primitives simplifying the motor control and allowing combination with simple activation signals. This approach can also be used with different robot arms by just redefining the mapping of the primitives to the robot kinematic. This approach uses the principles for motion representation using motor primitives and SNN and extends the motion architecture presented in Chapter 3 for online target reaching in closed-loop scenarios.

Benchmark experiments in simulation with a robot arm were presented to extensively cover the working space by reaching different points and returning to the start point, and test boundary targets and random points in sequence. As mentioned in Section 4.2.2, there are oscillations in some cases while reaching the target point. This is caused because the raw output spike activity is used to decode motor commands, and it is noisy. Whereas in the human body, there are muscles that integrate the activation signals to smooth the motions.

In this experiment, the TCP position and the position of the target are obtained from the simulation. However, reaching involves visual feedback for online motion (Filimon et al. [102] and Saunders et al. [188]). In order to test in a real robot, perception

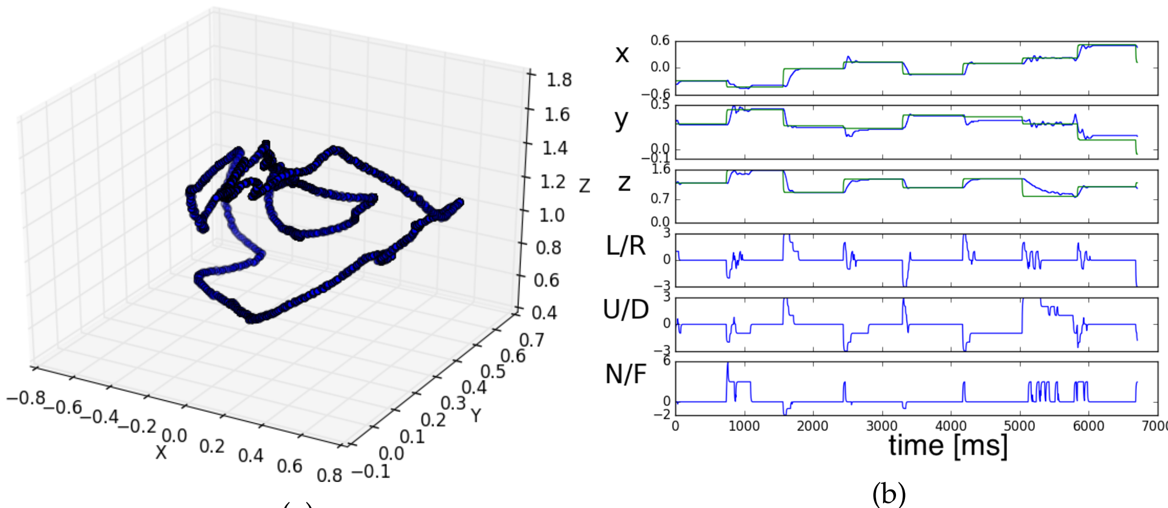


Figure 4.31.: Experiments: limits and random targets in sequence (Tieck et al. [23]). Random targets are generated within the working space of the robot. After reaching, a new target is set without returning to the start position. (a) Trajectory of the TCP position in 3D space. (b) Performance plot and primitive activation. In x, y and z the TCP position in blue and the target position in green. The last three rows show the primitive activation — "L/R", "U/D", and "N/F" — based on the error signal.

has to be integrated with an event-based camera to get TCP and target "relative positions" or better, just an indication of the direction and the magnitude of the error as $[-1, -0.5, 0, 0.5, 1]$. The vision system proposed in Kaiser et al. [9] could be integrated to perform motion prediction and determine the error signal (Kaiser et al. [133] and Probst et al. [175]). Event-driven systems being controlled in the robot joint space using motor primitives can take advantage of the unique characteristics of SNNs.

4.5. Summary

The modeling of the arm motion as a combination of motor primitives simplifies the control. Low-level primitives represent the motion of each joint, two in the shoulder and one in the elbow. There is a reduction of the control parameters, allowing different activations and combinations of the primitives. It is possible to combine the primitives offline before the motion or online during the motion to generate new adaptive motion trajectories. Three benchmarking experiments were presented to model and control the motion of a robot arm.

The first one, multimodal activation, presented a framework for motor control that allows activation, combination, and parameterization of motor primitives. This method shows interesting properties of biological systems. On the one hand, the representation of the motions is hierarchical, going from simple joint movements to complex arm motions and integrating signals from other higher brain areas. On the other hand, motions can be activated by different modalities. Motions are represented with SNN using motor primitives as fundamental building blocks (see Section 3.2). This representation allows the combination and parameterization of the motions. A mechanism using the efforts in the joints and the proprioception was integrated to detect contact and trigger a retract reflex.

The second one, motion adaption, presented a network for adaptive motions using an architecture based on the principles of motion generation in the central nervous system. The network could pre-shape motions and generate new trajectories before its execution by combining primitives using selective disinhibition. The SNN was able to control a real humanoid robot in real-time in a closed-loop scenario. This approach can be used with different robot arms and is not dependent on a specific kinematic structure.

The third one, target reaching, presented how a combination of motor primitives and a discrete error signal, the network could perform online target reaching without planning and avoiding the complexity of calculating the IK. The experiments showed that motor control could be simplified by using motor primitives for a robot arm. Especially for target reaching, it has been shown that they can reduce the number of controlled parameters and the amount of information to process (Yang et al. [226]). Motions can be modeled in a parametric way with motor primitives and combined with simple activation signals. By using motor primitives, this approach can also be used with different robot arms by just redefining the mapping of the primitives to the robot kinematic. This experiment extends the architecture for multimodal activation presented in Section 4.2 for online target reaching in closed-loop scenarios.

5. Coordination of motor primitives and compliant control for anthropomorphic robotic hands

Humans learn grasping motions and adapt them during execution based on the shape and the intended interaction with objects. There are studies on human motor control providing insights into the mechanisms involved in grasping. For example, there is work on the evidence of muscle synergies for grasping (Sburlea et al. [189]), the relation between human responses and the stiffness regulation in the hand (Crago et al. [75]), and the generalization of muscle patterns as building blocks for grasping (Scano et al. [190]). Besides, the work in Cutkosky [78] shows that only a small grasp repertoire is used. Furthermore, a principal component analysis revealed that the first two components determine 80% of the variance of all grasps (Santello et al. [186]). An anthropomorphic robotic hand offers many possibilities to investigate further the neural response of grasping motions (Kim et al. [137]), to evaluate different affordances (Ruehl et al. [183]), and to use the synergies from human demonstration for grasping control (Ficuciello et al. [101]).

In the following, a framework is presented to perform different grasping motions with a robotic hand with SNN based motor primitives. The hand modeling is made considering single finger motions independent of other fingers in analogy to the motions of an arm. Additionally, another layer of primitives is added on top to coordinate the fingers for different grasp motions. In the following, three scenarios are considered: grasping motions, triggering finger reflexes, and soft-grasping.

Grasping motions are modeled with motor primitives on different layers in a hierarchy of joints, controlled by the fingers and coordinated by the hand. Affordances are motions that humans perform to grasp different objects — pinch and cylinder, for example, (Cutkosky [78]). The challenges to generate grasping motions are to develop a suitable SNN network architecture to represent grasping motions and an appropriate learning mechanism.

It is possible to enhance the grasping behavior making it more adaptive by adding finger reflexes to the hand control. Additionally, by incorporating human signals in real-time, the training and programming of the robot are more flexible. A reflex is an involuntary and direct response to sensor stimulation and can be either a complete execution or inhibition of a motion (Byrne et al. [64]). The challenge is how to apply this principle to execute finger primitives fully.

Soft-grasping is modeled with two control loops combined. One is based on motor primitives, and the other is a compliant controller activated as a reflex when contact is detected. The human brain is driven by events and goal-oriented behavior with no explicit planer for grasping. From the view of neuroscience and bio-cybernetics, there is a combination of multiple control loops working together to grasp an object. Based on the sensor feedback, humans can quickly adapt the hand motion if the object moves or deforms. This process is called soft-grasping (Caldwell et al. [65] and Bonilla et al. [56]). The challenges are how to detect contact with the object and how to adapt the motions accordingly.

5.1. Modeling the motion of a 5-finger robot hand

The experiments for demonstrating grasping with SNNs are performed with a 5-finger robotic hand. The control interface is based on ROS to communicate the SNNs with both the simulation and the real robot. The hand has 9 active DoF as illustrated in Figure 5.1 — 7 in the fingers, finger spread and thumb opposition.

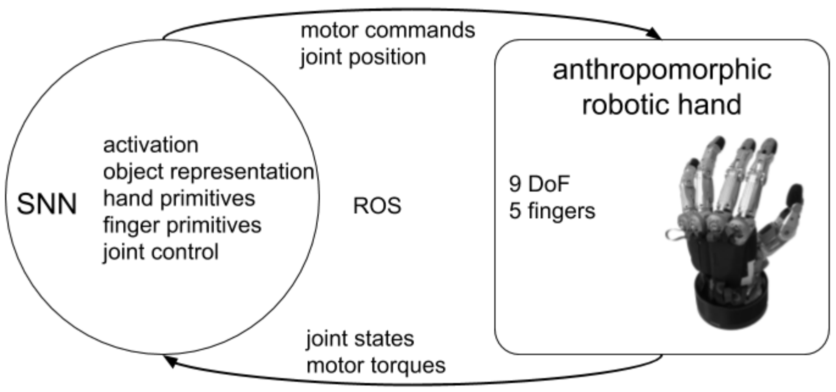
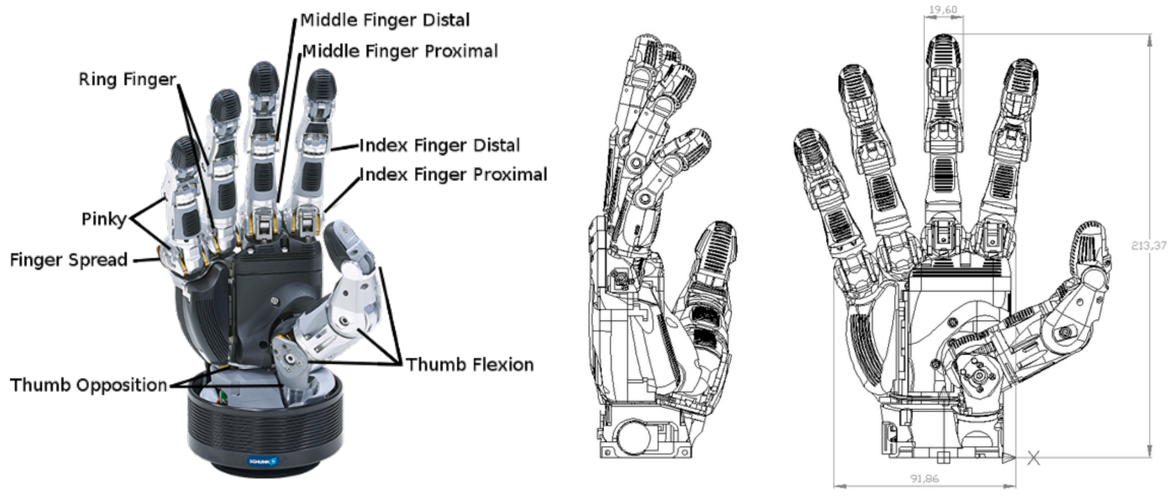


Figure 5.1.: Hand motion modeling. The robotic hand is controlled with the joint position interface. It has five fingers with nine active DoF, and it has no force or contact sensors. Primitives are defined for each finger and for the hand to coordinate the fingers. A grasp type signal is used to activate specific object grasps.

In order to control the hand with SNN, there is an output population for each of the joints. The individual fingers are modeled as open kinematic chains, similar to the modeling of the robot arm (Section 4.1). There are motion primitives for each finger that represent the synergies between the joints involved during the motion. The base finger motions are represented by sampling the joint states of one finger while closing it. Then the primitive sub-networks are trained to represent the sequences. In this way, the motion of each finger is independent of all others. On top of that, there is a layer that represents the hand to coordinate the fingers. The hand network is also a motion primitive that controls the activation of the finger primitives instead of controlling joints. There is a contact detection mechanism to trigger finger reflexes and to activate compliant control.

For the experiments an anthropomorphic 5-finger hand Schunk SVH is used (see Figure 5.2). The hand is developed for tasks such as human-like grasping, human-robot collaboration, and service robotics (Schunk [194]). The electronics are integrated into the wrist. Due to its characteristics, the hand targets a different level of manipulation and grasping tasks. To control the robot hand, the official ROS SVH driver is used (Heppner [121]). The robotic hand has 9 active DoF as shown in Figure 5.2a.



(a)

(b)

Figure 5.2.: Robotic 5-finger hand Schunk SVH (Heppner [121]). (a) Joint description. (b) Schematics and dimensions.

5.2. Learning grasping motions from human demonstration

A hierarchical control model is proposed to control a robot hand with motor primitives using human motion data as a demonstration. It consists of a layer representing the motion of individual fingers and a higher-level layer to represent different grasping motions and coordinate the fingers. The human hand is a complex system that can perform a wide range of motions with great flexibility and adaptation, for example, playing piano or grasping unknown objects. Humans can remember grasp motions and modify them during execution based on the shape and the interaction with objects.

Grasping motions are defined as affordances that humans can perform to grasp different objects. To model grasping, the challenge is to find a proper network architecture to represent grasping motions and a learning mechanism to train them. In this sense, grasping motions are modeled with motor primitives in different layers in a joint hierarchy, controlled by the fingers and coordinated by the hand. An illustration of the problem is presented in Figure 5.3.

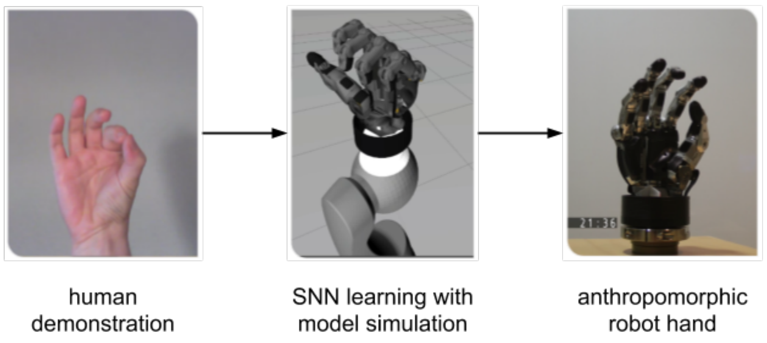

human demonstration
SNN learning with model simulation
anthropomorphic robot hand

Figure 5.3.: Grasping motions problem definition. Given a set of human demonstrations for grasping motions, learn a SNN model based on motor primitives to control a robot hand in simulation. Then use the SNN to control a real anthropomorphic robot hand.

The way movement is represented and executed in biology is an active field of research, especially concerning hand movements. However, studies show that only a small grasp repertoire is used (Cutkosky [78]). Furthermore, a principal component analysis of a set of human grasps revealed that the first two components determine 80% of the variance of all grasps (Santello et al. [186]).

In SNNs, plasticity is used for learning by changing the synaptic weights. There are approaches using STDP as a learning mechanism in a neurorobotics context. For instance, to learn transformations of spatio-temporal data between coordinate systems (Davison et al. [87] and Song et al. [201]). Approaches for learning robot kinematics in simulation (Srinivasa et al. [203]) and with a real robotic arm (Bouganis et al. [57]) have been proposed.

In the following, a hierarchical SNN is presented to learn and perform different grasping motions. The SNN combines two different network types, one for the fingers and one for the hand. The general architecture of this approach is presented in Figure 5.4. This section incorporates findings and methods from Tieck et al. [16].

The finger networks learn different motor primitives as the synergies between the joints. The hand network represents different grasping affordances coordinating the

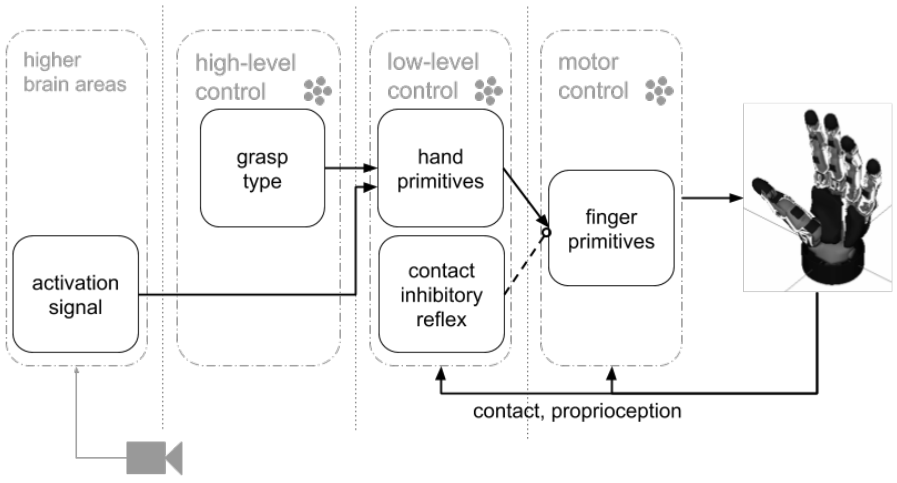


Figure 5.4.: General view of the closed-loop architecture with SNNs for grasping motions. From right to left the layers have an increasing level of abstraction. In *motor control* there are the finger primitives, in *low-level control* the hand primitives and the inhibitory reflex, in *high-level control* the grasp type signal, and in *higher brain areas* the activation signal.

finger networks, thus reusing the learned motor primitives. Both the hand and the finger networks are trained independently using STDP. After learning from human demonstration, the SNN is evaluated in simulation and with a real anthropomorphic robot hand. The network exposes the ability to learn finger coordination and synergies between joints to grasp real objects.

5.2.1. Methods

This SNN approach is inspired by the biological concepts of hierarchical motion representation (Bizzi et al. [55]) and motor primitives (Bernstein [54]) for grasping using muscle synergies (d'Avella et al. [84]). The following assumptions are made for the fingers and the hand. The hand makes different types of grasp motions when picking different objects, such as picking a pen from a table (pinch) or holding a tennis racket (cylinder). In these examples, the motion of a single finger is defined by a motor primitive that represents its joint synergies during the motion.

The motion representation and control are modeled using two types of networks, one for the fingers and one for the hand (see Figure 5.5). The finger networks control the movements of single fingers independent of the task, while the hand network coordinates the activation of the finger networks for a specific grasp motion. The network receives as proprioception (sensor data) the joint positions of the robot. Grasping motions are recorded from human demonstration to train the SNN.

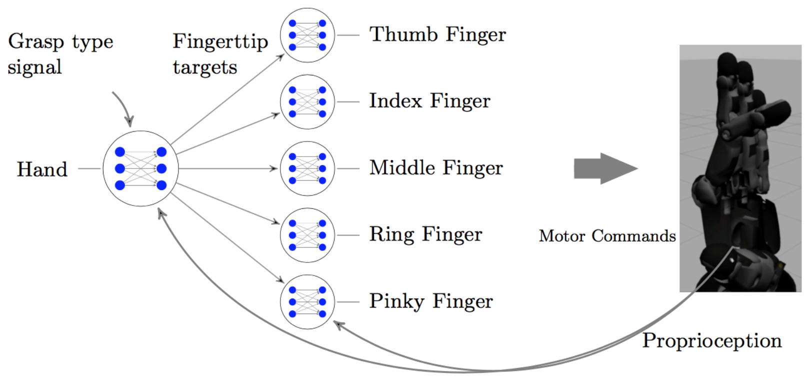


Figure 5.5.: General view of the SNN (Tieck et al. [16]). The hand network (*left*) receives the proprioception (joint positions) of all fingers and a grasp type signal to generate fingertip targets. The hand network coordinates the finger networks. Each finger network (*middle)* receives its proprioception (joint positions) and a fingertip target to generate motor commands.

Finger networks

A single finger motion is abstracted to be planar (2D) and performed by two joints, proximal and distal. A finger network (see Figure 5.6a) learns the synergies between the joints to represent a motor primitive. The network has an input and an output layer connected all-to-all with plastic synapses that are learned with associative learning (see Section 3.6.2). The input layer is divided into four populations. Two encode the joint angles θ_1 and θ_2 from the finger proprioception (joint positions), and two represent the fingertip target as a normalized direction vector of the spatial changes with $\Delta\bar{d}_1$ and $\Delta\bar{d}_2$ in finger tip coordinates (Cartesian). The output layer is divided into two populations $\Delta\theta_1$ and $\Delta\theta_2$, for the angular changes of each joint.

Hand network

The hand network (see Figure 5.6b) represents the different grasp types as the coordination of the individual finger networks and thus, reusing the learned primitives. The network learns to associate the proprioception of all fingers and the grasp type with the corresponding fingertip targets. A signal coming from higher brain areas is introduced to determine the grasp type. This signal could be generated in another network to represent grasp affordances from vision (Kaiser et al. [11]). As in the finger networks, a simple two-layer architecture was not suited to learn multiple grasping motions due to the number of input and output populations with high correlations. This caused the creation of local attractors by the STDP learning rule. A higher dimensional hidden layer was added to reduce correlations and get a sparse representation of the input to overcome this problem. Each input population is projected with static synapses to the hidden layer with a small out-degree creating only

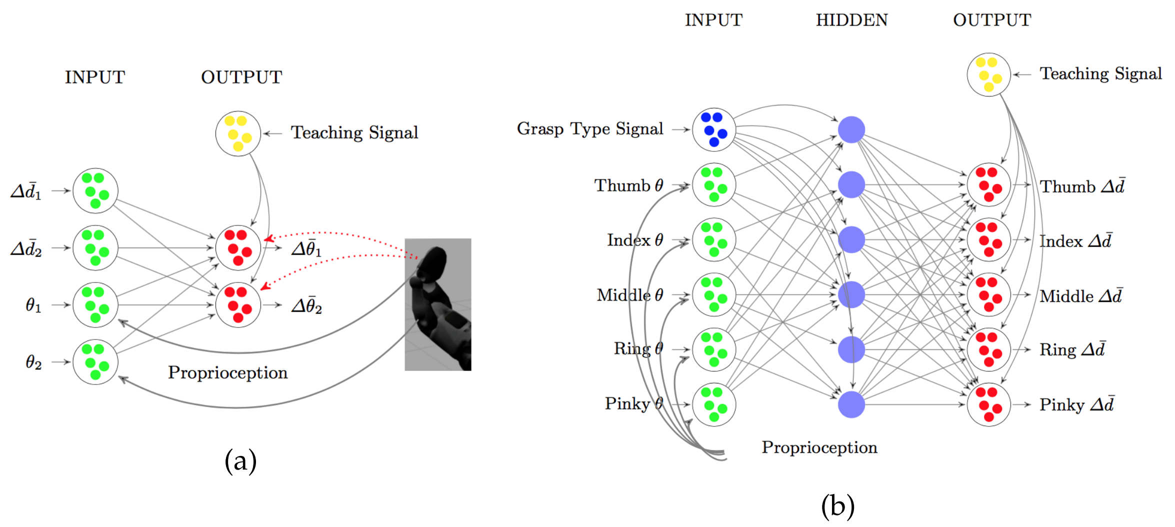


Figure 5.6.: Detailed view of the SNN sub-networks (Tieck et al. [16]). Input and output layers are divided into populations. For learning, a teaching signal is connected to the output layer. (a) The Finger networks associate the proprioception (θ_1, θ_2) and the fingertip target $(\Delta\bar{d}_1, \Delta\bar{d}_2)$, with the joint changes $(\Delta\bar{\theta}_1, \Delta\bar{\theta}_2)$. Each finger network receives tactile feedback (*dotted*). (b) The hand network generates the fingertip targets $\Delta\bar{d}$ to coordinate the finger networks. The inputs, proprioception θ and the grasp type, are sparsely projected to the hidden layer (*middle*).

a few static synapses between the input and the hidden layers. The hidden and the output layer are connected all-to-all with plastic synapses that are learned with associative learning (see Section 3.6.2).

Tactile feedback

A signal to represent tactile feedback is added to stop the motion when there is contact with an object. Each finger network has a single tactile neuron connected with very strong inhibitory synapses to the output layer (see Figure 5.6a). The firing frequency of this neuron is higher than the maximal activation frequency of the output neurons. Once active, it will maintain the membrane potential of the output neurons below the threshold to prevent them from firing and stop the motion.

5.2.2. Experiments

The capabilities of the SNN are evaluated with two different grasp motion types — pinch and cylinder. The experiment setup is presented in Figure 5.7. The control schema is shown in Figure 5.7a. The SNN was implemented with NEST 2.10. The world simulation is made with the Gazebo physics simulator using a model of the robot hand. ROS is used as middleware for modular design and inter-component communication. To control the hand, the *schunk_svh_driver* (Heppner [121]) is used. This architecture allows the transition from simulation to the real robot. A motion capture system (*LeapMotion*) is used to record training data from human demonstration using a 3D visualization tool (Figure 5.7b). The SNN is evaluated in simulation and on the real robot hand (Figure 5.7c).

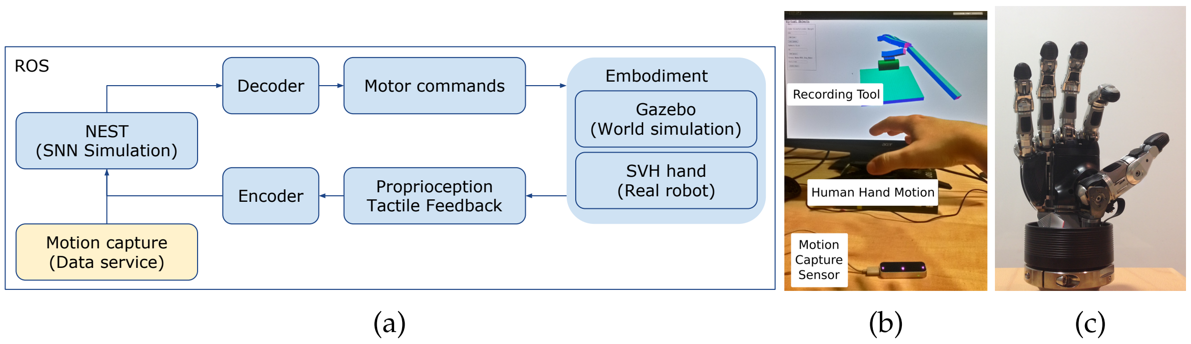
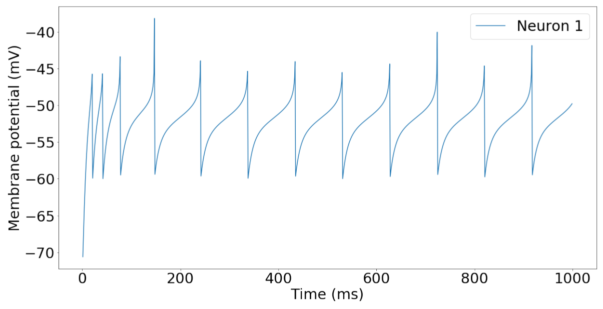


Figure 5.7.: Experiment setup (Tieck et al. [16]). (a) Control schema. The communication between the components is implemented with ROS. The spike encoder and decoder are described in Section 3.5.2. (b) Motion capture system to record human hand demonstrations. (c) 5-finger Schunk SVH robot hand.

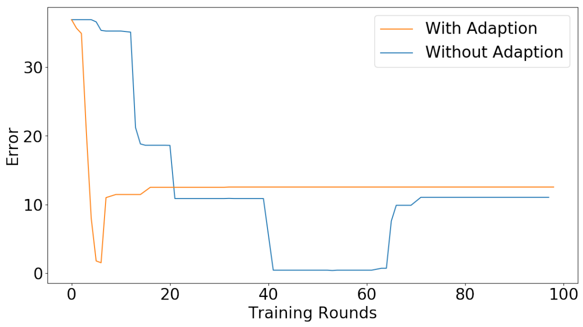
Implementation details

The SNN is implemented with a total of 5100 neurons. Each input or output population has 100 neurons. Each finger network has 7 input and output populations (see Figure 5.6a), for a total of 700 neurons. The hand network has 12 input and output populations (see Figure 5.6b), and a hidden layer that is larger than all the input layers together, with 1400 neurons, for of a total of 2600 neurons. The Leaky-Integrate-and-Fire (LIF) neuron model is used with alpha-function shaped post-synaptic currents. For the grasp type input population, a variant of LIF with spike-time adaptive neuron model is used to stabilize the constant spiking activity. The mean squared error (MSE) of the SNN output with and without adaptive neurons in the grasp type population is shown in Figure 5.8.

A data service selects the samples from human demonstration to train the SNN. Two types of grasps were recorded, cylinder and pinch. Each grasp type has 2 different examples. The sensor operates at $115Hz$, and the signal was down-sampled to $22Hz$ to increase the differences between consecutive samples. Each sample is presented to the network for $40ms$, followed by $50ms$ of pause to relax the neuron potentials and stabilize the output.

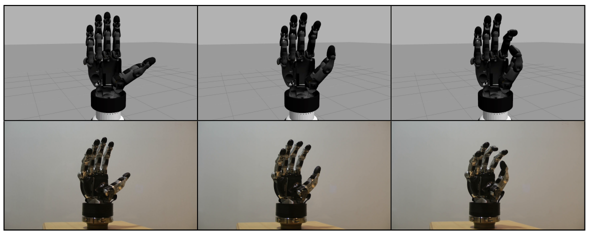


(a)

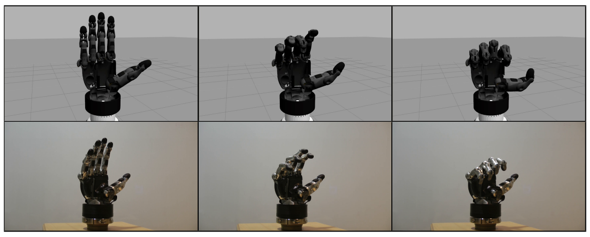


(b)

Figure 5.8.: Adaptive neurons in the grasp type population (Tieck et al. [16]). (a) The membrane potential adapts to continuous input. (b) The mean squared error shows that the SNN learns faster and more stable with adaptation.



(a)



(b)

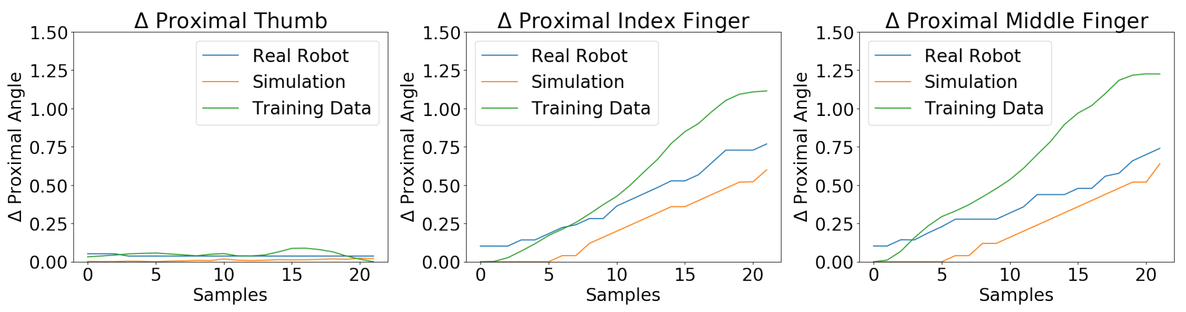
Figure 5.9.: Frame sequences for different grasp motions generated by the SNN (Tieck et al. [16]). Evaluation in simulation (*row 1*) and the real robot (*row 2*). (a) Pinch. (b) Cylinder.

Performing grasping motions

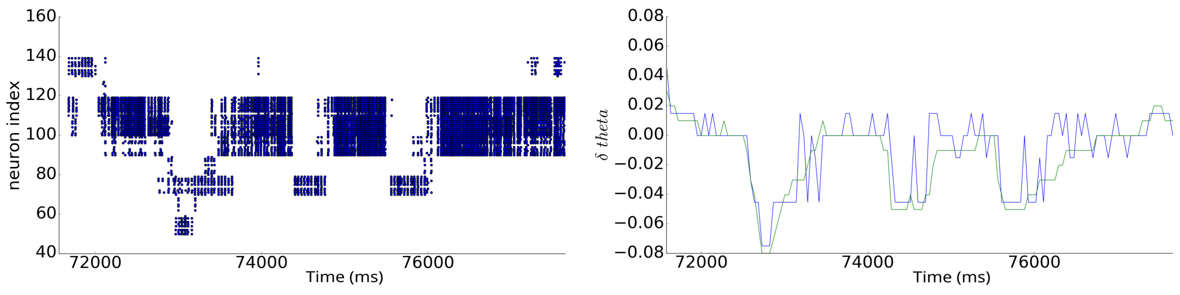
During the experiments, the network activity was recorded during the execution of two different grasp motions (pinch and cylinder). A frame sequence presenting the generated grasp motions is shown in Figure 5.9. The simulated model and real robot performed similar grasp motions, demonstrating the transfer from the simulation to the real robot.

The training data (joint values) are compared with the proprioception (sensor data) obtained from the simulation and the real robot (see Figure 5.10a). Both the simulation and real robot showed a similar behavior as the demonstrated data, which means that the generated motions have similar joint synergies. A segment of the SNN activation for the proximal joint of the index finger is shown in Figure 5.10b. The spike trains from the SNN and the decoded values are compared against the training data. Notice that the SNN output has the structure of the training data.

Figure 5.11a) shows an example of the weight development with and without adaption. Adaption leads to a more stable weight development. While performing a



(a)



(b)

Figure 5.10.: Joint control evaluation and network activation (Tieck et al. [16]). (a) Comparison of training data, simulation, and robot for the proximal joint (*thumb, index, and middle fingers*) during a cylinder grasp motion. The SNN can generate similar synergies for the motion trajectory. (b) A segment activation for the proximal joint of the index finger. On the left the spike trains, and on the right the decoded output values of the SNN, *blue* vs. training data *green*.

cylinder grasp motion, tactile feedback was simulated in two fingers to trigger the inhibitory mechanism to stop the motion.In Figure 5.11b the SNN activity is presented, showing only the spike activity of the most active neuron in one of the output populations for the index and thumb fingers.

5.2.3. Discussion

These experiments are a proof of concept of a biologically inspired control architecture with SNNs for a robot hand to perform grasping motions. The network is capable of learning motor primitives using STDP and execute different types of grasping motions. The grasping motions are represented by the synergies between the joints of the fingers. With the hierarchy of a hand sub-network coordinating finger sub-networks, it is possible to reuse and combine motor primitives (individual finger movements).

All sub-networks learn with the same supervised associative learning mechanism using STDP on human demonstrated grasp motions. The experiments showed that

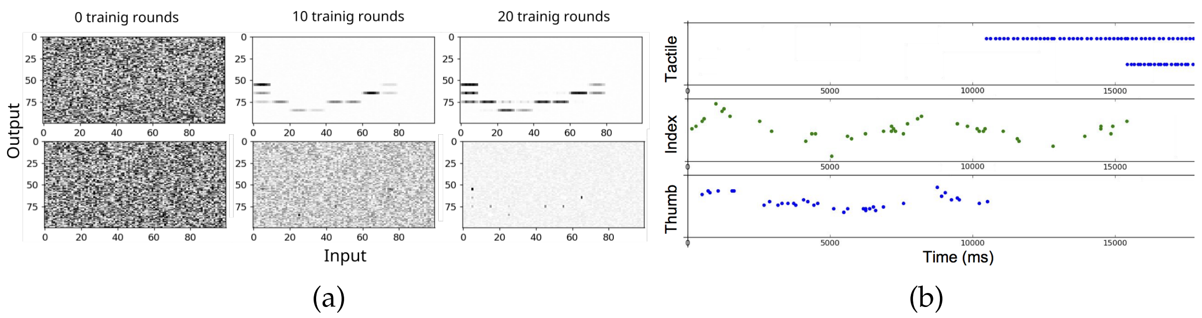


Figure 5.11.: Weight development and tactile feedback (Tieck et al. [16]). (a) Weight development of the index finger network during learning, *row 1* with adaption and *row 2* without adaption. (b) The tactile feedback signal inhibits the activation of two fingers at different times.

the finger networks learn motion synergies by associating the proprioception and the fingertip target with joint changes. The hand network learns to coordinate the fingers, generating the fingertip targets for each finger network given the proprioception and the grasp type signal. After learning, the SNN was able to control both the simulation and the real robot hand. By using a sparse projection to high-dimensional space in the hand network and using a winner-takes-all readout mechanism, notable noise reduction and stable control are achieved. Tactile feedback was incorporated into the finger networks to stop the motion in the case of contact. This behavior can be used to adapt grasping motions to the shape of an object.

A muscle model for the SNN activation instead of the joint position could be incorporated for smooth control. The grasp type used in the hand network is currently an arbitrary representation without semantic information, which could be extended to signals coming from other networks. For example, visual signals coming from higher brain areas can be incorporated to represent grasp affordances (Kaiser et al. [9]) and learn from demonstration as the mirror system (Reithler et al. [177]).

5.3. Triggering finger reflexes using EMG signals

For human robot interaction, the approach presented in Section 5.2 is extended incorporating signals from an Electromyography (EMG) sensor to activate the finger primitives as a reflex in realtime. The interaction of humans and robots (HRI) is of great relevance for the field of neurorobotics. It provides insights on motor control and sensor processing mechanisms in humans that can be applied to robots. Nevertheless, there are different hypothesis explaining how the human motor system work.

Motor primitives can be activated in different ways, for example, as a reflex. A reflex is an involuntary response to sensor stimulation and can be either an immediate complete execution or inhibition of a motion (Byrne et al. [64]). The challenge is how to apply this principle to execute a finger primitive. It is possible to make the grasping behavior more adaptive with finger reflexes incorporated into the hand control. Additionally, by incorporating human signals in real-time, the training and programming of the robot are more flexible. The problem is illustrated in Figure 5.12.

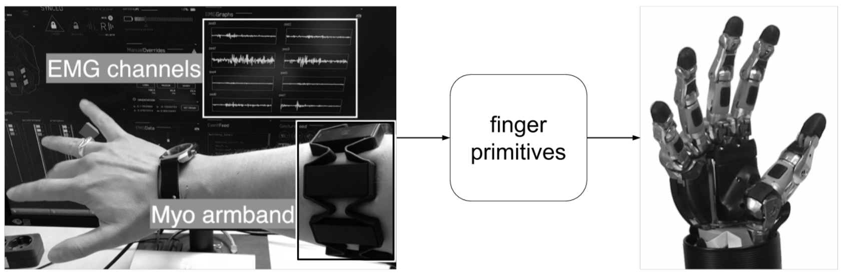


Figure 5.12.: Finger reflexes problem definition (Tieck et al. [25]). Human muscle signals are used to trigger finger motions for an anthropomorphic robot hand. A Myo EMG armband sensor is placed on the forearm, the SNN detects which finger was flexed and triggers a reflex activation of the corresponding finger primitive to control the robotic hand.

EMG is a common tool in medicine and biomechanics. It is used to monitor and study the electrical activity of the muscles. There are different methods to record EMG signals, and they can be either invasive or non-invasive. Research is focusing on processing and classification of EMG signals for clinical diagnoses (Chowdhury et al. [69]) or prosthetic applications (Johannes et al. [132]).

An essential aspect of this approach is the fact that the classification and the motion generation are implemented using a SNN. There are two reasons for doing this. First, the real biological system must do something similar to this using real neurons. Certainly, in biology, the classification would not be based on EMG sensors but would rather be based on neural activity somewhere in the brain, but the classification and generation of movement over time would still need to occur. This means that this system can be seen as an initial model of that biological process. Second, there is a pragmatic/engineering reason to implement this system using artificial spiking neurons. Prosthetic applications benefit from low-power hardware implementations with neuromorphic hardware, which require the model to be implemented with SNNs.

In the following, a system is proposed to control a robot hand using muscle signals from a human. The EMG is recorded with a non-invasive sensor, and the classification is used to trigger the activation of single finger motor primitives as reflexes. The general architecture of this approach is presented in Figure 5.13. This section incorporates findings and methods from Tieck et al. [25, 26].

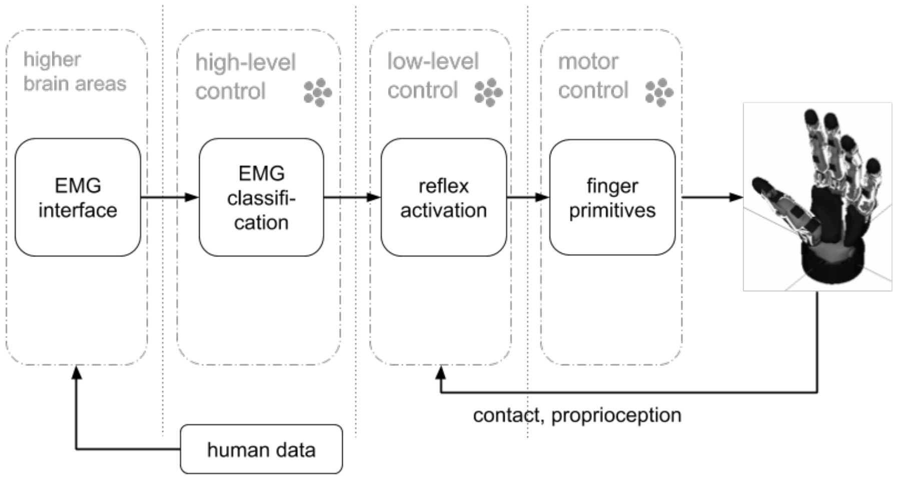


Figure 5.13.: General view of the SNN for triggering finger reflexes using motor primitives. From right to left the layers have an increasing level of abstraction. In *motor control* the finger primitives, in *low-level control* the hand primitives and the reflex activation, in *high-level control* the EMG classification, and in *higher brain areas* the EMG interface for human data.

A SNN was implemented to classify EMG data and trigger the generation of motion as a reflex. An EMG sensor with eight channels was used to record human muscle activity while moving different fingers. First, EMG data was encoded to spikes, and the signals were classified to identify the active finger. After that, the activation signal triggered a neural oscillator to generate motion using a motor primitive. Then, the primitive is mapped to the robot kinematics. Finally, the spikes are decoded to motor commands for the robot. The mapping of myoelectric activity to motor control functions for a task provides an interface for robotic applications with human interaction and a platform to study brain functioning. SNNs provide a challenging but interesting framework to interact with human data.

5.3.1. Methods

The goal is to control a robotic hand with human muscle signals processed by a SNN. For this purpose, specific characteristics for the components are defined. The EMG sensor has to be non-invasive, the finger motions are represented with motor primitives triggered at once to resemble reflexes, and the robot hand has to be controllable with a ROS interface. The SNN is divided into two sub-networks. The first sub-network provides the EMG data interface and classification. The second sub-network provides motion generation and robot control. Human muscle signals are captured with a surface EMG sensor. The first sub-network classifies the EMG signals to detect which finger was active. An activation signal is generated and passed to the second sub-network to trigger single finger reflexes on the real robot. Finger reflexes are modeled according to the robot kinematics. They are implemented with an oscillator that activates a motor primitive, which is mapped to motor commands. In Figure 5.14 an overview of the main components is presented.

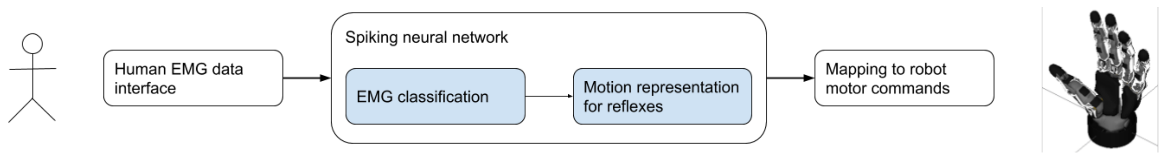

Human EMG data interface
Spiking neural network
EMG classification
Motion representation for reflexes
Mapping to robot motor commands

Figure 5.14.: Concept pipeline with main components (Tieck et al. [25]). Human muscle activity is recorded with a non-invasive EMG sensor, and the data is encoded to spikes. The first sub-network performs a classification to detect which finger was active, and the second sub-network generates motion and maps it to motor commands using motor primitives.

Human EMG data interface and training data

To record EMG data, a Myo (ThalmicLabs [212]) armband is used. It has eight equally spaced segments with non-invasive EMG sensors and operates with a sampling rate of $200Hz$. The armband is positioned around the middle of the forearm (see Figure 5.12). When a finger is flexed, the muscle's electric activity is recorded. To record consistent data with the sensor, the segment with the LED light has to be placed approximately at the same forearm position. After wearing the Myo on and off, slight variations did not influence the recordings to make the trained network unusable. Each channel encodes the individual measurement as $int8$ values.

For each user, a training dataset is required with multiple samples. A sample consists of a continuous sequence of finger flexion in one hand. Each finger has to be flexed down and then extended again. This procedure is repeated starting from the thumb to the pinky. The training data is recorded as a time-continuous EMG stream of all eight channels with appropriate labels for the time windows during which a finger was pressed. A sample recording is shown in Figure 5.15.

Notice that individual channels of the EMG sensor have similar activation for different fingers. Thus, the signal of one channel is not enough to identify the motion of a finger. Therefore, the classification network uses a combination of all eight channels, which provides a unique representation for each finger.

Sub network for EMG classification

After recording training data, the first sub-network is trained for classification. The detailed architecture for EMG classification is presented in Figure 5.16. The raw EMG data from the sensor is feed to the SNN. A population of neurons encodes the signal

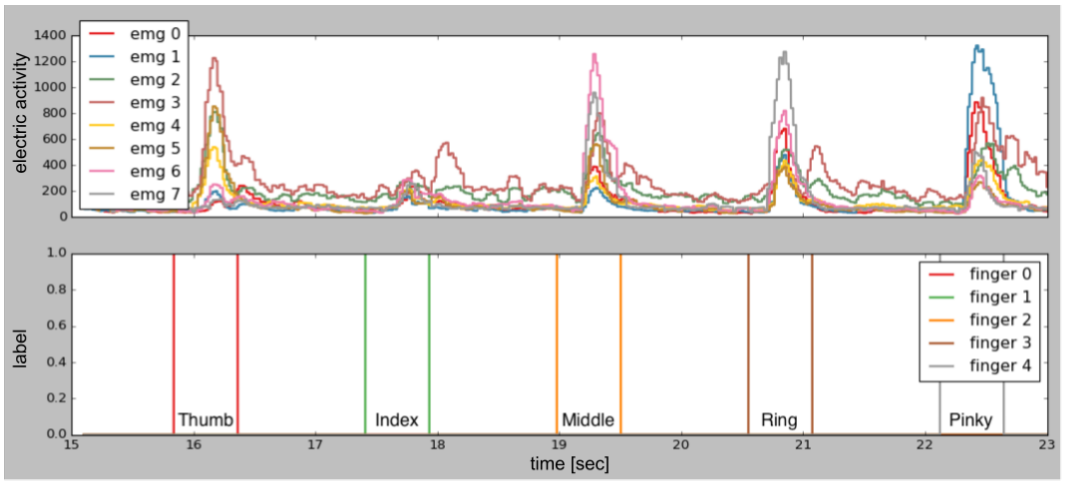


Figure 5.15.: A sample recording for the training of all five fingers in sequence (Tieck et al. [25]). From left to right, the peaks are the EMG activations with the corresponding labels. Each finger is fully flexed and then extended.

stream of EMG input into spikes using stochastic population encoding. A second population of neurons is then trained offline with the whole training dataset for a user as described above.

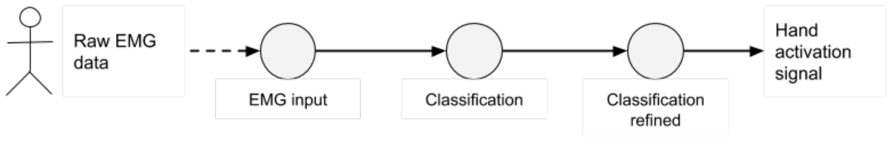


Figure 5.16.: Detailed view for the EMG classification sub-network (Tieck et al. [25]). Each circle represents a population of spiking neurons. Raw EMG data is recorded from the user and is encoded into spikes. The signals are classified to determine which finger was activated. The classification signal is refined and amplified to obtain a clear hand activation signal for the motion generation sub-network.

The learning rule for offline training the classification population is PES using the labels from the training data as error signals E. PES is implemented in Bekolay et al. [52], and was first presented in MacNeil et al. [153]. For the weights w_{ij} from pre-synaptic population *i* to post-synaptic population *j*, the update rule is defined as

$$\Delta w_{\mathrm{ij}} = \kappa \alpha_j e_j \cdot E a_i, \tag{5.1}$$

with κ a scalar learning rate, α the gain or scaling factor, e the encoder for the neuron, E the error to minimize, and a the desired activation.

After the classification by the second population, the signals are low in amplitude and close to each other. Therefore, a population is added to refine the classification by amplifying the signals above a threshold and generating the hand activation signal. The resulting activation signal is passed over to the motion generation sub-network. All populations are connected all to all. Examples for classification of all the fingers are provided in Figure 5.21.

Sub-network for motion generation and mapping to the robot

A population processes the hand activation signal from the previous classification to trigger the appropriate finger reflex. A reflex is a single execution of a motor primitive based on specific stimuli. Accordingly, the motion generation sub-network of the SNN is divided into reflex activation and motor primitive layers. The motor primitives are modeled according to Section 3.2, and the reflex activation as in Section 3.3.3. The whole architecture for the representation of reflexes is presented in Figure 5.17.

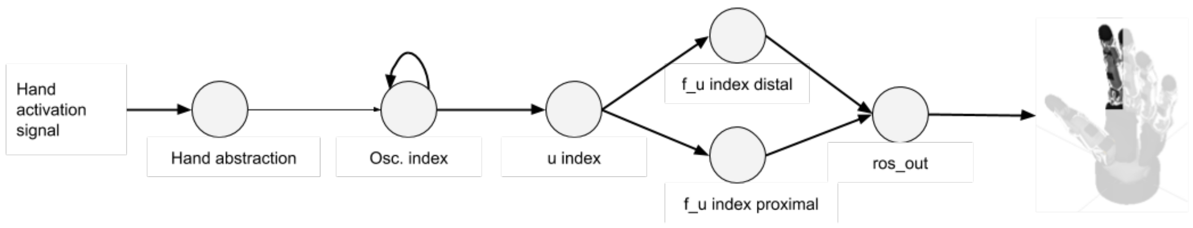


Figure 5.17.: Detailed view for the motion generation sub-network (Tieck et al. [25]). Each circle represents a population of spiking neurons. The hand abstraction population processes the hand activation signal to extract individual finger activations. Reflexes are modeled as oscillators that oscillate only once to activate a motor primitive. The neural activity is decoded to motor commands for each finger.

Integration off all components

A detailed architecture of the full SNN is presented in Figure 5.18. Notice that the primitives for the thumb, ring, and pinky are mapped to one actuated joint, whereas the index and middle finger primitives are mapped to two joints.

5.3.2. Experiments

The experiment setup is shown in Figure 5.19, and consists of a human user, an EMG sensor, a robot hand, and the simulation of the SNN. A user wears the EMG sensor (Myo armband) in the forearm, and the signals are sent via Bluetooth to the computer. The computer receives the EMG data and inputs it to the SNN simulation running in Nengo (Bekolay et al. [51]). The computer communicates at the same time with the

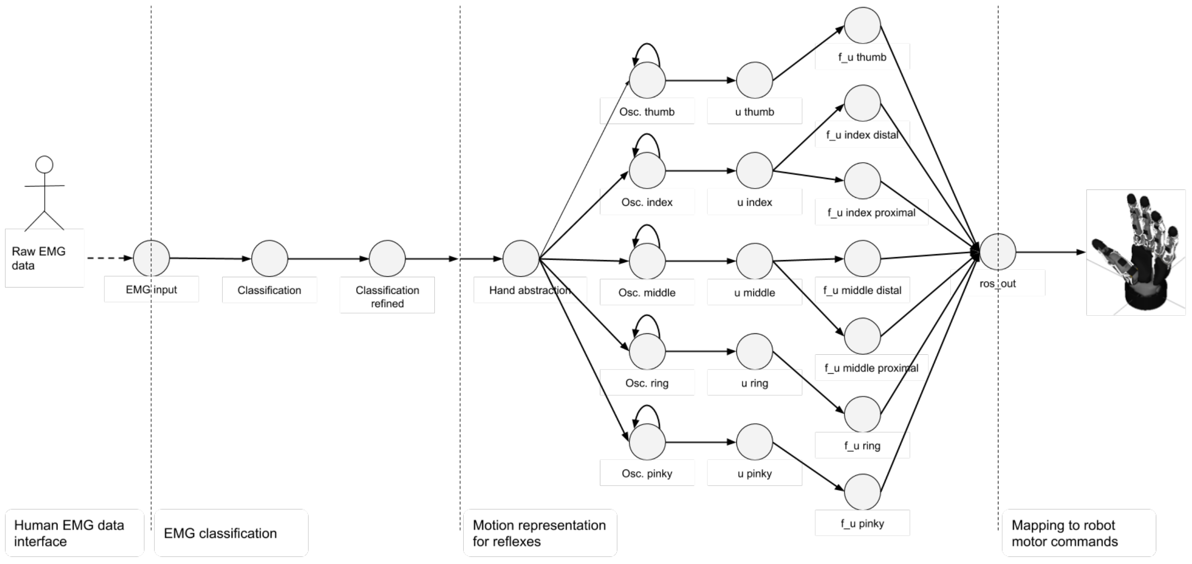


Figure 5.18.: Detailed architecture of the SNN with EMG classification and motion generation sub-networks (Tieck et al. [25]). Each circle represents a population of spiking neurons. The dotted lines divide the conceptual components, named on the bottom according to Figure 5.14.

robot hand (Schunk SVH) via ROS (Quigley et al. [176]). To control the robot hand, the official Schunk ROS driver is used (Heppner [121]).

Implementation details

The SNN was implemented with Nengo using LIF neurons. An sample of the whole network running is shown in Figure 5.20. The structure corresponds with Figure 5.18. The eight EMG channels are encoded by a population as stochastic spike rates based on their values. After training the network offline with different datasets from the same user, the EMG classification is performed. The classification signal is then passed over to trigger the motion generation. The reflexes are implemented as oscillators that activate motor primitives. The motor primitives are mapped to the robot kinematics as defined in the methods section.

Training data

For each user, a set of training data is required to train the SNN. Training data for the classification network was recorded in one session of 60 seconds. During that time, individual fingers were pressed against the palm and subsequently returned to a resting pose. The fingers are flexed in sequence from thumb to index finger, with each flexion lasting between $300ms$ and $500ms$. Together with the resting time, one

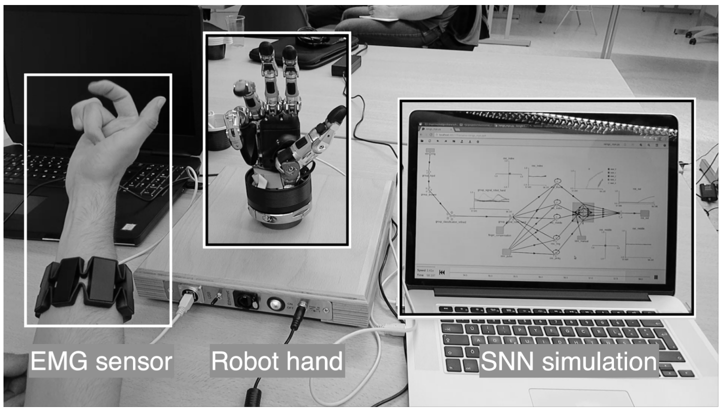


Figure 5.19.: Experiment setup (Tieck et al. [25]). (*left*) The EMG sensor (Myo) is placed in the forearm of the user. (*right*) The SNN is simulated in Nengo. (*middle*) The robot hand is a Schunk SVH 5-finger hand. All components are connected using ROS.

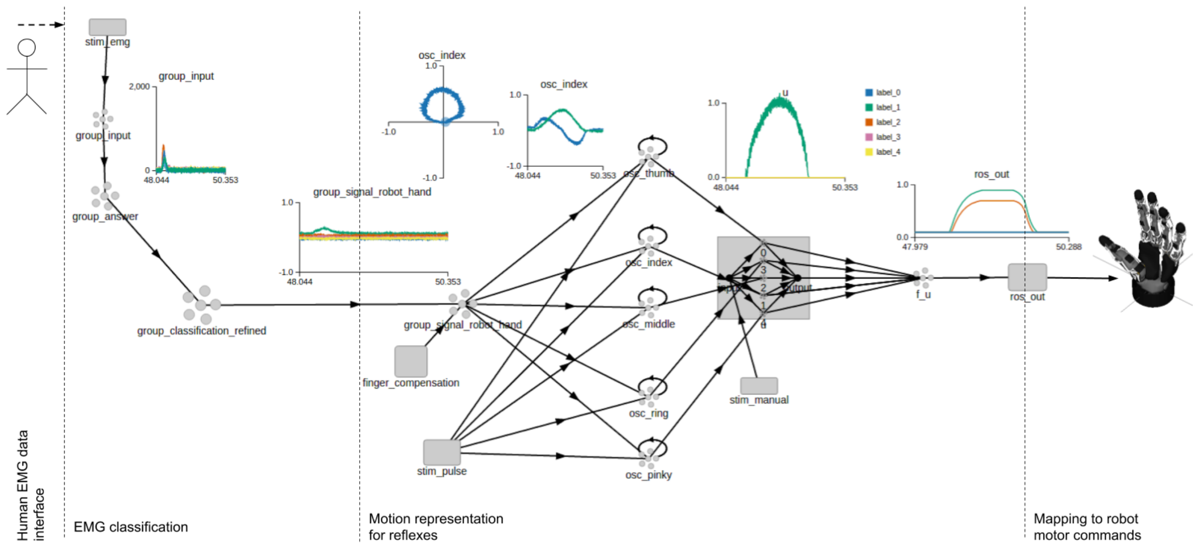


Figure 5.20.: Full SNN implemented in Nengo (Tieck et al. [25]). There are four main components: human EMG data capture and manipulation, EMG classification, motion generation, and finally, the mapping to the robot. The structure corresponds to Figure 5.18.

cycle takes around $7.5s$, and a total of 8 cycles are performed. A sample recording with all 5 fingers is presented in Figure 5.15. The data is labeled during recording for each finger, and all eight EMG channels are active.

Processing of EMG data and classification

The EMG classification sub-network is presented in Figure 5.16. The first group of 800 neurons (*EMG input*) was activated with the raw EMG data. The second group of 500 neurons (*classification*) was trained with pre-recorded training data to classify the fingers. Then, the third group of 500 neurons (*classification refined*) was used to separate and amplify the signals. A final group of 500 neurons (*Hand activation signal*) was trained to generate one single signal for a specific finger and was connected to 5 groups representing the different fingers for the robot hand.

In Figure 5.21, samples of the SNN are presented showing the activation of the different fingers. As it can be seen, the eight channels of EMG have different data for each finger. The signals are processed with the SNN, and the activation of the different populations can also be observed. The output of the classification is a dominant activation of one of the populations representing each finger.

Motion generation and interface to the robot hand

The data presented corresponds to a reflex motion of the index finger. The corresponding activity of the SNN for a reflex motion of the index finger is presented in Figure 5.22. The signal that triggers motion generation comes from the classification sub-network. An oscillator is activated for each finger. Observe the circular activation of the oscillator population when decoded in a plane XY. u is decoded and mapped to one or more joints in the robot hand from this circular activation. Observe that the mapping is performed to two joints of the index finger. Finally, the neural activity is decoded and send over ROS to the robot hand. The resulting motion of the robot is presented in Figure 5.23.

In order to evaluate the accuracy of the classification, one random user was selected. Then he was asked to perform a sample of 50 trials with each finger. The EMG data was feed to the trained network, and the classification output for each trial was recorded. The results are summarized in Figure 5.24. Only the pinky finger had a 1.0 accuracy for this user, which means that all the trials were classified correctly. For the other fingers, there are either false detections or no classification at all.

The table in Figure 5.25 shows the mapping schema for the robot hand. The "Joint name" column corresponds to the ROS topics described in Heppner [121] to control the joints. A primitive is used for each finger, indexed in column "Primitive". Note

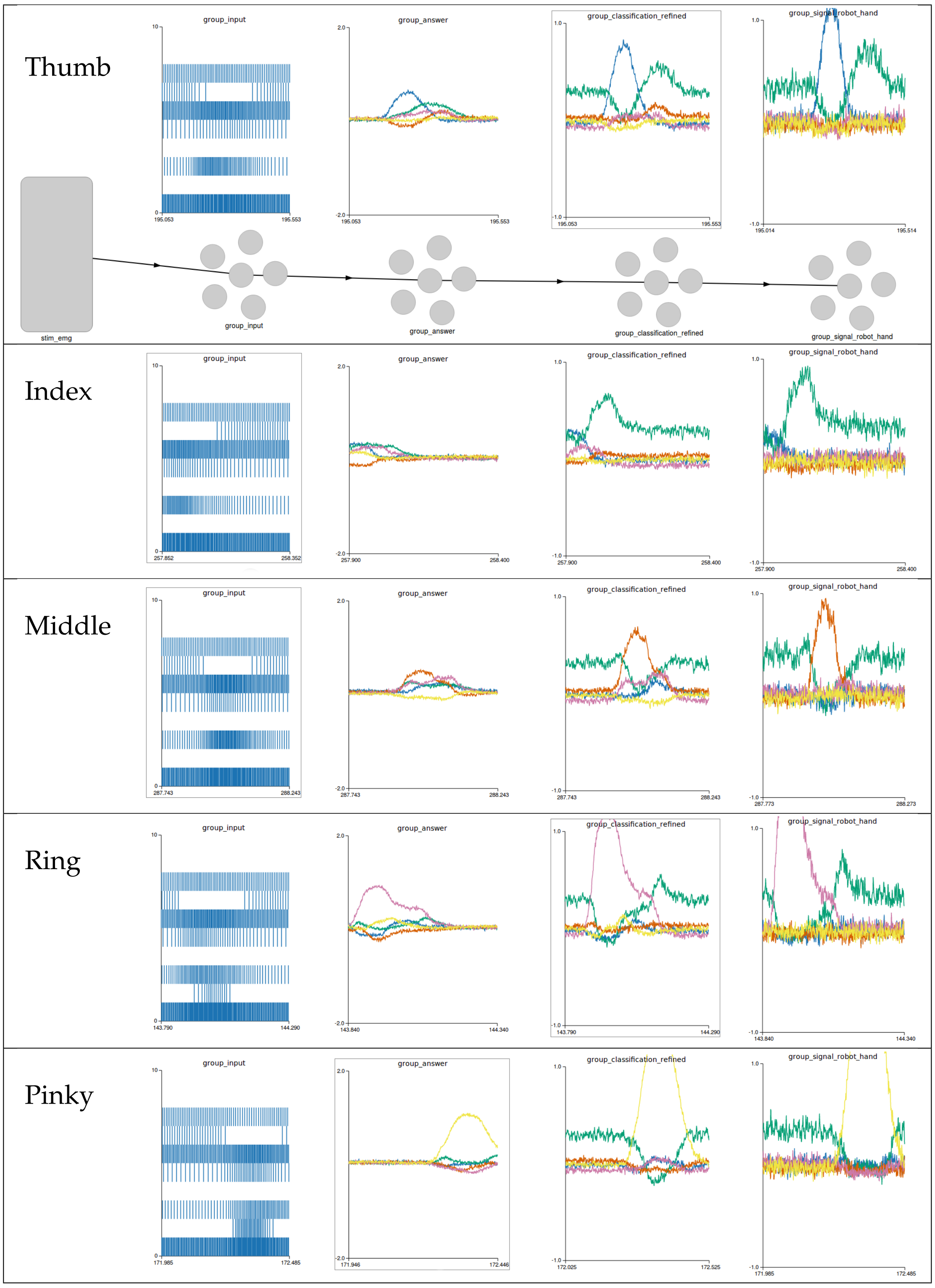


Figure 5.21.: Classification of EMG for the different fingers (Tieck et al. [25]). From left to right, the plots show the EMG encoded into spikes, the classification output, the refined classification, and hand activation signals.

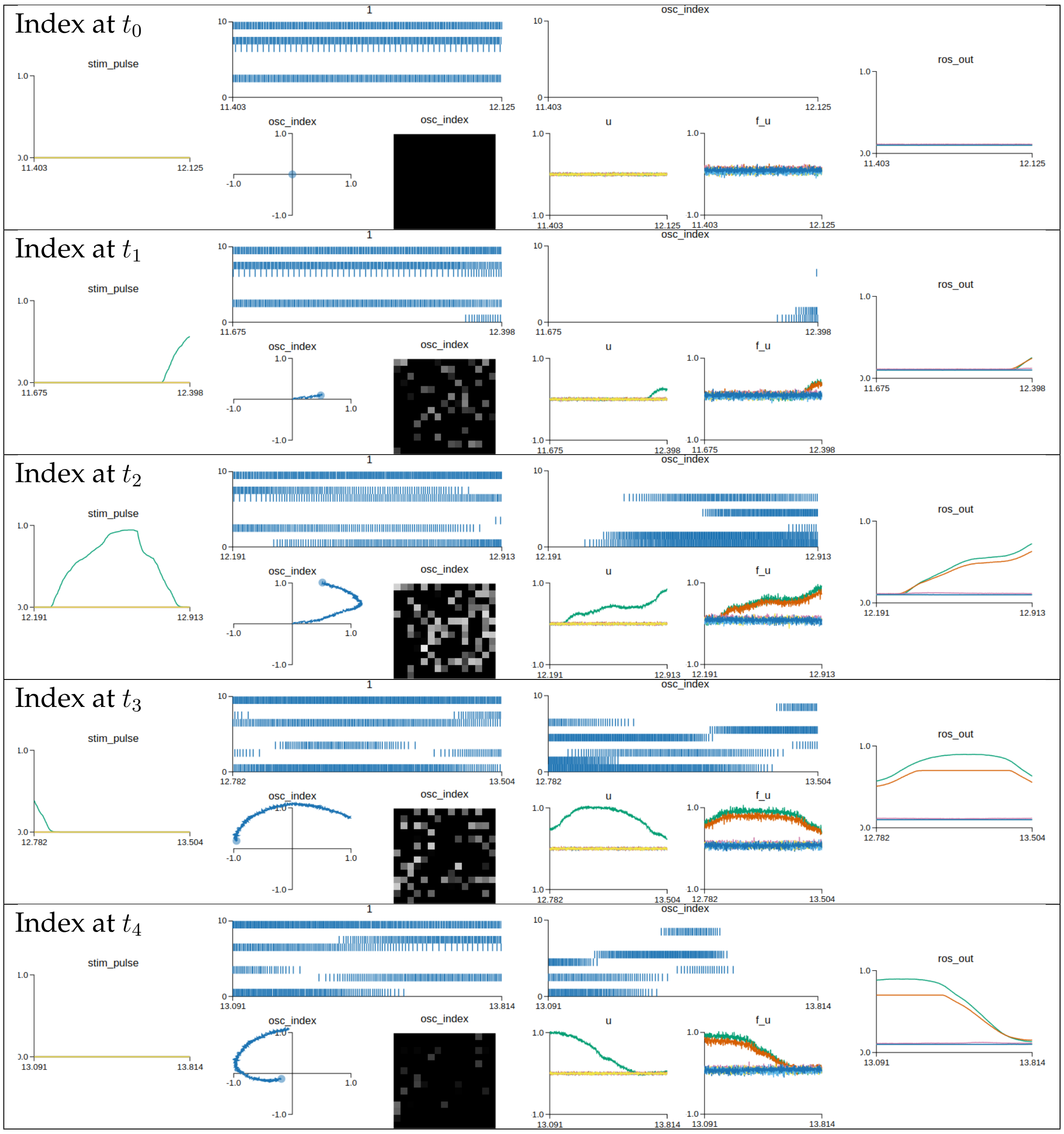


Figure 5.22.: Motion generation for the index finger (Tieck et al. [25]). (*left*) Finger activation signal from the EMG classification. (*middle left, group of three plots*) Spike train of activity in the oscillator, decoded activity in the plane XY, and a raster plot color-coded by the neuron's activity. (*middle right, group of three plots*) Decoding of u and mapping $g(f(u))$ to the robot kinematics. (*right*) Motor commands for the robot.

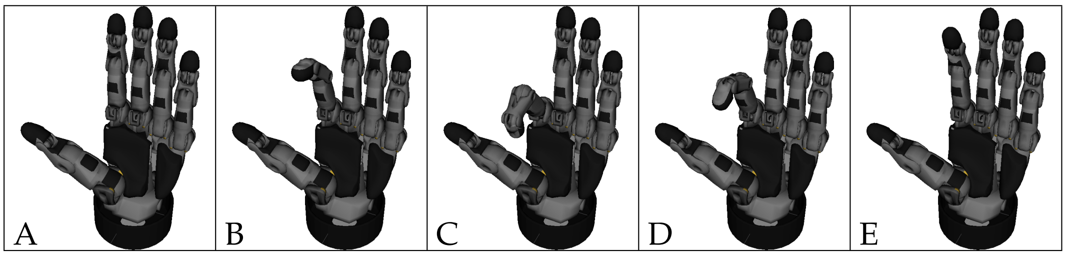


Figure 5.23.: Frame sequence of the index finger motion generated by the SNN (Tieck et al. [25]). The motion corresponds to the activation in Figure 5.22.

Finger moved	**Number of trials**	**Thumb class**	**Index class**	**Middle class**	**Ring class**	**Pinky class**	**None class**	**Accuracy**
Thumb	50	46	0	0	1	0	3	0.92
Index	50	0	43	1	0	2	4	0.86
Middle	50	0	3	40	0	0	7	0.80
Ring	50	0	1	2	44	0	3	0.88
Pinky	50	0	0	0	0	50	0	1.00

Figure 5.24.: Table of classification accuracy (Tieck et al. [25]). For one random user, 50 trials for each finger were performed recording which finger was detected by the SNN.

that the two joints in the index and middle fingers are mapped to the same primitive. The "min" and "max" values for each joint complete the table.

5.3.3. Discussion

The proposed SNN activates motion reflexes on a robotic hand based on human EMG data. The network classifies the EMG signals to detect finger activation. Based on it, single-finger reflexes are triggered. The finger reflexes are modeled with motion primitives and mapped to the robot kinematics.

As can be seen in Figures 5.15 and 5.21, the index finger showed almost no discernable signal in the raw EMG data. For the index finger, the signal is not clear, and the output is sometimes ambiguous. As a consequence, the classification sub-network generates a weak and low activation signal for the index finger that is propagated through the following populations and leads to a fake classification of other fingers. This approach focused on single finger movements, so data with multiple fingers were not considered, only movements in quick succession of single fingers. The EMG signals were used to trigger the execution of the reflexes. Additional research is required to use EMG signals to perform discrete control of the finger positions.

Using a second Myo EMG sensor can provide additional input from different muscle areas of the arm and improve classification results. Ideally, the second EMG sensor

Joint name	**Primitive**	θ_{min}	θ_{max}
Thumb_Flexion	0	0	0.3
Thumb_Opposition	-	-	-
Index_Finger_Distal	1	0	0.9
Index_Finger_Proximal	1	0	0.7
Middle_Finger_Distal	2	0	0.9
Middle_Finger_Proximal	2	0	0.7
Ring_Finger	3	0	0.7
Pinky	4	0	0.7
Finger_Spread	-	-	-

Figure 5.25.: Table for the robot mapping schema (Tieck et al. [25]). The mapping is defined with the joint name, the primitive used for each finger, and the joint angle intervals for each joint. The joints "Thumb_Opposition" and "Finger_Spread" remained constant.

could be located close to the wrist, closer to the fingers, and the hand (Tenore et al. [211]). A more extensive set of training data could improve the classification.

5.4. Compliant control for soft-grasping with a hierarchy of motor primitives

In order to control the whole hand, the principles to model finger primitives in Section 5.3 are extended with two additional degrees of freedom — thumb opposition and finger spread. The contact detection circuit presented in Section 3.3.4 is incorporated to trigger reflexes and activate a compliant controller to perform soft-grasping. Humans developed advanced and flexible grasping capabilities with evolution thanks to a combination of an adaptive hand and efficient control. Nevertheless, most robotic applications use a vacuum, 2-finger, or custom-made grippers (Wolf et al. [222]) which are acceptable for production applications. However, these methods lack adaptability in other environments where grasping novel objects without knowing their exact geometric and physical properties is required.

Humans do not just plan a grasping motion and then execute it. There is actually a combination of control loops working together to grasp an object. The hand can adapt its motion and force based on sensor feedback from the object if it moves or deforms. This capability is called soft-grasping (Caldwell et al. [65] and Bonilla et al. [56]). The challenges are how to detect contact and force with the object and how to adapt the motions accordingly. In this sense, soft-grasping is modeled with two combined control loops, one is based on motor primitives, and the other is a compliant controller activated by a reflex mechanism. An illustration of the problem is presented in Figure 5.26.

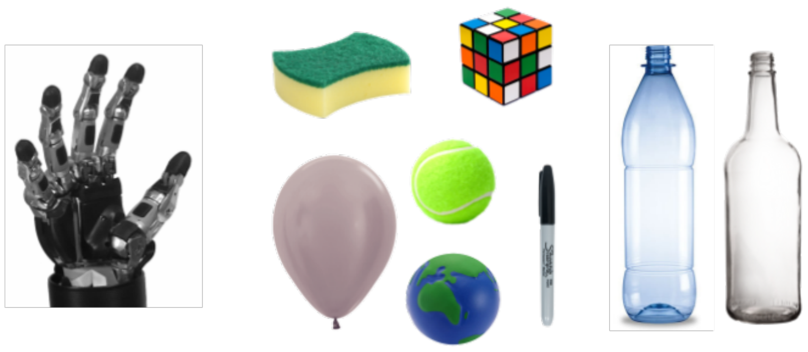


Figure 5.26.: Soft-grasping problem definition. Given a limited number of hand primitives, adapt them to grasp objects with different shapes, stiffness and sizes without knowing their exact geometric and physical properties. Use compliant control without force sensors to control a real anthropomorphic robot hand.

This approach is motivated by the biological principles for motion representation with motor primitives, the concepts of adaptive and compliant control (DeWolf et al. [91]), and the characteristics of event-based computation (Zambrano et al. [228]) with SNNs. There are studies that show evidence of muscle synergies for grasping (Sburlea et al. [189]), the relation between human responses (Abler et al. [41]) and the stiffness regulation in the hand (Crago et al. [75]), the classification of grasping types (Cutkosky [78]), and the generalization of muscle patterns as building blocks for grasping (Scano et al. [190]).

Compliant control with a robotic hand can be performed with soft (Thuruthel et al. [213]) or flexible hardware (Bonilla et al. [56]), or with software using sensor feedback (Caldwell et al. [65]). Different approaches combine software compliant control with a torque sensor and robust modeling (Scherzinger et al. [192]), or with online learning for adaptive control (DeWolf et al. [92]), or with cerebellar principles (Capolei et al. [66]), or with reflexes and predictive control (Urbain et al. [217]).

In the following, a system is presented for soft-grasping with an anthropomorphic robotic hand. The approach takes inspiration from biology and integrates the principles of motor primitives with SNNs to model the hand with a hierarchy, to model finger reflexes, to coordinate multiple primitives, and to combine different activation modalities. The general architecture of this approach is presented in Figure 5.27. This section incorporates findings and methods from Tieck et al. [21].

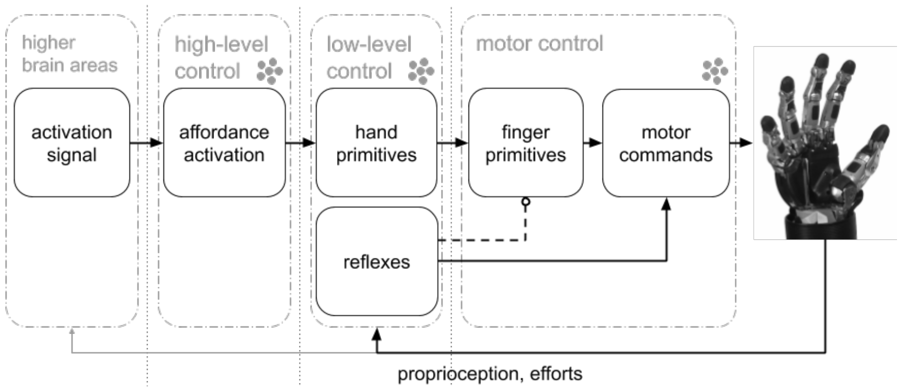


Figure 5.27.: General view of the closed-loop control architecture with SNN for soft-grasping using motor primitives and reflexes (Tieck et al. [21]). From right to left the layers have an increasing level of abstraction. In *motor control* there are the finger primitives, in *low-level control* the hand primitives and the reflexes, in *high-level control* the affordance activation mechanisms, and in *higher brain areas* the activation signals.

Grasping is modeled with motor primitives in a hierarchy, with finger primitives representing synergies between joints and hand primitives representing different affordances coordinating the fingers. This modeling simplifies the control of the hand and allows to generalize each grasp for different objects. The compliant controller is triggered by a contact detection mechanism modeled as the circuits of interneurons in the spinal cord. Objects with different shapes, stiffness, and sizes are graspable without knowing their exact geometric and physical properties. It is not necessary to compute the inverse kinematics or to calculate complex contact point planning. The approach can represent the adaptive grasping capabilities of a human hand, and it can be used on different robots. Soft-grasping with anthropomorphic hands is a key capability for robots interacting in an environment with objects shaped for humans (Pfeifer et al. [170]).

5.4.1. Methods

An approach for soft-grasping with an anthropomorphic robotic hand is proposed to grasp objects with different shapes, stiffness, and sizes using SNNs. Grasping motions are represented with a hierarchy of motor primitives extending the methods in Sections 5.2 and 5.3. The network combines two control loops to generate complex grasping motions. The first one is fast and reactive, generated by the trajectory control from the motor primitives, to close the hand using human affordances (Cutkosky [78]). The second one is precise and adaptive, generated by the compliant controller, triggered with a reflex using the motor currents.

The detailed architecture for the SNN is presented in Figure 5.28 with four main components: finger primitives, hand primitives, affordance activation and reflexes. Each oval is a sub-network. The connection between compliance and the joints is simplified in the diagram; it is also all to all. The finger primitives represent the joint synergies for a hand closing motion. The hand primitives represent different affordances coordinating the fingers. The affordance activation mechanism generates the activation patterns for the hand primitives. There are two types of reflexes activated by contact, one inhibits the movement of the fingers, and the other activates the compliant controller. Contact detection is modeled as the circuits of interneurons in the spinal cord. The compliant controller uses the efforts from the motors to control the force the fingers can apply. The effort is an indirect measurement of the torque.

Finger primitives and robot kinematics

The finger motion is modeled with a motor primitive representing the joint synergies between the finger joints during a closing motion. The principles to model the finger motor primitives are based on Section 3.2. Additionally to the five fingers, the modeling was extended with two more degrees of freedom — thumb opposition and finger spread. Thus, there are seven finger primitives — thumb, thumb opposition, index, middle, ring, pinky, and finger spread — as shown in Figure 5.28.

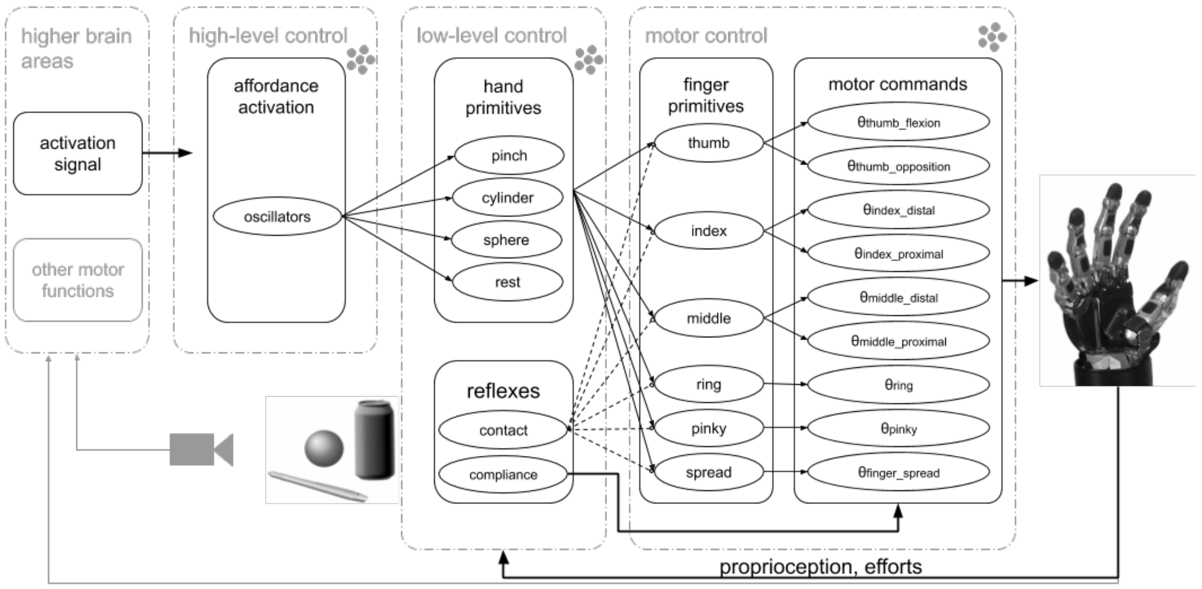


Figure 5.28.: Detailed view of the closed-loop architecture for soft-grasping with SNN (Tieck et al. [21]). The *motor neurons* generate *motor commands*. The *finger primitives* represent joint synergies. The *hand primitives* represent different grasping motions. In *reflexes*, contact triggers reflexes to inhibit the motion of the fingers and activate the compliant controller. The *affordance activation* provides continuous activation signals using oscillators.

Hand primitives and control hierarchy

The hand motion is also modeled with a motor primitive, but instead of mapping to joints, it maps to the activation parameters of the finger primitives. The hand primitives are organized in a hierarchy coordinating the finger primitives. An initial hierarchical modeling of the hand was presented in Section 5.2, and a similar idea was also used to coordinate multiple legs in Section 6.2. The hand primitives represent grasping affordances for sphere, cylinder, and pinch according to (Cutkosky [78]) and the rest position. By using motor primitives for grasping, the complexity of the hand control is reduced to one activation parameter for each affordance. This type of activation reflects the muscle synergies of human grasping (Scano et al. [190]).

A hand primitive is modeled as a mapping of the activation parameter u to a sequence of activations of the finger primitives during the execution of a grasp. The initial grasping pose (pre-shaping) and the final pose with the hand closed, define the primitive as the min open and max closed parameters (see Figure 5.34). Each hand primitive is connected to all finger primitives. There are four hand primitives as shown in Figure 5.28. A sub-network in hand primitives and the initial and final configurations for all the joints are required to define a primitive to represent a new grasping motion for a different affordance. The rest of the SNN can be reused, and the compliant controller adapts the motions online to the shape of the objects.

Affordance activation mechanisms

The affordance activation mechanisms create the activation patterns for the hand primitives. An external activation signal is used to activate the hand primitives. A population of neurons generates neural activity for the duration of the grasping motion. It is an oscillator that oscillates only once (see Section 3.3.2), and the activity is decoded for the activation parameter u as

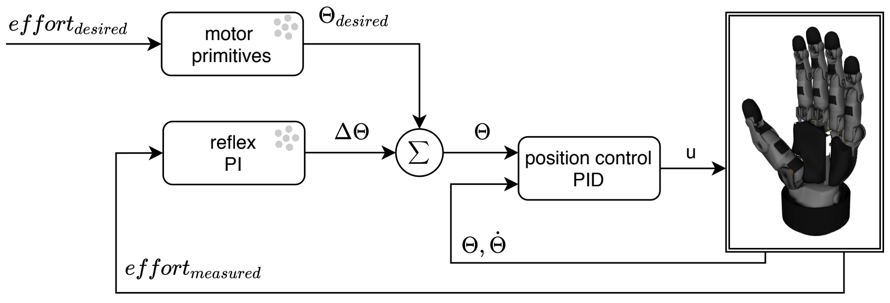
$$u = -\frac{1}{2}cos(t \cdot \frac{2\pi}{T}) + \frac{1}{2}. \tag{5.2}$$

Reflexes and contact detection

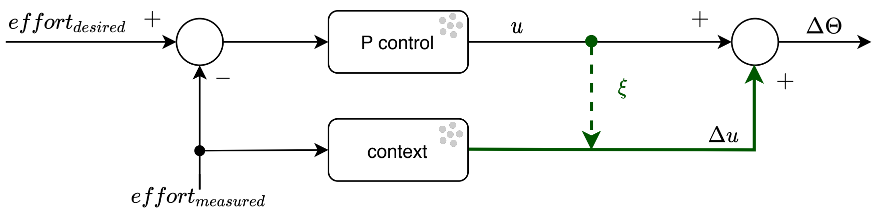
The reflexes (see Section 3.3.3) and the circuit for contact detection (see Section 3.3.4) are the components which provide the adaptation and flexibility to the grasping motions required for soft-grasping. These mechanisms are extended to activate the compliant controller and to change the activation parameters from the SNN. $\Delta\Theta$ is calculated as the change over time of the joint position, using the actual Θ_t and the previous Θ_{t-1}, provided by a delayed recurrent connection. The interneuron is excited by the effort feedback from the motor and inhibited by $\Delta\Theta$. This way, the interneuron only detects contact if the effort increases and the corresponding joint is not moving. Therefore, changes in the effort caused by the non-linearity of the robot dynamics are ignored. The effort is an indirect measurement of the torque. Two types of reflexes are triggered with contact. The first type provides inhibition and stops the motion. When contact is detected in one finger, the reflex inhibits the respective primitive, and the joint position is mapped as the new target position. The second type of reflex mechanism activates a compliant controller for that finger.

Compliant controller and adaptation

The control schema is a cascaded setup of two controllers, the motor primitives and the compliant controller (see Figure 5.29). The two controllers are combined using the feedback from the motors. With the measurement of the motor current, an effort of each joint can be estimated. The effort is an indirect measurement of the torque. The compliant controller uses the effort feedback to control the force a finger can apply. It is a PI effort controller, extended with online learning for adaptation It is activated with contact, and it is inhibited if no contact is detected or if the hand is opening. Target efforts can be set to each joint individually to determine the force and sensitivity of the grasp. The target values for the effort controller can be changed for each finger on the fly by the network. This characteristic provides even more flexibility and another degree of freedom for the control.



(a)



(b)

Figure 5.29.: Compliant controller schema (Tieck et al. [21]). The effort is an indirect measurement of the motor torque. (a) Cascaded compliant controller for each finger. (b) Adaptive part with online learning, corresponding to *reflex PI* in (a).

The controller was initially modeled as a classical PI controller implemented with SNN. The I part was divided into two parts with different parameters to reduce oscillation. A fast-reacting I part with an offset compensates the delayed system answer, preventing the system from overshooting while the system recovers from a delayed response. A slow reacting I part is adjusted by the system delay and produces less oscillation. No D part was used because the effort measurements from the motors are very noisy. The digital I part can be described as

$$I_{(k)} = k_I \cdot \sum e_{diff,(k \cdot \Delta t)} \cdot \Delta t \tag{5.3}$$

where k_I is the factor of the I part and e_{diff} is the control error between the target value and the actual measured value. This can be converted into the differential equation with the additional P component

$$y_k = y_{k-1} + k_I \cdot u_k \cdot \Delta t \tag{5.4}$$

with the controller output y_k.

The initial contact points must change if the object moves or deforms due to the interaction between the fingers and the object. Ideally, a part in the controller can learn online to compensate for these changes without calculating the exact contact points or the inverse kinematics. For this, an adaptive control schema is proposed (see Figure 5.29b). The adaptive part works as an additional I part of the controller with dynamic parameters (DeWolf et al. [91]). For the online adaptive part (green connections), the PES rule is used as implemented in NEF (MacNeil et al. [153]). The number of neurons and the learning rate determine the factor of the adaptive control, and define how fast the adjustments are made. PES adjusts the decoders Δd_i of a connection to minimize an error signal. The change in weights w_{ij}, is given by

$$\Delta w_{ij} = \Delta d_i \cdot e_j \alpha_j \tag{5.5}$$

$$\Delta d_i = -\frac{\kappa}{n} \cdot \xi a_i, \tag{5.6}$$

where α_j is the gain, κ the global learning rate, n the number of neurons, ξ the error signal, e_j the encoder of the postsynaptic neuron and a_i the presynaptic activity. The pre-synaptic population is indexed by i and j indexes the post-synaptic population. The resulting connection Δu is added to control signal and it is defined as

$$\Delta u(t) = \sum_{i=0}^{n} d_i \cdot a_i(x(t)), \tag{5.7}$$

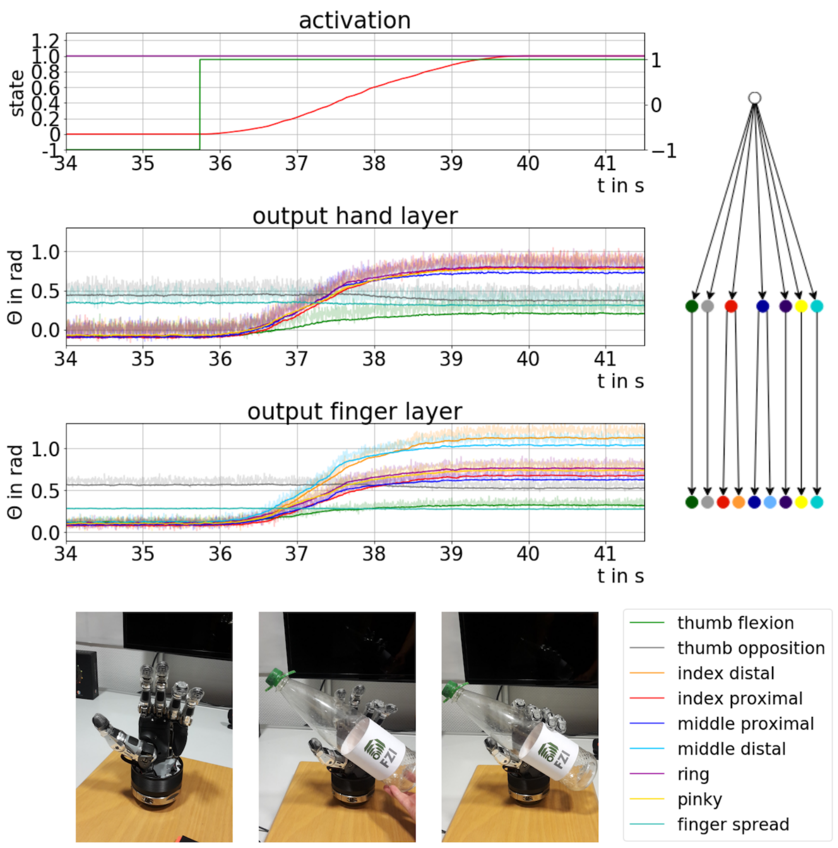
where $a_i(x(t))$ is the activity of neuron i given the input $x(t)$. The error signal ξ is given by the classical PI part and the correction provided by Δu is affected by the learning rule adapting d_i.

5.4.2. Experiments

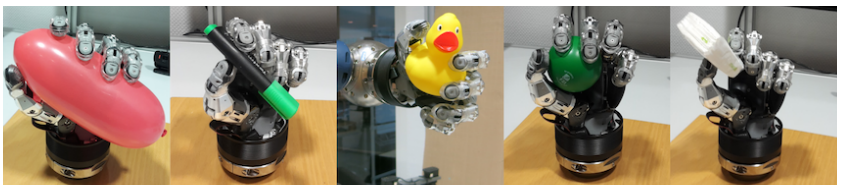
An anthropomorphic Schunk SVH 5-finger hand was used to evaluate the performance of the SNN for soft-grasping. For the experiments, the hand was mounted in a test base and a robotic arm. Three types of grasping motions were modeled — sphere, pinch, and cylinder. The affordances were activated with an external signal to trigger the motion. First, the activation of the different motor primitives and how the affordances adapted to different objects were evaluated. Then, the sensitivity of the compliant controller and its activation was evaluated. Finally, it was evaluated how the adaptive controller can learn online and how it compares to the PI controller.

Motor primitives activation and affordance evaluation

A grasping motion showing the activation of the motor primitives is presented in Figure 5.30a. The plots show the activation signal and the activations of the hand and finger layers. Observe in *output finger layer* that the network generates a smooth and continuous trajectory that reaches stable final states for each finger joint. The hierarchical structure of the primitives is illustrated on the right as a tree. It is color-coded with the plot lines, and the resulting grasping motion as a frame sequence is shown on the bottom. Several objects were selected to evaluate how the different affordances adapt, with different shapes, stiffness, and sizes. Among them: a plastic bottle, a softball, a tennis ball, a sponge, a rubber duck, different balloons, a pen, and a tissue pack (see Figure 5.30b).



(a)

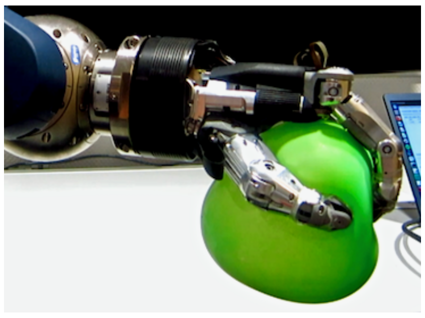


(b)

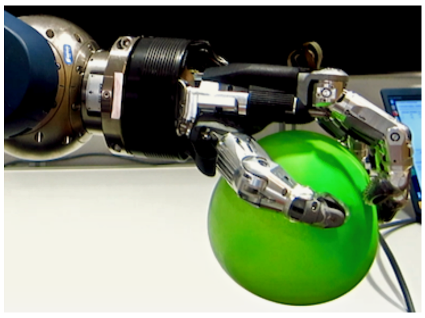
Figure 5.30.: Activation of different affordances (Tieck et al. [21]). (a) Affordance activation, network output of the hand and finger primitives, and a frame sequence of a grasp. (a) Experiments with different objects.

Compliant control evaluation

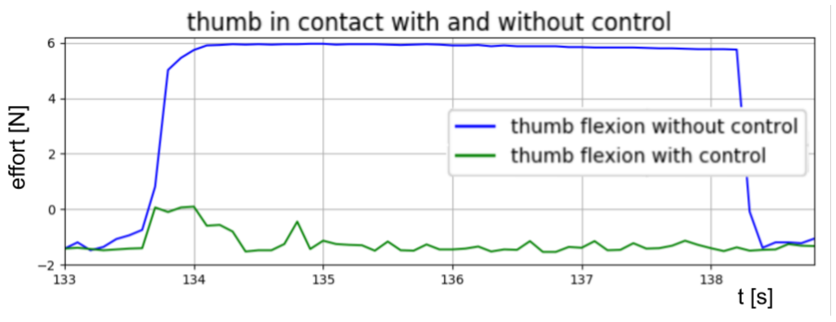
The effort threshold controls the force that a finger can apply to the object being grasped. The motor drivers of the robots usually have the option to set a safety max-



(a)



(b)



(c)

Figure 5.31.: Compliant control with different threshold parameters (Tieck et al. [21]). Using (a) high and (b) low effort thresholds. (c) Effort plot for the thumb flexion for both cases.

imum value for the allowed effort. The motor driver clips the control to protect the robot, which causes deviations of the control error and results in delays at the start of the control. Changing this on the fly is problematic as the configuration of the motor driver has to be reloaded. With this SNN, it is possible to change the effort threshold on the fly by the network, which provides flexibility and another degree of freedom for the control. The compliant controller proved to be very sensitive, and the threshold could be set to be very low, allowing even the manipulation of balloons (see Figure 5.31). In Figure 5.31a the hand is pressing hard using the maximum effort threshold, whereas in Figure 5.31b it is pressing soft using the effort threshold. In Figure 5.31c the effort plot for the thumb finger for the maximum and minimum effort. Notice that the motor driver clips the maximum. This maximum value is actually the lower value that the motor driver can control because the effort caused by the non-linearities of the robot dynamics will be higher, and the motor driver will not move the robot. The internal motor driver defines maximum efforts for each joint in as: thumb flexion 6N; index proximal 5.5N; index distal 4.1N; middle distal 4.1N; middle proximal 5.5N; ring 2N; and pinky 1.6N. The controller can go below these values and maintain the effort between -0.5N and +0.5N around the target value. The minimum effort that can be achieved depends on the joint, and it is between -1.5N and 1.0N thanks to the contact detection mechanism (see Section 5.4.1).

A detailed grasp with the adjusted parameters is shown in Figure 5.32. The output of the *effort control* is converted to a joint position and is added to the finger layer output. When contact is detected the reflex network *switch* activates the controller and the *measured position* is affected by the effort controller.

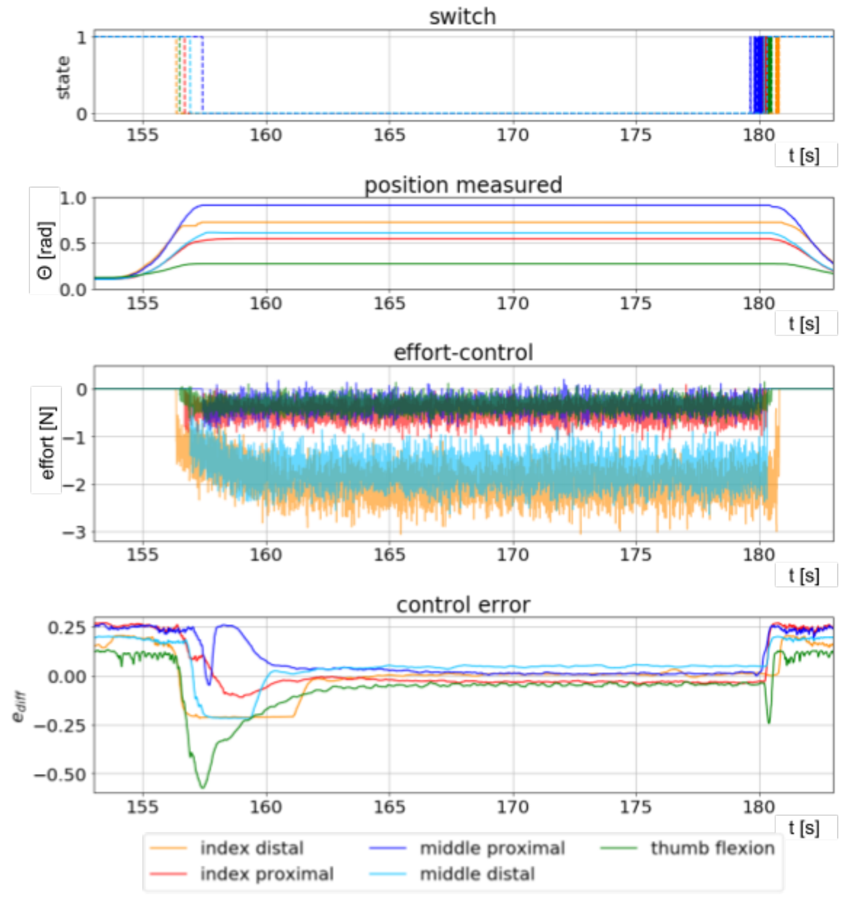


Figure 5.32.: Activation of the effort controller (Tieck et al. [21]). From top to bottom, output of the reflex network *switch*, the *position measured*, the effort feedback *effort-control* and the *control error*.

Adaptive control with online learning evaluation

The network for the classical PI control is larger than the one for the adaptive controller. The parameters for the PI controller are presented in Figure 5.34 (*bottom*). The initial controller parameters are calculated with the Ziegler Nichols method (Ziegler et al. [230]). Then they are manually tuned to the system to avoid noise and oscillations. An offset was added to the controller to compensate for the delay between contact detection and reflex activation. The learning rate and the number of neurons were adjusted with a manual parameter search to tune the adaptive control loop.

With higher learning rates, the system reached the desired efforts faster but also oscillated. This effect was caused by the system's latency, caused through the filters to reduce the noise of the spiking neurons. With the addition of an adaptive I part, the controller is reduced to a P controller, which is easier to parameterize.

With online learning, the controller can adapt and learn over multiple grasps. In Figure 5.33, four consecutive grasps of the same object are shown. The first diagram shows the activation of the control triggered by the contact detection. If a contact is detected, the inhibitory neurons are inhibited (*selective disinhibition* (Sridharan et al. [202])) and the control is activated. The plot *u_correction* is the signal added by the adaptive control to the PI control. The plot *control_error* shows the control error of four joints that are related by synergies – distal and proximal joints of the index and middle finger. In the fourth attempt, the control error is controlled faster than at the beginning. The delay is compensated, and the gripping force is maintained to a constant value. The PI controller needs 5s on average to minimize the control error into a range of 0.05. With the adaptive controller, this effect can be minimized to a delay of 3s, after five grasps on average. As the repetition of a grasp with the robot can not be the same every time, some joints need more grasps to be adapted. The middle finger proximal (dark blue) signal is overshooting in the second and fourth grasps. This effect is caused by the adjustment of the middle finger distal (light blue); nevertheless, after a while, it stabilizes.

SNN implementation and parameters

The SNN is implemented with the neurosimulator Nengo (Bekolay et al. [51]) with LIF spiking neurons. The motor primitives are represented with 22 ensembles and 4400 spiking neurons, based on the modeling of Section 5.3. The reflexes with the classical PI compliant controller are represented with 36 ensembles and 20700 neurons, and with the adaptive compliant controller with 21 ensembles and 7350 neurons. ROS (Quigley et al. [176]) is used as a communication layer. The hand is controlled using the official ROS driver (Heppner [121]). The Schunk SVH 5-finger hand has nine active degrees of freedom — thumb distal, thumb opposition, index distal and proximal, middle distal and proximal, ring distal, pinky distal, and finger spread.

The selected affordances are based on (Cutkosky [78]) with the parameters based on (Scano et al. [190]). The parameters for the finger and hand primitives and the compliant controller are presented in Figure 5.34. In the *top* the corresponding Θ_{min} and Θ_{max} of each joint to define the finger primitives. In the *middle* the corresponding min and max values for the *cylinder, pinch and sphere* primitives — for the *rest* primitive the min and max values are set to zero. In the *bottom* the parameters for the PI, offset, and adaptive P parts for each finger.

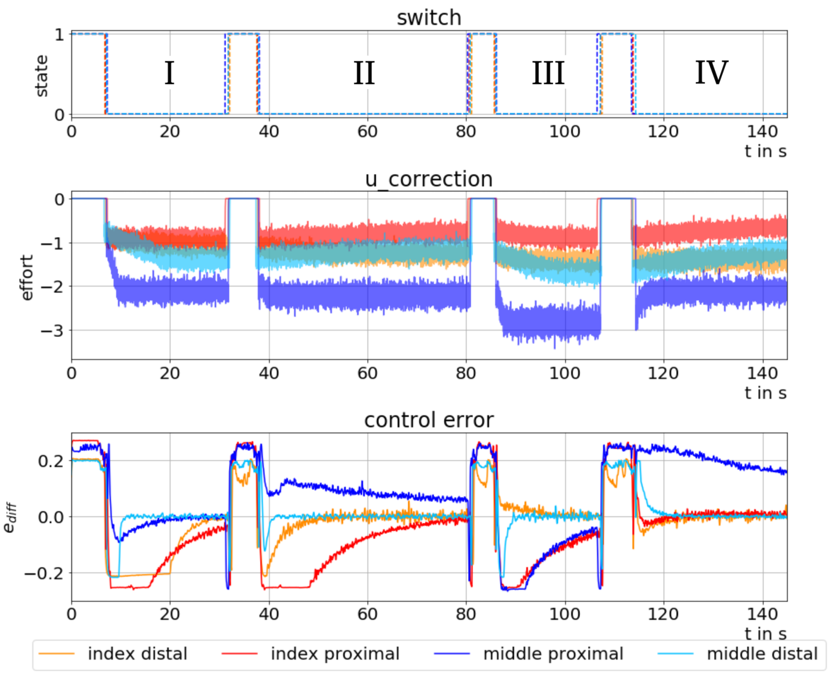


Figure 5.33.: Learning in the adaptive controller after four consecutive grasps (Tieck et al. [21]). *switch* is the activation of the control triggered by contact detection. *u_correction* is the signal added by the adaptive control to the PI control. *control_error* shows the control error of four joints — distal and proximal joints of the index and middle fingers.

Parameters finger primitives:

Finger	proximal		distal	
	Θmin	Θ_{max}	Θmin	Θ_{max}
thumb	0,1	0,9		
thumb opposition	0,25	0,9		
index finger	0,1	0,8	0,1	1,33
middle finger	0,1	0,8	0,1	1,33
ring finger	0,1	0,9		
pinky	0,1	0,9		
finger spread	0,2	0,5		

Parameters hand primitives:

Finger	cylinder		pinch		sphere	
	min	max	min	max	min	max
thumb	0	0,4	0	0,6	0	0,4
thumb opposition	0,5	0,5	1	1	0,2	0,8
index finger	0	0,9	0	0,9	0	0,9
middle finger	0	0,9	1	1	0	0,9
ring finger	0	0,9	1	1	0	0,4
pinky	0	0,9	1	1	0	0,9
finger spread	0,4	0,5	0,3	0,3	0,5	0,5

Parameters PI control:

	thumb	index finger		middle finger		ring finger	pinky
	flexion	distal	prox	prox	distal		
P part:	0.3	0.6	0.2	0.2	0.6	0.6	0.6
I part:	0.1	5	1.5	1.5	5	4.5	4.5
I part with offset:							
factor:	1	1	1	1	1	1	1
offset:	-0.1	-1.1	-0.5	-0.5	-1.1	-0.8	-1.1
adaptive P part:	0.2	1.8	1	1	2.5	2.0	1.3

Figure 5.34.: Parameter tables (Tieck et al. [21]). From top to bottom, the finger primitives, the hand primitives, and the compliant controller parameters.

5.4.3. Discussion

A system was presented using motor primitives organized in a hierarchy of joints, fingers, reflexes, and grasping affordances to perform soft-grasping with an anthropomorphic robot hand. The compliant control is implemented in the same SNN using a cascaded PI effort controller that was extended with online learning for adaptive control. Soft-grasping is mainly made with mechanical features and compliant hardware. Indeed, most robots are not hardware compliant and do not have force sensors (i.e. the Schunk SVH hand). Nevertheless, the experiments showed that with the combined control loops and using the current of the motors, it was possible to perform soft-grasping without calculating the inverse kinematics or complex contact point planning.

The controller adapted the grasping motions to objects with different shapes, stiffness, and sizes without knowing their exact properties. Three main grasping types were modeled — sphere, cylinder, and pinch — but there is no limitation, and the SNN can be extended with more motions. The compliant controller shows high sensitivity, and the threshold could be set as low as to allow the manipulation of balloons (see Figure 5.31). Even with the intrinsic inaccuracy of the current measurements and necessary filters, the efforts could always be maintained below the maximum limits, which means that the compliant controller was actively controlling the joints. The plot in Figure 5.33 shows the online adaption of the controller with every repetition of the same grasping motion.

An adjustment of the network parameters can be made to fine-tune the controller using pre-training with domain randomization in simulation as in Vandesompele et al. [219] and Andrychowicz et al. [45].

5.5. Summary

The modeling of the hand motion as a network that coordinates finger sub-networks with motor primitives simplifies the control. The motion of each finger is modeled independently in a similar way to a robot arm (see Section 4.1). By coordinating the activation of the fingers, it is possible to model grasping affordances. An adaptive behavior for soft-grasping was modeled and implemented using reflex activations to detect contact. Three experiments were presented to model and control the motion of the hand.

The first experiment, towards grasping motions, presented a proof of concept for a biologically inspired SNN control architecture for grasping motions capable of learning and executing different types of grasp motions. The network is capable of learning STDP and execute different types of grasp motions. The grasping motions are represented as synergies between the joints. With the hierarchy of a hand network coordinating finger networks, it is possible to reuse and combine motor primitives (individual finger movements).

The second experiment, triggering finger reflexes, presented a SNN that activates motion reflexes on a robot hand, classifying human EMG data. The network classifies the EMG signals to detect which finger was activated. Based on the classification, single-finger reflexes are triggered. The finger reflexes are modeled with motion primitives and mapped to the robot kinematic. This experiment extends the architecture for grasping motions presented in Section 5.2.

The third experiment, soft-grasping, presented a system based on motor primitives organized in a hierarchy of joints, fingers, reflexes, and grasping affordances to perform soft-grasping with an anthropomorphic robotic hand. The compliant control is implemented in the same SNN using a cascaded PI effort controller extended with online learning for adaptive control. Soft-grasping is often implemented with mechanical features in the robot hardware, but not all robots are hardware compliant and do not have torque sensors as the hand that was used. This experiment extends the architecture and combines all the elements of towards grasping motions presented in Section 5.2 and triggering finger reflexes presented in Section 5.3.

6. Generalization and extension of the motor primitives control architecture for multi-legged locomotion

Locomotion is one of the most diverse and flexible capabilities in nature. There are many studies on the flexibility of multi-legged locomotion in mammals and insects. Specific Spatio-temporal relations between different legs and between the joints have to be considered to generate walking movements (Bässler et al. [50]). In the case of insects, these relations can be modeled with the Cruse rules (Cruse et al. [77]). A biologically-inspired walking system needs to consider the aspects of error tolerance and the capability of self-organization (Cruse et al. [76] and Pfeifer et al. [169]). There are different mathematical models for walking using central pattern generators and proprioceptive sensor feedback (Holmes et al. [126] and Ijspeert [128]). Additionally, the parametrization and generation of the patterns can be modeled as behaviors. The principles for behavior-based networks (Albiez [43]) can be used for behavior-based and adaptive control (Kerscher et al. [136]).

This chapter presents methods and experiments to control different motions for a multi-legged robot driven by motor primitives with SNNs. The modeling of a multi-legged robot follows the same approach outlined in previous chapters, mapping the kinematic structure of the robot to a hierarchy of motor primitives. A multi-legged robot is similar to the hand in terms of kinematic structure but with multiple legs instead of five fingers and using a different coordination layer to generate walking behaviours. The previously presented approaches to represent hand and arm motion using motor primitives are extended to model multiple legs. In the following, a six-legged robot is considered in different walking situations.

Multi-leg locomotion represents the mechanisms to control single-leg movements, coordinate multiple legs, and generate complex walking behaviours. The locomotion problem is defined as the control of the legs together of a robot in a way that enables it to move in different directions effectively. The challenge is how to model the motion of individual legs and how to coordinate them.

6.1. Modeling the motion of a multi-legged robot

The experiments are performed in simulation with the six-legged robot LAURON V (Roennau et al. [180]). The control interface is based on ROS, and the simulation for training is compatible with the NRP. The multi-legged robot has a total of 24 active DoF, 4 on each leg as illustrated in Figure 6.1.

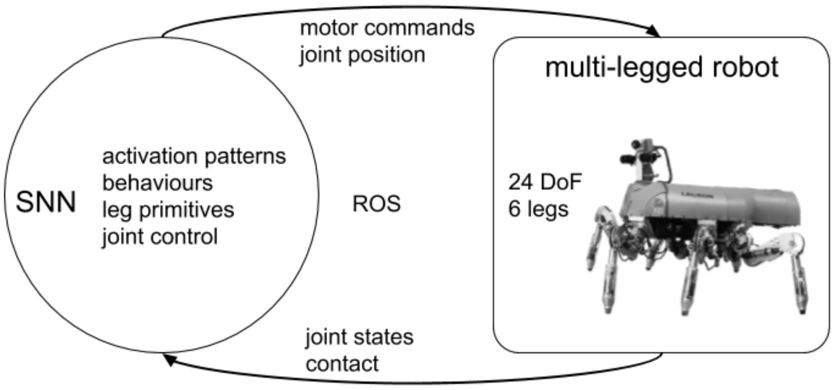


Figure 6.1.: Modeling the motion of multiple legs. The multi-legged robot is controlled using the joint position interface. It has six legs with 24 active DoF, and it has ground contact sensors. Primitives are defined for each leg. A behavior-based architecture is integrated to coordinate the legs and generate different walking patterns.

There is an output population for each of the joints to control a multi-legged robot with SNNs. The legs are modeled similar to the fingers in Section 5.1 mapping the kinematic structure of the robot. There are two primitives for each leg, one for each of the swing and stance phases. The leg trajectories are generated in simulation using the existing controller for the robot by sampling the joint states. Then, the primitive sub-networks for each leg are trained to represent the sequences. In this way, the motion of one leg is similar but independent to all others. The swing and stance phases are modeled as complementary behaviors (Albiez [43]) that alternate with only one active at a time. The legs are coordinated by a behaviour based architecture that generates different activation patterns to control the activation of the leg primitives.

LAURON V (Roennau et al. [180]) is a six-legged robot inspired by the stick insect (see Figure 6.2). The robot is developed for different field tasks, such as mobile manipulation and space exploration missions (Heppner et al. [122]). Because of its robust hardware, adaptable behavior-based control, and numerous sensor systems, the robot is well equipped for inspection and service tasks in rugged terrain and areas inaccessible or dangerous to humans (FZI [106]). The robot has 4 DoF on each leg, and the head can move in two directions for a total of 26 DoF. The control of LAURON V uses *MCA2* (Uhl et al. [216]) with a behaviour-based control (Kerscher et al. [136]) to generate walking gaits and control the posture.

6.2. Synchronization and coordination of motor primitives for multi-legged locomotion

In order to demonstrate the generalization capabilities of the modeling approach with motor primitives, the mechanisms used to represent the motion of the arm (Chapter 4), and the hand (Chapter 5) are combined and extended to model the motion of multiple legs for locomotion. Biologically-inspired robots have attractive hardware solutions, but they often have challenging kinematics, and classical robotics control mechanisms are not always able to take advantage of them (Pfeifer et al. [169]). These systems have high-dimensional configuration spaces caused by their complex morphologies. A human hand, a musculoskeletal or tendon-driven system, and multi-legged robots are good examples.

The problem of robot locomotion is defined as the control of multiple legs to move a body in different directions. The challenge is to model the motion of individual legs and to coordinate them. In this sense, multi-legged locomotion represents the mechanisms to control single-leg movements, coordinate the legs, and generate complex walking behaviours. An illustration of the problem is presented in Figure 6.2.

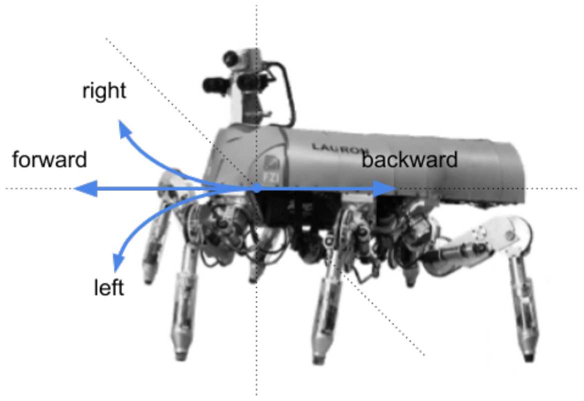


Figure 6.2.: Multi-legged locomotion problem definition. Define leg motor primitives and generate a set of base walking behaviours that can be combined and parameterized — forward, right, left and backward.

This approach is inspired by the biological concepts of motor primitives (Bernstein [54]) as motion building blocks, and the way motion is represented as a hierarchy (Bizzi et al. [55]) that allows reuse and combination of motions. Pfeifer et al. [169] explains how complex behaviors emerge from multiple, parallel, loosely coupled processes that are combined, mainly via the interaction with the environment.

LAURON V (Roennau et al. [181]) is a six-legged robot for space exploration missions. This environment is not predictable, has rugged terrain, and presents unexpected situations that require flexible and adaptive control. The control of LAURON V is based on *MCA2* (Uhl et al. [216]) with a behaviour-based control architecture (Kerscher et al.

[136]) that implements walking gaits and posture control. Behavior-based networks for adaptive control for a bio-inspired robot are introduced in Albiez [43].

In the following, an architecture is presented combining classical behaviour-based control (Kerscher et al. [136] and Albiez [43]) with motor primitives implemented with SNNs. It has a high-level control interface to integrate control signals from other control systems or networks. The main components of the approach are presented in Figure 6.3. This section incorporates findings and methods from Tieck et al. [19].

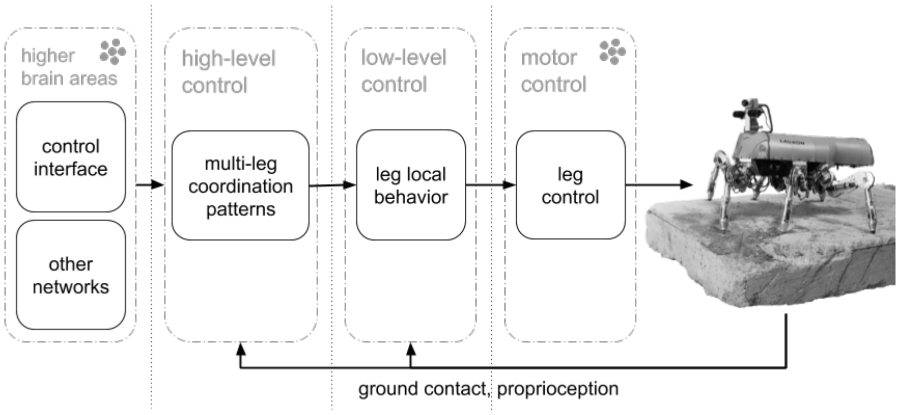


Figure 6.3.: General view of the closed-loop architecture for multi-legged locomotion (Tieck et al. [19]). From right to left the layers have an increasing level of abstraction. In *motor control* there are the individual leg motor control with motor primitives, in *low-level control* the local behaviours for each leg, in *high-level control* the activation patterns for multi-legged coordination and in *higher brain areas* a high level control interface.

A six-legged robot is somehow similar to the hand in its kinematic structure but with six legs instead of five fingers. The modeling of the hand (Section 5.3) is extended with an additional coordination layer for locomotion. The individual motion of the legs is modeled with motor primitives with SNNs. On top of the motor primitives, a behaviour-based control architecture is added to coordinate the legs. By using different control patterns, it is possible to activate the primitives to generate different walking gaits. A high-level control interface enables external control or input from other networks to activate the patterns. Five different experiments in the simulation are presented for the evaluation– walking forward, in a circle, in zig-zag, over an obstacle, and with a Braitenberg network. These experiments show the flexibility of modeling motions using motor primitives with SNNs.

6.2.1. Methods

An approach for locomotion with a multi-legged robot is proposed to model and control different gait patterns by combining SNNs and behaviour-based control. The motions of the legs are generated with motor primitives extending the methods presented in Sections 4.2 and 5.3. A behaviour-based architecture is added on top of the SNN to coordinate the legs and generate different gait patterns. The robot LAURON V (Roennau et al. [180]) is used for the evaluation.

A detailed view of the architecture with the networks expanded for one leg is presented in Figure 6.4. Each oval is a sub-network. The kinematics of a robot hand is similar to LAURON V but with six legs instead of five fingers. Accordingly, the leg modeling is similar to that of a finger but with more active joints. The motion of the legs is synchronized using ground contact. A behaviour-based network is implemented to generate smooth transitions between swing and stance phases, combine the motor primitives and change their parameters. Different activation patterns are used to coordinate the six legs and generate different walking gaits. A high-level control interface is added to input the desired motion direction and integrate control signals from other sources. With this interface, it is possible to model walking forward, in a circle, in zig-zag, over an obstacle and incorporate signals from a Braitenberg network.

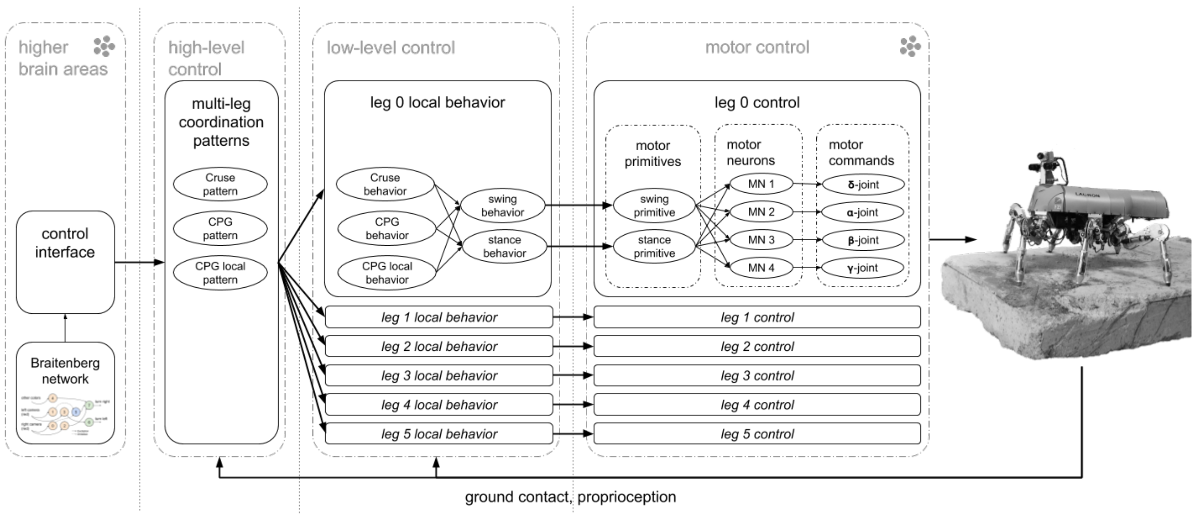


Figure 6.4.: Detailed view of the closed-loop architecture for multi-legged locomotion (Tieck et al. [19]). The networks in *leg control* and *leg local behaviour* are expanded for one leg. The *motor neurons* generate *motor commands*. Two *motor primitives* control the stance and swing phases. The *local behaviors* correspond to the phases and activation patterns, and are activated by the *multi-legged coordination patterns*. A high-level *control interface* allows experiment control and integration of signals from a Braitenberg network.

Leg control with motor primitives

Each leg has four joints — α, β, γ, δ — and the respective kinematic structure is presented in Figure 6.5a. Joint motor commands are used to control the legs. In the simulated model, bumpers sensors are added at the end of each leg to detect ground contact. Contact is used to modulate the changes of phases, inhibit the motion, and adapt the movement. A leg movement cycle has two phases: a swing phase with no ground contact and a stance phase with ground contact (see Figure 6.5b). The α joint is used to move forward and backward. The β joint is used to move down to the ground and lift the legs. The min and max values of the γ and δ joints are the same for both the swing and the stance primitives.

The swing and stance movements are modeled with motor primitives (see Section 3.2). Both primitives control all four joints of a leg. Each joint has a base trajectory defined with an activation function and a minimum and a maximum value. For the α joint, the base trajectory is a sinusoidal function defined between 0 to 1. For the β joint, the function is defined from 0 to 1 and back to 0. Two primitives are defined with inverse trajectories to walk backward by swapping the minimum and maximum values for the α joint.

The SNN for the swing and stance primitives is presented in Figure 6.5c. The inverted reflection value between swing or stance is used as input between 0 and 1 for the neuron population u of the motor primitive. The population $f(u)$ represents the activation function for the motor primitive. The population $g(f(u))$ outputs the motor commands scaled and weighted for the joints [α, β, γ, δ]. The motor commands are forwarded with the output node to the $goal\_out$ node that checks which behavior is active and inhibits the values from the other behavior. For turning left and right, the step width of the inner legs is changed for both the swing and the stance primitives. Activation values change dynamically to generate a smooth transition between phases and to move in different directions.

Leg local behaviours

For each leg there is a swing and a stance behaviour (see Figure 6.6a). The patterns are the actual leg coordination, and there is a corresponding behaviour for each pattern (see Figure 6.4). The formalization and the definition of a single behavior and the interplay between behaviors are taken from Albiez [43]. They are adapted to interact with SNNs. A behaviour is defined as a three-tuple $B = (r, a, F)$, where r is the reflection or target evaluation function, a is the activity function, and F is the transfer function. Additionally, a behavior also receives sensor input $\vec{e}$ and the motivation ι from higher layers. The actuator output $\vec{u}$ of a behaviour is defined as $F(\vec{e}) = \vec{u}$.

The swing and the stance behaviours are complementary, so only one behavior is motivated at a time. The exact mechanism is used to make sure that only one walk-

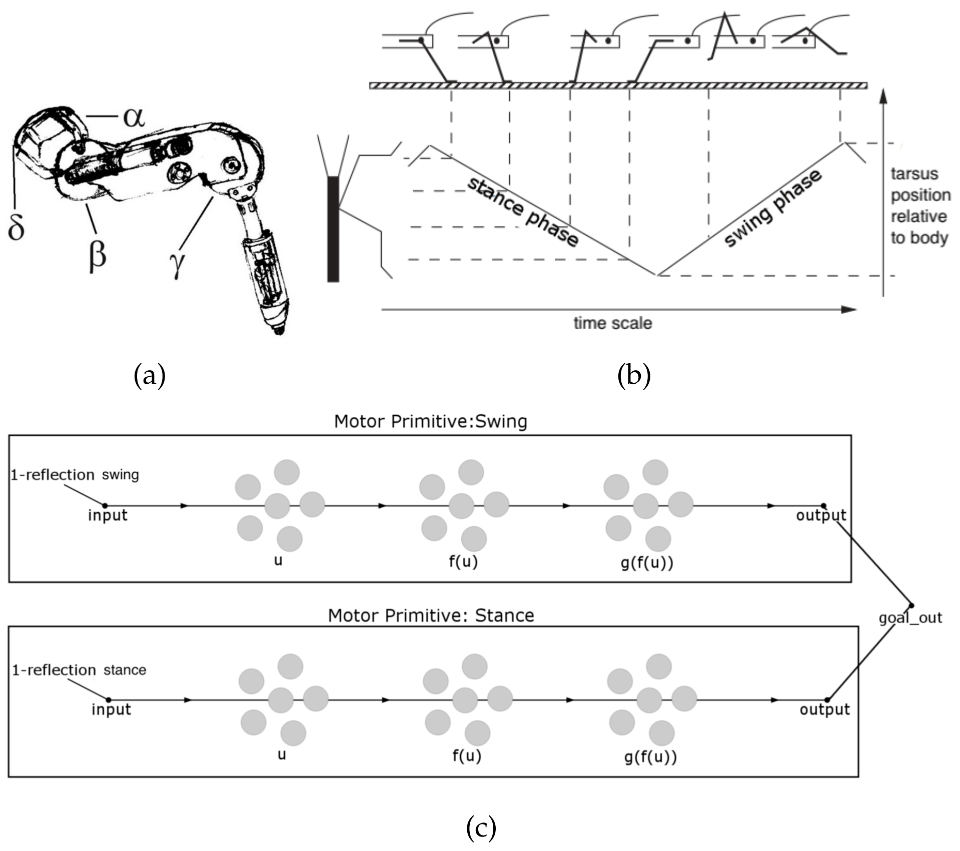


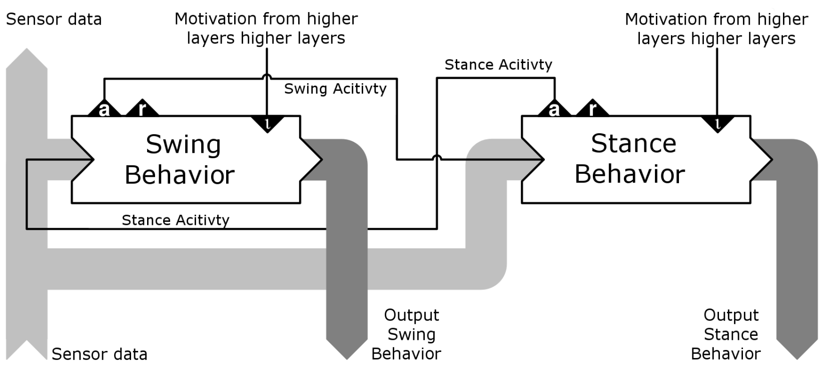
Figure 6.5.: Modeling leg motion with motor primitives (Tieck et al. [19]). (a) Kinematic structure of one leg with four active joints. (b) Main phases of leg motion of stick insects — swing and stance (adapted from Wilson [221]). (c) The SNN for the swing and stance phases.

ing pattern is active at a time. The motivation, activity and reflection of swing and stance are always alternating as shown in Figure 6.6b. The swing and stance behaviors can be influenced from the outside in two ways: by changing the motivation or by changes in the system state represented by the sensor input $\vec{e}$. Both mechanisms can extend or inhibit the activity of a single behavior.

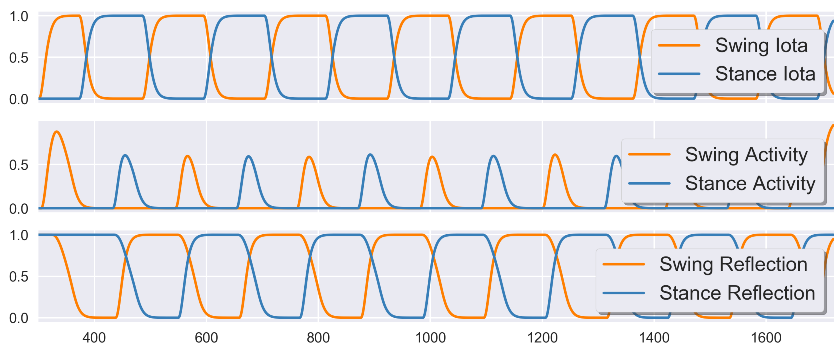
The Cruse, CPG and CPG-Local behaviors in "leg local behavior" (see Figure 6.4) handle the actual leg coordination and activate the underlying swing and stance behaviors that activate the motor primitives. These behaviours are motivated by the corresponding patterns in "multi-legged coordination patterns".

Multi-legged coordination patterns

The patterns coordinate the legs, determine which leg is in the swing or stance phase, and motivate the corresponding behavior. The patterns are also implemented as com-



(a)



(b)

Figure 6.6.: Leg local behaviors (Tieck et al. [19]). (a) Modeling of the swing and stance phases as complementary behaviours (adapted from Albiez [43]). (b) Complementary behaviour of swing (*orange*) and stance (*blue*). From top to bottom, the plots show the motivation, the activity, and the reflection, alternating because only one behavior is active at a time.

plementary behaviors so that only one pattern is active at a time. A pattern class controls the motivation, activity, and reflection for each pattern. The pattern class also detects if the robot is walking if a direction is given and a pattern is active. The motor commands are only sent to the joints only if the robot is walking. Two leg groups are defined for a tripod gait — group 0 with legs $[0, 3, 4]$; and group 1 with legs $[1, 2, 5]$ (see Figure 6.7b. For the tripod gait, the swing and the stance phase will alternate for each leg group.

The Cruse pattern (see Figure 6.7a) implements the first three Cruse rules (Cruse et al. [77]) using the states of the legs to evaluate the rules and switch between phases. The swing phase of one leg inhibits the start of the swing phase of the next leg. The start of the stance phase excites the start of the swing phase of the next leg. The position of the previous leg excites the start of the stance phase. The Cruse rules support the stability of the tripod walk and adaptation to the ground.

The CPG pattern (see Figure 6.7b) implements tripod walking and synchronizes the activity of both leg groups. This pattern uses the reflections of both groups' underlying swing and stance behaviors, and they wait for each other to be finished with the swing or stance phases.

The CPG-Local pattern (see Figure 6.7c) also implements tripod walking but with no outer synchronization mechanism. This pattern does not consider in which state the other legs are and controls each leg independently. In the beginning, a leg group will be motivated to swing and the other to stance, and after that, each leg will take into account its swing and stance reflection to determine if it will swing or stance next. Since no information from the other legs is used, the generated pattern will be asynchronous after some time, and the generated gait is unstable.

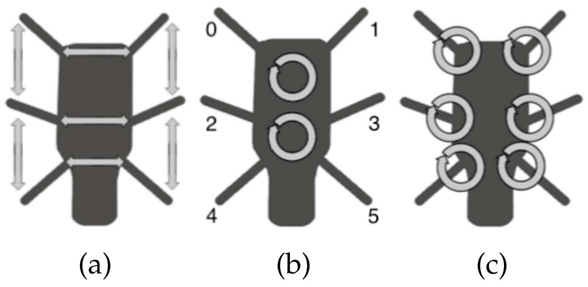


Figure 6.7.: Multi-legged coordination patterns (Tieck et al. [19]). Schematics to visualize the different activation patterns: (a) Cruse rules, (b) central CPGs and leg numbering, and (c) local CPGs.

Control interface and Braitenberg network

A high-level control interface is added to integrate control signals from other sources. The control interface provides high-level control signals for each movement — turn left, turn right, walk forward, and backward. The experiments presented are implemented by using this interface in the NRP, either sending commands from the experiment control state machine or integrating another network.

To generate a complete goal-oriented behaviour, a Braitenberg vehicle (Braitenberg [58]) implemented with SNNs is integrated to control the pattern activation layer. The sensor input of a Braitenberg vehicle controls the motion directly. Different behaviors can be generated depending on how the sensors are connected to the actuators. For the experiments, the Braitenberg vehicle spins around until the color red is detected, and when red is detected, it walks straight towards it (see Figure 6.8).

6.2.2. Experiments

Five different experiments were defined in simulation using the NRP — walking forward, in a circle, in zig-zag, over an obstacle, and with a Braitenberg network. All the

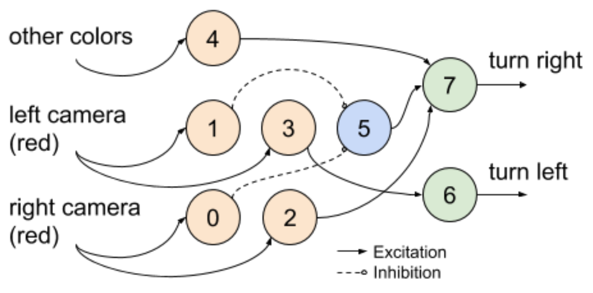


Figure 6.8.: A Braitenberg vehicle with attractive behaviour implemented with SNN (Falotico et al. [2]). Input neurons $[0, 1, 2, 3]$ (*orange*) detect red. Neuron 4 is a bias, using other colors as input to rotate. When red is detected, neuron 5 (*blue*) inhibits the bias and both output neurons (*green*) are activated.

experiments were conducted with each walking pattern — Cruse, CPG, CPG-Local.

General experiment setup

The general setup is an experiment in the NRP with the robot model of LAURON V. For the experiment control, the state machine sends the desired walking direction to the control interface in the first four experiments. In the fifth experiment, the Braitenberg network sends the desired walking direction directly to the control interface. The simulation time is used as a reference to compare different experiments.

The SNN control architecture is implemented in Nengo (Bekolay et al. [51]) and it is based ROS as communication interface with the NRP. The *motor control* layer and the *Braitenberg network* from Figure 6.4 are implemented with SNN. The parameters that adapt the motor primitives for the different motions are presented in Figure 6.9. The following aspects were evaluated: the taken path, the generated walking gait, the distance covered in the given time, and the differences among walking patterns. For the experiments, the following variables were recorded: the real-time timestamp, the trajectory of the movement of LAURON V, the motivation signal, the activity, and reflection of the pattern. Additionally, for each leg, it was recorded: the ground contact, the generated joint values in swing and stance phases, the motivation signal, the activity and reflection of each behavior, the direction, and the active phase.

Walking forward

The first experiment is to walk forward in a straight line in an empty world with flat ground. The experiment setup is presented in Figure 6.10a. The experiment control sends the direction "walk forward" for 40 seconds. This experiment serves as a baseline to compare the three walking patterns in a simple scenario.

		Swing Primitive				Stance Primitive					
		min		max		min		max			
	side	alpha	beta	alpha	beta	alpha	beta	alpha	beta	gamma	delta
Left	0	-0.125	1.5	0.125	2.0	0.125	1.5	-0.125	1.0	-2.0	0.0
	1	0.25	1.5	-0.25	2.0	-0.25	1.5	0.25	1.0	-2.0	0.0
Right	0	-0.25	1.5	0.25	2.0	0.25	1.5	-0.25	1.0	-2.0	0.0
	1	-0.125	1.5	0.125	2.0	0.125	1.5	-0.125	1.0	-2.0	0.0
Forward	0	-0.25	1.5	0.25	2.0	0.25	1.5	-0.25	1.0	-2.0	0.0
	1	0.25	1.5	-0.25	2.0	-0.25	1.5	0.25	1.0	-2.0	0.0
Backward	0	0.25	1.5	-0.25	2.0	-0.25	1.5	0.25	1.0	-2.0	0.0
	1	-0.25	1.5	0.25	2.0	0.25	1.5	-0.25	1.0	-2.0	0.0

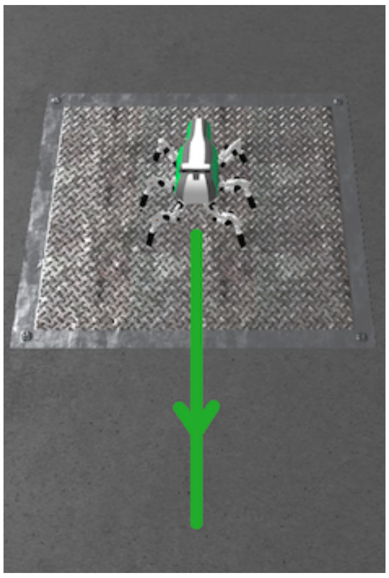
Figure 6.9.: Table with a summary of the experiment parameters (Tieck et al. [19]). The sides of the robot are 0 for left and 1 for right. The values (*min*, *max*) of the joints ($\alpha, \beta, \gamma, \delta$) for each primitive (*swing*, *stance*) are changed for the different motions (*left*, *right*, *forward*, *backward*).

The resulting path for each walking pattern is shown in Figure 6.10c, each line represents 40 seconds. The x and the y axes are the coordinates of LAURON V in the world reference frame of the simulation. At the same time, the CPG pattern was the fastest, covering about $2.5m$, the CPG-Local covered about $1.9m$, and the Cruse pattern was the slowest covering about $1.5m$. Figure 6.10b shows on top the gait diagram and on the bottom the values of the β joint of the Cruse pattern. The general shape of the α and β join trajectories are very similar for all patterns. The β joint trajectory is highlighted to indicate that the leg is swinging, and it is complementary to the α joint. Since the robot is walking in a straight line and the min and the max values of the alpha trajectories are not changing, the amplitude does not change. The tripod gait can be recognized in the beta trajectories and the gait diagram as three legs are always in the swing phase, and the other three legs are in the stance phase.

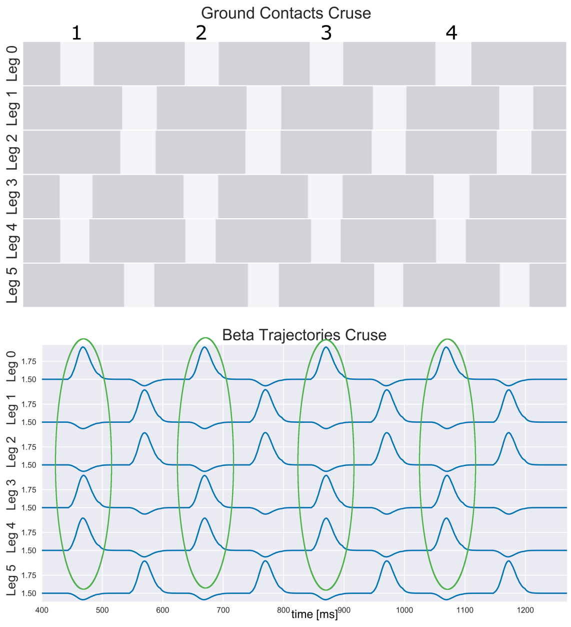
Walking in circle

The second experiment is to walk in circles in an empty world with flat ground. The experiment setup is presented in Figure 6.11a. The experiment control sends the direction "turn right" for 3 minutes. After the robot walked a complete circle, the experiment is stopped, and the collected data is analyzed.

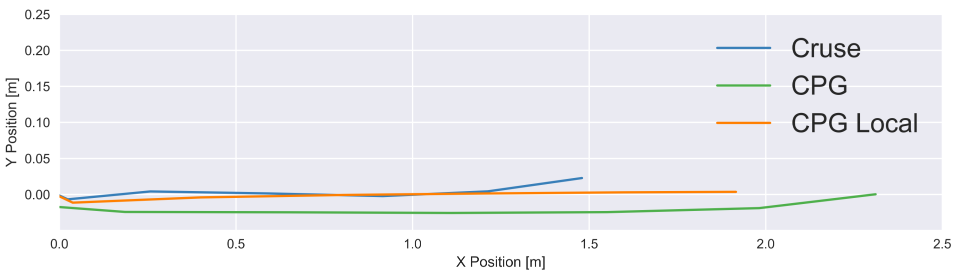
The resulting path for each walking pattern is shown in Figure 6.11b, each line represents 3 minutes of movement. The x and the y axes are the coordinates of LAURON V in the world reference frame of the simulation. The ideal path was walking on a circle turning to the right. The radius of the circle can be changed by modifying the parameters of the behavior.



(a)



(b)

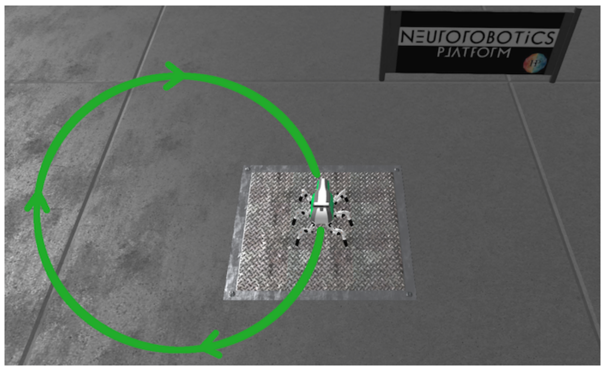


(c)

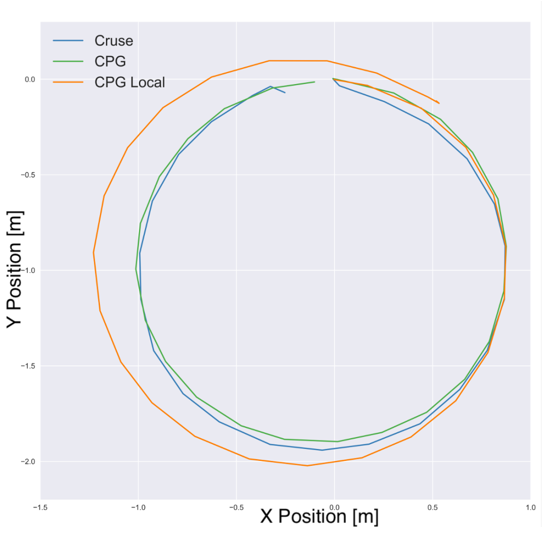
Figure 6.10.: Experiment 1: walking forward (Tieck et al. [19]). (a) Simulation setup. (b) On top the gait diagram and on the bottom the values of the β joint of the Cruse pattern. (c) The resulting path for each walking pattern. The different trajectories are caused by the execution time of the patterns.

Walking in zig-zag

The third experiment combines the two previous experiments and includes moving left and backward in an empty world with flat ground. LAURON V has to walk in zig-zag: which is defined as walk forward, then turn left, then walk backward, and finally turn right. The experiment setup is presented in Figure 6.12a. The experiment control will trigger the different directions — walk forward, turn left, walk backward and turn right — each for a specific time to create a zig-zag path. Overall, the simulation time is 2 minutes and 30 seconds. In this complex experiment, all directions can be tested. It can be observed how each pattern manages the changes in direction and if the transitions between the directions are smooth.



(a)



(b)

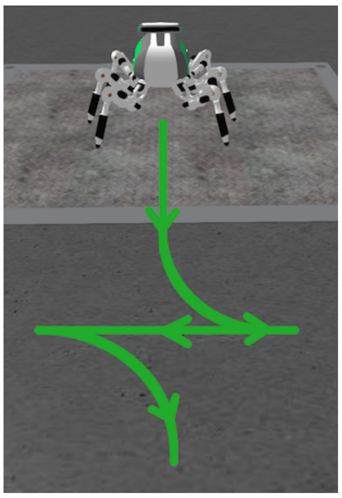
Figure 6.11.: Experiment 2: walking in circle (Tieck et al. [19]). (a) Simulation setup. (b) The resulting path for each walking pattern. The different leg coordination in the patterns is evident in the trajectories while turning.

The resulting path for each walking pattern is shown in Figure 6.12b, each line represents 2 minutes and 30 seconds of movement. The x and the y axes are the coordinates of LAURON V in the world reference frame of the simulation. The CPG pattern was able to start the right turn earlier because of the leg positioning. Since the CPG pattern is faster for walking forward, the front right leg was already in front when the turn came, and therefore it could be taken faster. For the CPG-Local and the Cruse patterns, which are slower, the left front leg was in front when the turn was started.

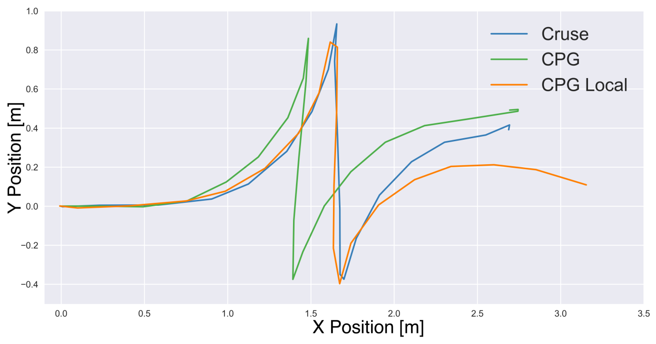
Walking over an obstacle

The fourth experiment is to walk forward and overcome an obstacle. In this experiment, an obstacle is placed in the walking path of the robot. A $5cm$ high box was added to the experiment of walking straight. The experiment setup is presented in Figure 6.13a. The physical properties of the box are defined to keep it static, even if the robot hits or steps on it. Accordingly, the friction between the box and the ground was changed to be very high. Since the real robot was designed for exploration, this experiment will test how the implemented patterns respond to uneven terrain. The swing motion has to be stopped when ground contact is detected to overcome an obstacle, and the stance motion has to be extended. The experiment control sends the direction "walk forward" for 40 seconds.

A frame sequence from the experiment of CPG pattern is presented in Figure 6.13c. Figure 6.13b shows on top the gait diagram and on the bottom the values of the α and



(a)



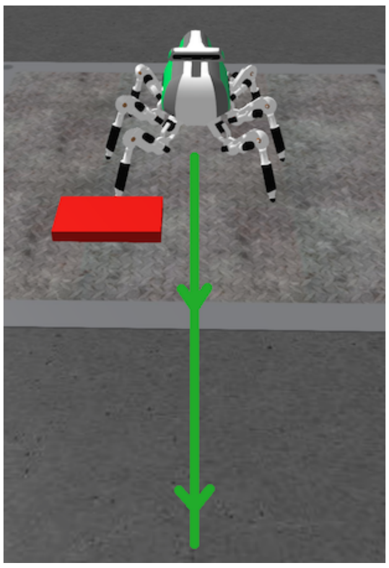
(b)

Figure 6.12.: Experiment 3: doing a zig-zag (Tieck et al. [19]). (a) Simulation setup. The zig-zag path is defined as walk forward, turn left, walk backward and turn right. (b) The resulting path for each walking pattern.

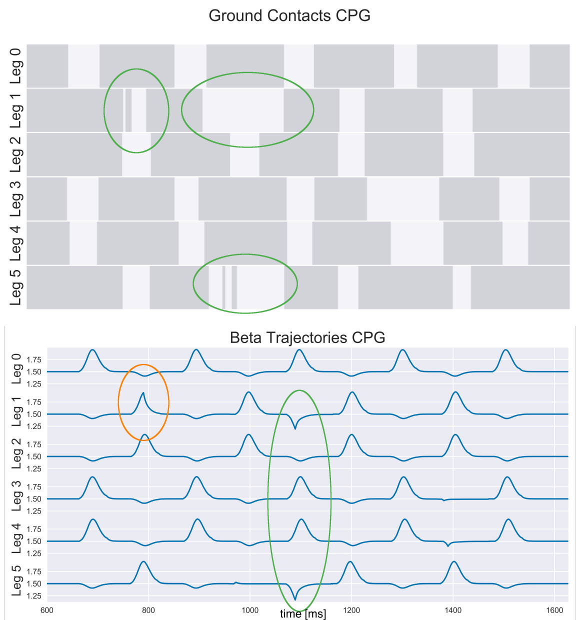
β joints of the CPG pattern. The darker bars indicate ground contact in the stance phase and the brighter ones air time of the swing phase. Similar to experiment 1: Walking forward, the CPG pattern was faster than the CPG-Local pattern. The first circle shows that leg 1 hits the obstacle before moving over it. This behavior is undesired because the leg should not be sliding over the obstacle like that. It is caused by the simulation not being accurate and the models not being perfectly calibrated. The following two circles for leg 1 and leg 5 show when those legs were in the air because LAURON V stepped on the obstacle. However, the diagrams show that the CPG pattern can re-establish ground contact after being on top of the obstacle — this was also the case for the CPG-Local. The Cruse pattern (a) cannot re-establish contact, and it is stuck on the obstacle with legs 1 and 5 in the air. The circles mark the interesting parts of the trajectories. The orange circles mark the unwanted behavior of leg 1 sliding along the obstacle. The green circle marks the stance phases with the changed values to re-establish ground contact.

Braitenberg vehicle with color detection

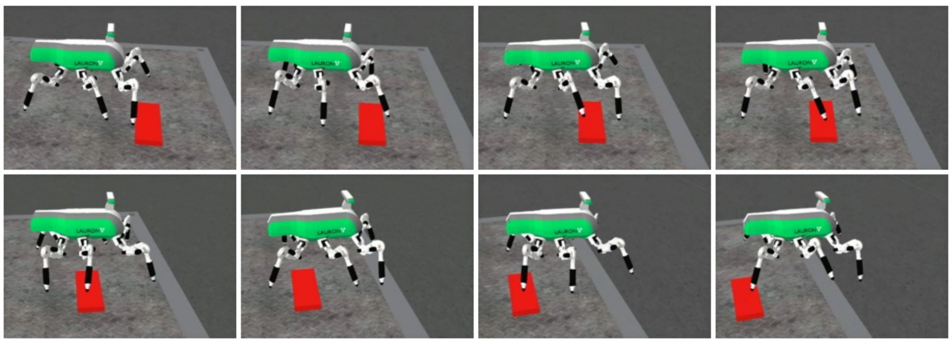
In the fifth experiment, LAURON V is placed inside a virtual room with two displays (see Figure 6.14a). The displays are blue, and after 100 seconds, the experiment control changes the colors to red. After 140 seconds of simulation, the experiment is stopped. With this experiment, the high-level control interface is tested by interfacing it with a Braitenberg SNN implemented in NEST in the NRP. To start, both screens are blue, the Braitenberg network controls the robot to walk in a circle, and once the red color is detected, it sends the direction "walk forward".



(a)



(b)



(c)

Figure 6.13.: Experiment 4: walking over obstacles (Tieck et al. [19]). (a) Simulation setup. (b) On top, the gait diagram and on the bottom the values of the β joint of the CPG pattern. The contact with the object can be recognized in both. (c) A frame sequence from the experiment with the CPG pattern.

The path for each walking pattern is shown in Figure 6.14b, each line represents 2 minutes and 20 seconds of movement. The x and y axes are the robot's coordinates in the world reference frame of the simulation. The stars indicate when the display switched red, and the arrows indicate when the direction switched to forward. The displays are marked as red horizontal bars on the edges. The path of the Cruse and the CPG pattern are almost identical. As soon as the displays switched red, both started to move forward as the display was visible for the robot. The CPG-Local pattern is faster than the other two, and therefore, the robot already passed the first display when it switched red. The experiment was stopped after 140 seconds, and this time is not enough for the CPG-Local pattern to turn the robot towards the display.

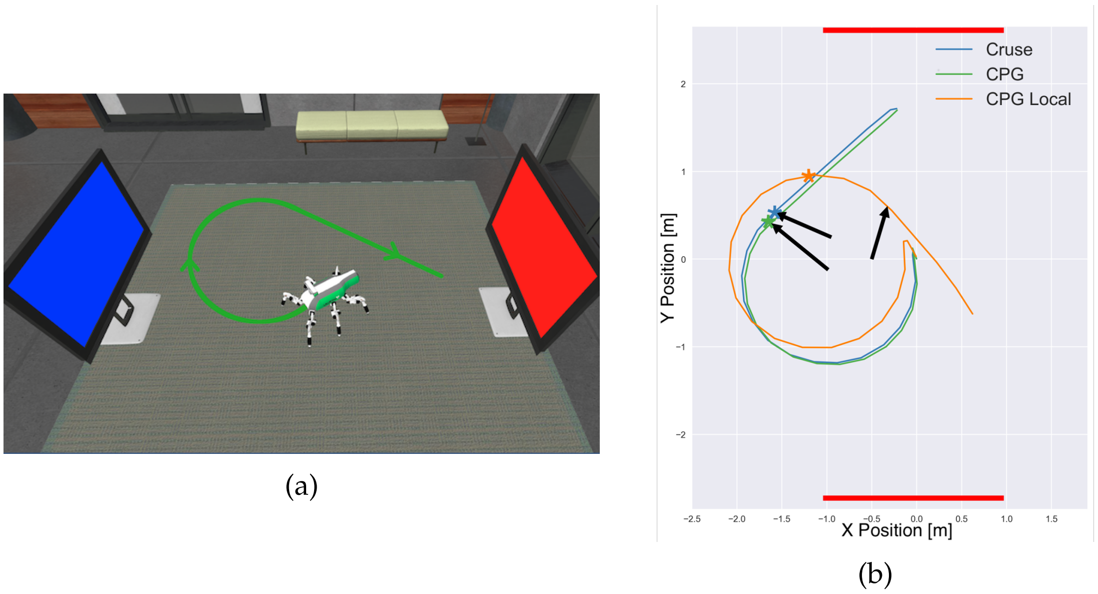


Figure 6.14.: Experiment 5: a Braitenberg vehicle with color detection (Tieck et al. [19]). (a) Simulation setup. (b) The resulting path for each walking pattern. The stars mark when the displays are switched red, and the arrows mark when the direction is changed to forward. The red bars represent the screens.

6.2.3. Discussion

The main contribution of this work is to show that classical behaviour-based control can be combined with motor primitives implemented with SNNs for motion representation. This approach provides flexibility for multi-legged locomotion because motions can be reused and combined in different ways, which can be seen with the different walking behaviours and implemented experiments. This approach shows that the basic SNNs building blocks for motion modeling can be adapted to different robot morphologies like arms, hands, and legs.

The proposed network was able to control a six-legged robot (LAURON V) with SNNs for locomotion behaviors in a simulated environment to walk in all directions — forward in a straight line, backward, left, right. Individual leg motion was modeled with motor primitives to realize the swing and stance phase using SNNs. Multi-legged coordination with different activation patterns using the Cruse rules, CPG and CPG-Local, is implemented as a behaviour-based network. A complete evaluation of the different behaviors was performed, and a high-level control interface to integrate control signals from other sources was developed. Everything is implemented and developed as experiments within the NRP.

For future work, it would be interesting to improve the contact detection and model calibration in the simulation. This facilitates transfer learning from simulation to real robots. It is fundamental to work with realistic and calibrated parameters for inertia and friction to avoid sliding and simulation glitches. It would be interesting to integrate additional sensory feedback. Until now, only the bumper sensor for ground contact detection was used. Also, it would be beneficial to refine the transitions during changes in direction to be very smooth. Other sensor information could be used to have better synchronization between the behaviors. The approach could also be extended to robots with unknown body parameters and the generation of stable control and walking patterns for them (Buettner et al. [62]).

6.3. Summary

The modeling of the leg is similar to that of a finger in Section 5.1, but it uses more active joints. Locomotion is modeled similar to the hand to coordinate the legs similar to affordances. The walking behaviours are modeled with rhythmic or repetitive activations and controlled by different pattern generators.

The experiment in leg locomotion presented a system that combines classical behaviour-based control with motor primitives implemented with SNN. This approach provides flexibility by using motor primitives as motions can be reused and combined in different ways. Different experiments with different walking behaviours and environments were implemented. Additionally, the system can be used on different robots by learning the base low-level motor primitives and implementing a new behavior. The experiment shows that the modeling approaches with motor primitives for the arm (Chapter 4) and the hand (Chapter 5) can be generalized and extended for other motor control tasks with different robot morphologies.

7. Discussion of results and outlook

This chapter provides the final considerations for this thesis. A summary of the achieved results, contributions, and insights is presented, followed by the open problems and follow-up directions to this work. Finally, the outlook presents a discussion about the challenges of using SNNs for neurorobotics and the role of neurorobotics for neuroscience.

7.1. Summary of the contributions

The main goal of this thesis is to design and implement SNN-based motion control schemas for manipulation and grasping tasks. The approach is based on motor primitives implemented with SNNs.

The experiments for manipulation showed how a human-sized robot arm could be controlled using motor primitives with SNNs. An important aspect is demonstrating how to combine and parameterize motor primitives using different activation modalities to generate waving and reaching motions as a benchmark for SNNs. Combining a base motor primitive with a set of correction motor primitives for open-loop control allows the generation of pointing motions to different targets on a given plane in 3D space. Target reaching of arbitrary points in 3D space is achieved by using a set of motor primitives driven by the target's position as error feedback in closed-loop.

The experiments for grasping showed how an anthropomorphic 5-finger robot hand could be controlled using motor primitives with SNNs. Grasping motions are modeled by adding an extra layer of motor primitives to coordinate the finger motor primitives. The generation and execution of finger motor primitives as a reflex triggered by human EMG signal classification is discussed. The adaption of grasping motions to different objects is modeled using a cascaded force-torque controller to perform soft grasping.

This thesis proposes principles for model free-motion representation exploring the potential of SNNs. The methods do not require extensive learning, calculating the IK and validating configurations for planning. In the following, a summary of the contributions and limitations of this thesis is presented.

- **Motions are modeled with brain-inspired mechanisms using SNNs**: In biology, motor primitives represent the synergies between different muscles during a motion (d'Avella et al. [84] and Chinellato et al. [68]). This thesis proposes a generic modeling approach applying spiking motor primitives as building blocks. In biology, motion is represented in a hierarchical and distributed way in the body and the nervous system (Pfeifer et al. [169]). In this thesis, spiking motor primitives are arranged in different layers in a hierarchy to model complex motions and behaviours, allowing the reuse and combination of different motions. The combination is achieved using multimodal activations, using perceptual information about the target, or defining motor primitives to coordinate other motor primitives. In the brain, the cortex has a topological sensor and motor representation that match the body (Penfield et al. [168] and Lorente de No [148]). In this thesis, the topology of the proposed SNNs reflects the robot's kinematic structure as demonstrated with a robot arm, a 5-finger anthropomorphic hand, and a multi-legged robot. Studies have shown that the activity in the motor cortex as a whole presents a brief but strong rotational component that can be understood as a dynamical system that drives motion, producing motor activity (Churchland et al. [70] and Russo et al. [184]). This thesis presents different mechanisms for motion activation by modeling a motion generation layer that produces rotational neural activity with a relatively constant and normalized amount of spikes over time. There are circuits of interneurons in the spinal cord combining inhibitory and excitatory connections that activate different reflexes (Jankowska [131]). In this thesis, a contact detection circuit is modeled as an alternative selection mechanism with inter-neurons considering the force-torque feedback and the changes in velocity in the joint.

- **Motor primitives can be combined to generate complex motions**: There are two mechanisms to generate complex motions using motor primitives. The first mechanism is to have different activation modalities to activate the motor primitives. Examples of this are voluntary, rhythmic, and reflex activations to activate the same motor primitive and generate complex waving and reaching motions. In one experiment, a target representation signal was used to activate a base motor primitive and different correction motor primitives to point at different targets on a plane in 3D space. In another experiment, a target representation signal was decomposed to activate three motor primitives to reach different targets in 3D space. The second mechanism is to have a motor primitive activating multiple motor primitives at the same time. In one experiment, motor primitives were defined for different affordances to coordinate the motion of the fingers. In another experiment, various walking behaviours act as motor primitives to activate different legs at a time to obtain the tripod gait. Additionally, the activation of motor primitives can also be parameterized in terms of speed and amplitude. This parameterization was used to change the arm-waving motion and applied to change the relative amplitude of the motion of the legs on one side of the robot to make it turn or change the gait. An important point to mention is that the precision of the SNN increases with the number of neurons. This fact implies that a balance needs to be made between the application requirements and the computing resources available.

- **Motions can be activated and adapted based on sensor feedback**: Sensor feedback is integrated with different ways to drive the activation of motor primitives. For the arm, a contact detection intercircuit is used to trigger a retract reflex. A discrete feedback signal based on the target error was incorporated to activate the motor primitives and move the arm around in analogy to visual-servoing. For the hand, contact sensors in the fingers were tested in simulation to stop the motion of the fingers, and the joint states were used to generate the motor commands. A contact detection intercircuit stops the motion of the fingers as a reflex and triggers the compliant control for soft-grasping. The controller processes the joint states feedback and the current from the motors for closed-loop compliant control without a force-torque control interface to the robot and without compliant hardware. For the legs, the joint states and ground contact sensors were used in simulation to couple the individual motion of the legs with the ground contact. The experiments were limited to the available sensors and interfaces in the robots. The use of haptic and force-torque sensors can further enhance the activation modalities and feedback representations.

- **The motions of multiple kinematic chains can be coordinated**: The experiments with a 5-finger anthropomorphic hand and a multi-legged robot showed different mechanisms to coordinate multiple kinematic chains. For the hand, a set of motor primitives were defined to represent different affordances and coordinate the fingers. The output of these motor primitives is not mapped to motor commands for the joints; instead, it is mapped to the activation parameters of the finger motor primitives. These motor primitives create a higher level in the hierarchy to model grasp motions — sphere, pinch, and cylinder affordances. A non-spiking behaviour-based control architecture is added on top of the motor primitives to coordinate the six legs of the robot. Different behaviours model different activation patterns by using CPG based control or applying the Cruse rules (Cruse et al. [77]). With this architecture, it is possible to change the speed of the legs in one side of the robot to turn or to define a gait pattern.The trade-off of this approach is that by adding more kinematic chains to the system, the model becomes more complex, and the runtime of the SNN is affected. In this sense, the use of neuromorphic hardware can solve this issue.

- **The SNNs approach can be adapted to different robots and combined with other control systems**: The experiments demonstrate that the modeling principles with SNNs can be applied to control industrial robot arms, anthropomorphic 5-finger hands, and multi-legged robots. The experiments also showed that the SNNs could be integrated with other non-spiking control systems or combined with other SNNs. The proposed approach has a generic control interface based on ROS to communicate with different robot drivers and interact with the simulators, such as Gazebo and the NRP. One experiment introduced a sub-network to classify EMG signals to trigger reflexes and activate the motor primitives of the fingers. Integrating human signals opens up new possibilities because prosthetic applications benefit from low-power hardware implementations such as neuromorphic hardware. In another experiment, a classical controller was integrated to move an arm and place the hand close to the objects to perform soft-grasping with SNNs. In a further experiment, the individual leg motion was implemented with a SNN. On top of it, a non-spiking behaviour based architecture was integrated for the activation patterns. On top of it, another SNN of a Braitenberg vehicle was integrated to provide high-level control signals. These experiments show the flexibility of the SNN-based modeling approach for motion representation using motor primitives. The approach can also be used with different robot arms and hands by redefining the mapping of the motor primitives to the robot kinematics. However, the integration of other non-spiking control systems requires additional encoding and decoding steps for the input and output signals of the SNN, which adds further delays to the control loop. Besides, the control interfaces for the robots are not designed for event-based computation, and the tools and interfaces for SNNs are not optimized for robot control. This factor requires additional computation, which yields further delays in communication.

- **Learning**: The methods presented in this thesis are based on two learning strategies, one to generate and pre-train the network offline in a supervised way and one to adapt the weights online using an error signal. To generate the SNN models, the methods from the NEF (Eliasmith et al. [99]) are applied with the software package Nengo (Bekolay et al. [51]). This software allows the generation of large-scale SNNs by breaking the networks down into smaller parts. The connection weights for each sub-part are optimized separately, and then they are combined into one large neural network. Performing this optimization (i.e., finding connection weights) locally means that large systems can be generated without following the traditional neural network approach of optimizing over huge amounts of training data. However, the trade-off is that explicit claims must be made about what each sub-part of the model is doing, which requires expert knowledge. For a robot arm, the motions are defined as functions or as teach-in examples. For the robot hand, the motions are defined from human demonstration or as teach-in examples. For the multi-legged robot, the individual leg motor primitives are defined as functions. For online learning, the PES rule is used. PES was first introduced in MacNeil et al. [153]. This learning rule modifies the decoder of a connection to minimize an error signal. A mechanism using PES was presented to adapt an existing motor primitive using an error signal based on the desired new trajectory. Besides this, the adaptive PI controller of the fingers used PES to adapt the output of the controller and learn over multiple grasping attempts. This mechanism compensates for differences in the dynamics of unknown objects. An essential characteristic of this approach is that the SNN requires only one example or mathematical description of each motion to train the motor primitives. After learning, the SNN adapts the motion to the different objects. This capability is a real advantage compared to other biologically-inspired grasping and manipulation approaches based on DL (Starke et al. [204] and Andrychowicz et al. [45]), because training data and large datasets are expensive and not easily generated with real robots. The training of a single DL model on a GPU produces more CO2 than an average car during its whole lifetime (Strubell et al. [208]).

7.2. Open problems

The open problems and future research to this thesis can be summarized in two categories: integration of visual information and efficient SNN execution with neuromorphic hardware.

7.2.1. Integration of visual information

In order to incorporate visual information, event-based cameras (Gallego et al. [107] and Steffen et al. [205]) are a natural match for SNN. A visuomotor control loop can be achieved using visual signals as feedback or as a trigger to activate the motor primitives. The methods proposed in this thesis for arm and hand control with SNNs respond to the question, *how?* Visual information can be used to respond to the questions *where?* and *what?* — where to move the arm, what to grasp. An initial setup of such a system is presented in Figure 7.1 (Kaiser et al. [3]). A setup with event-based stereo cameras is mounted on a robot head performing microsaccadic eye movements to perceive the objects (Kaiser et al. [10]). This setup can determine target points is 3D space (Steffen et al. [206]) for reaching. Microsaccades (Kaiser et al. [3]) can be used to detect and classify the type of object and identify which grasping affordance to use. Another aspect is the integration of visual input to drive the motions and to learn tasks by demonstration (Reithler et al. [177]). The flexibility of the proposed hierarchy of SNNs allows the integration of visual feedback from event-based cameras for motor activation (Kaiser et al. [9]). A promising strategy for efficient learning of new motions by demonstration can be achieved by minimizing the visual prediction error (Kaiser et al. [8]). Visual information can also be used to explore new motions and learn the direct and inverse dynamics models (Gilra et al. [111, 112]).

7.2.2. Efficient execution of SNNs with neuromorphic hardware

Brain-inspired technologies attract interest from research, engineering, and the industry, especially neuromorphic hardware and event-based computation. Leading

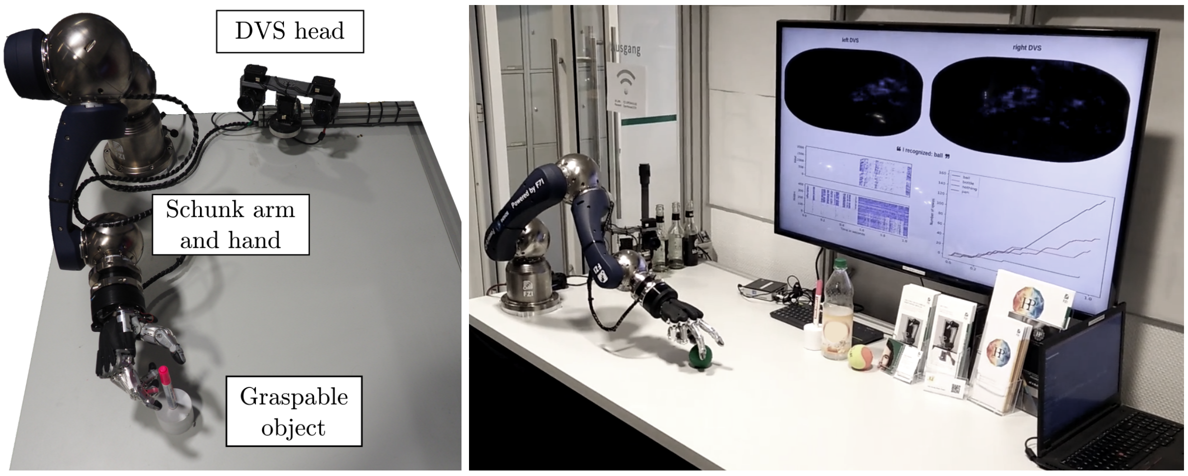


Figure 7.1.: Visuomotor integration on a real robot demonstrator for arm and hand motion with event-based cameras for embodied learning using SNN (Kaiser et al. [3]).

chip manufactures are entering this field, for example, Loihi from Intel (Davies et al. [85]) and Truenorth from IBM (Akopyan et al. [42]). They are following the developments of SpiNNaker from the universities of Manchester and Dresden (Furber et al. [104] and Höppner et al. [127]) and Brainscales from the university of Heidelberg (Pfeil et al. [171]) from the HBP. Nevertheless, there is only a small number of industrial applications for neuromorphic hardware. New paradigms are required to program SNNs to take advantage of the efficient real-time execution of event-based computation (Rhodes et al. [178]). Spike-based communication enables hardware optimizations that allow low energy consumption, and due to the sparse communication, it can also speed up the computation. Neuromorphic hardware is not yet mature (Knight et al. [139]), but the advantage of low energy consumption is of particular interest for mobile robots. In this sense, robotics can take advantage of the characteristics of SNNs and neuromorphic hardware to embed and execute SNNs directly on the robot. Neuromorphic hardware can be used to directly process the spike activity of the network to control motors (Donati et al. [96]) or to integrate event-based touch sensors (Haessig et al. [117]) to further exploit the characteristics of SNNs in terms of energy consumption and information processing (Zambrano et al. [228]). One of the most mature technologies to program SNNs on hardware is the Nengo simulator of the University of Waterloo (Bekolay et al. [51]). Nengo provides a rigorous way of defining SNN models and learning rules. Furthermore, there is also the possibility to run the models on different hardware backends like Loihi, SpiNNaker, and GPUs.

7.3. Outlook

Finally, a discussion about the challenges and the potential of using SNNs for neurorobotics and the role of neurorobotics for neuroscience is presented. Neurorobotics is a challenging field in the intersection between neuroscience, informatics, and robotics. As a result, it is difficult to implement models that can operate in a meaningful way in the different domains. In each of the fields, we have seen considerable progress in the last years: huge complex models of brain regions based on biological data, advances in AI and especially in DL to solve a multitude of tasks, and the design of advanced robot systems with complex kinematics.

7.3.1. Challenges of using SNNs for neurorobotics

The application of SNNs for engineering is still an open field of research. SNNs focus on the biological characteristics of neurons to model more closely the functionality of real neurons. SNNs are designed for spike-based communication, enabling research on brain-like learning and plasticity mechanisms. Experiments have shown that individual spikes are relevant for the brain, not just rates. SNNs can encode temporal information in their signals, making a network of spiking neurons with its synapses a dynamical system evolving in time. SNNs are computationally more powerful than traditional neural networks because they have a higher VC-dimension, as proved in Maass et al. [152]. In theory, any function that a sigmoidal neural network can compute can also be computed by a network of spiking neurons (Maass [151] and Maass et al. [152]). Nevertheless, the implementation of spiking neuron models is very challenging because there is no state-of-the-art learning algorithm, and there is no framework like those for DL (Paugam-Moisy et al. [167]). Additionally, the parameterization of SNNs is an art. Thus, new methods and paradigms are required to program SNNs and to be able to exploit their unique characteristics with the help of neuromorphic hardware (Zenke et al. [229]).

However, recent advances on biologically plausible backpropagation learning rules with spiking neurons offer a new dimension for applications using SNNs. The work in Neftci et al. [162] uses surrogate gradient mechanisms to learn with a performance similar to that of DL. The work in Kaiser et al. [5] uses the principles from feedback alignment to learn visual event streams in a robotics scenario efficiently. The work in Bellec et al. [53] uses eligibility traces as a biologically inspired mechanism reaching similar performance to BPTT. Finally, the work in Schiess et al. [193] presents a mechanism for learning with multi-compartment neurons using dendritic spikes as a biological substrate for backpropagation.

In the current state of maturity, the use of SNNs for robotics applications is a topic primarily for research, but it has great potential for industrial applications. However, the contribution of this thesis and similar works is worthwhile with the potential of obtaining insights about the brain and replicating them. The synergies and collaborations resulting from working in this interdisciplinary research field are a strong motivation for this research.

7.3.2. The role of neurorobotics for neuroscience

Robotics applications — like target reaching or grasping — provide benchmarking tasks and realistic scenarios to validate neuroscience models. These tasks require integrated sensorimotor interactions with dynamic environments (Senden et al. [196]). Anthropomorphic robot hands, such as the Schunk SVH 5-finger hand, provide a kinematic structure similar to that of humans. Thus, it is possible to use a motion representation based on human demonstrations so that the robot can interact with the same objects and in a similar way as humans.

The combination of visual information, arm motion, and soft-grasping capabilities can achieve a more natural grasping process, from recognizing the object to positioning the arm for grasping. The whole system implemented entirely with SNNs as a physical imitation of a biological system, together with an anthropomorphic hand, can be compared to brain neural responses (Abler et al. [41]) of the grasping process as shown by (Kim et al. [137]) and provide new insights into its sub-processes.

Neuroscience principles can be implemented to control complex biological inspired robots, and robots can be used to validate and understand brain mechanisms and learning processes. Intelligence and the human brain did not evolve independently from the body. Cognitive capabilities can emerge from a physically embodied system (Pfeifer et al. [169]). Robotics provides an embodiment to develop learning mechanisms and to test functional models of the brain.

Appendix

A. The Neurorobotics Platform technical details

The NRP is an open-source project that provides a collection of different tools and technologies for defining and performing neurorobotics experiments. The main feature of the NRP is that it combines a robot simulator (body and environment) with a neural simulator (brain) (Tieck et al. [17]). The robot simulator is Gazebo, while the neural simulator can be NEST, Nengo or TensorFlow. The middleware of the NRP that synchronizes and communicates both simulators is the closed-loop engine (CLE). The communication between the brain and the body is implemented using the *Robot Operating System (ROS)*[1]. The NRP is a client server application that can be used in an interactive or non-interactive mode (see Figure A.1). The interactive mode uses a web frontend, and it is mainly used to develop experiments. The non-interactive mode uses the virtual coach, a scripting python API, to run simulations in batch primarily for learning and optimizations.

[1] `http://www.ros.org/`

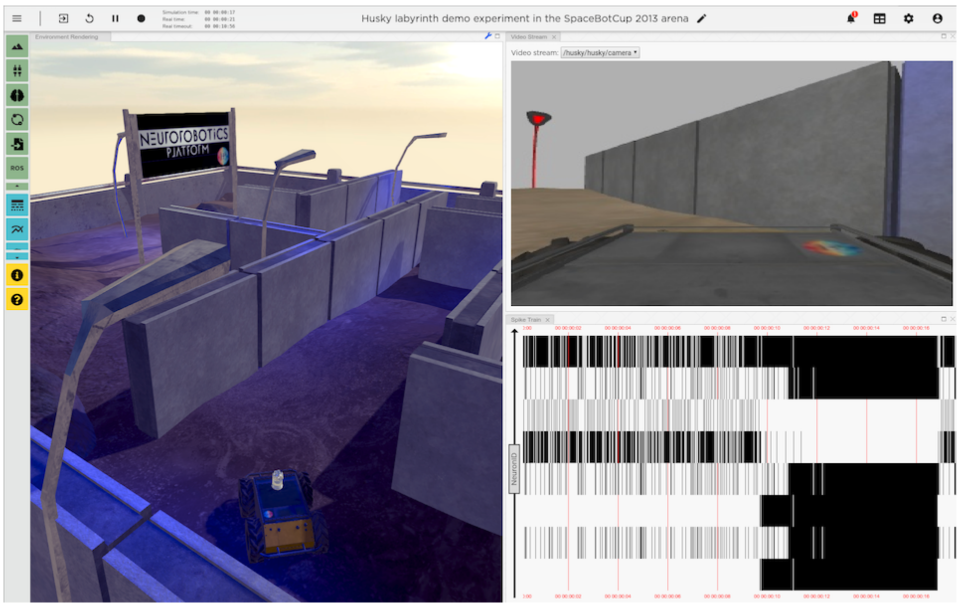

Husky labyrinth demo experiment in the SpaceBotCup 2013 arena
Video stream: /husky/husky/camera
NEUROROBOTICS PLATFORM
Spike Train
Neuron ID

Figure A.1.: Web interface to the NRP (Tieck et al. [17]). In this experiment, a Husky robot solves a maze by following the red lamps. The user can visualize the camera images (top right) and brain activity as a spike-train (bottom right) while the experiment is running.

A.1. Getting and using the NRP

The NRP can be used online or as a local installation. The online deployed platform runs on the high-performance computing (HPC) infrastructure of the HBP. The main benefit of using the online NRP is that it is easier to get started (no installation, no maintenance), and the simulations are running on powerful remote clusters.

The local installation can be done using a USB stick live boot, a docker container, or downloading the sources and compiling. The main benefit of using a local installation is that it provides more freedom for development as it provides access to all files and folders, using version control, and using any library or ROS component. The NRP is a web platform, and internally, the NRP communicates with ROS. With a local install, any ros-node can connect to a simulation. The most important folders for the user are the Models, GazeboRosPackages, and Experiments. The Models folder contains the template robot models (SDF and URDF), environments, and brain scripts PyNN. The GazeboRosPackages folder contains ROS-nodes and plugins for Gazebo — such as the Gazebo DVS plugin (Kaiser et al. [9]). In the Experiments folder, each sub-folder is a template experiment containing transfer functions and state machines. User storage containing user experiments and models resides in $HOME/.opt by default. Users can add any experiment there, either by cloning a template from the web interface or copying an experiment folder.

A.2. Modeling and development with the NRP

In order to model a problem to be solved with the NRP, which also applies to general neurorobotics problems, different aspects need to be considered for the brain, the embodiment (robot), the actions, and the perception (Tieck et al. [17]). For the brain, usually, a SNN or an external controller, the network topology has to be defined, the neuron model to use, the learning mechanism, and the encoding and decoding schemas. For the embodiment, usually a robot or an agent, it is important to define the encoding and decoding schemas according to the interfaces available, considering that robots already have hardware constraints and multiple sensors and actuators. For the actions, the actuators have to be defined, which motors or muscles will be used and what type of object interaction will happen, and how is the interaction with the environment. Finally, for the sensor perception, it is required to define how it is represented, what type of sensors are available (tactile, haptic, force, torque, position), which type of cameras to use (for example, event-based) and other sensors.

In the following, it is described how all this maps to the NRP components (Tieck et al. [17]). To interact with the NRP as a user, you either use the interactive web frontend or the virtual coach for batch processing. Both modes communicate with the NRP backend using REST calls. In the backend, the CLE synchronizes and communicates the neurosimulator with the physics simulator. A brain script, a robot model file, an environment model, and a state machine are needed to define an experiment. The interaction with the environment is represented with a state machine that defines the experiment protocol and controls the simulation. The brain script defines the network or the controller and the interfaces it will have for input and output signals. The robot model file defines the robot kinematics, the inertia, and the sensors and actuators. The communication between the robot and the brain is made using transfer functions provided by the user. Transfer functions can be robot to neuron, neuron to robot, or communicate with an external component. A complete yet straightforward example of developing a brain controller with SNNs for the iCub robot in the NRP is presented in the baseball tutorial experiment[2].

[2] https://url.fzi.de/baseballtutorial

Acronyms

ANN	Artificial Neural Networks vii, 11, 21, 26, 29, 31, 44, 60, 64
backpropagation	Error Backpropagation vii, 27, 31, 32, 167
BPTT	Backpropagation Through Time vii, 44, 167
CNS	Central Nervous System vii, 2, 16, 22, 46, 57
CPG	Central Pattern Generator vii, 18, 149–153, 155–158, 163
DL	Deep Learning vii, 2, 25–27, 46, 165, 167
DMP	Dynamic Movement Primitives vii, 22, 23
DoF	Degrees of Freedom iv, vi, vii, 2, 10, 22, 23, 34, 37, 46, 70, 106, 107, 144
EMG	Electromyography vii, 115–126, 141, 161, 164, 179
FZI	FZI Research Center for Information Technology (`https://www.fzi.de/`). i, vii, 71
GPU	Grapihcs Processing Unit vii, 44, 165
HBP	Human Brain Project i, ii, vii, 7, 41, 166, 172
IK	Inverse Kinematics vii, 1, 4, 6, 19, 46, 83, 102, 161
L2L	Learning to Learn vii
LIF	Leaky-Integrate-and-Fire vii, 28, 29, 90, 100, 112, 121, 138, 177
LSM	Liquid State Machine vii
NEF	Neuro Engineering Framework presented in Eliasmith et al. [99] and Stewart [207] (`https://www.nengo.ai/nengo/examples/advanced/nef-summary.html`). vii, 22, 42, 49, 60–62, 64, 164, 178

Nengo Nengo neurosimulator, a Python tool for building large-scale functional brain models presented in Bekolay et al. [51]. vii, 22, 43, 44, 49, 60, 75, 81, 100, 120–122, 138, 152, 164, 166, 171, 178

NEST Spiking Neural Network Simulator presented in Diesmann et al. [93] and Gewaltig et al. [110]. vii, 42, 43, 60, 89, 90, 111, 156, 171, 176

NRP Neurorobotics Platform presented in Falotico et al. [2] and Tieck et al. [17] (`https://bitbucket.org/hbpneurorobotics`). vii, 7, 41, 42, 144, 151, 152, 156, 158, 164, 171–173

PCA Principal Component Analysis vii, 12, 18

PES Prescribed Error Sensitivity proposed in MacNeil et al. [153]. vii, 62, 67, 119, 134, 164

PyNN Python library offering a common wrapper over different SNN simulators (backends) such as NEST, SpiNNaker and others, introduced in Davison [88]. vii, 43, 89, 172

PyTorch Generic deep learning framework presented in Paszke et al. [166]. vii, 44

RL Reinforcement Learning vii, 25

ROS Robot Operating System presented in Quigley et al. [176] (`http://www.ros.org/`). vii, 60, 70, 71, 75, 89, 90, 94, 100, 106, 107, 112, 117, 121–123, 138, 144, 152, 163, 172

SNN Spiking Neural Network i–vii, 3–7, 9, 22, 26, 27, 29, 31, 32, 36–38, 42, 44–49, 54, 58–61, 64–66, 68–70, 73–75, 80, 82, 83, 89, 90, 94, 96–106, 108–118, 120–123, 126, 129–132, 134, 138, 140–144, 146–149, 151, 152, 156, 158, 159, 161–168, 173, 176–179

STDP Spike Time Dependent Plasticity vii, 6, 31, 32, 108–110, 114, 141

TCP Tool Center Point vii, 33, 60, 66, 67, 70, 76, 77, 79, 82, 83, 89–91, 94, 95, 97–103, 178

TensorFlow Generic deep learning framework presented in Abadi et al. [39]. vii, 44

WTA Winner-Takes-All vii, 32

List of Figures

Bibliography

Own publications

This section lists all publications in which the author of this dissertation is either the first author or has contributed significantly to the publication as a co-author (in the form of problem definition, solution, discussion or experimental evaluation).

[1] Tobi Delbruck, Ibrahim Abe M. Elfadel, Shahzad Muzaffar, Germain Haessig, Bo Wang, Amine Bermak, Rui Graca, Luis Camunas-Mesa, Bathiya Senevirathna, Pamela Abshire, Bernabe Linares-Barranco, Saeed Afshar, Shih-Chii Liu, Runchun Mark Wang, Piotr Dudek, Stephen Carey, Jose de la Rosa, Marc Dandin, Sheung Lu, Vincent Frick, Teresa Serrano-Gotarredona, Paula Lopez, Melika Payvand, Advait Madhavan, Eric Fossum, J. Camilo Vasquez Tieck, Ian Williams, Yan Liu, Timothy Constandinou, Alexander Serb, Ricardo Carmona-Galan, Robert Nawrocki, and Walter D. Leon-Salas. “Lessons Learned the Hard Way”. In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020 IEEE International Symposium on Circuits and Systems (ISCAS). Sevilla: IEEE, Oct. 2020, pp. 1–18. ISBN: 978-1-72813-320-1. DOI: 10.1109/ISCAS45731.2020.9180983.

[2] Egidio Falotico, Lorenzo Vannucci, Alessandro Ambrosano, Ugo Albanese, Stefan Ulbrich, J. Camilo Vasquez Tieck, Georg Hinkel, Jacques Kaiser, Igor Peric, Oliver Denninger, Nino Cauli, Murat Kirtay, Arne Roennau, Gudrun Klinker, Axel Von Arnim, Luc Guyot, Daniel Peppicelli, Pablo Mactinaz-Cañada, Eduardo Ros, Patrick Maier, Sandro Weber, Manuei Huber, David Plecher, Florian Röhrbein, Stefan Deser, Alina Roitberg, Patrick Van Der Smagt, Rüdiger Dillman, Paul Levi, Cecilia Laschi, Alois C. Knoll, and Marc Oliver Gewaltig. “Connecting Artificial Brains to Robots in a Comprehensive Simulation Framework: The Neurorobotics Platform”. In: *Frontiers in Neurorobotics* 11 (JAN 2017). ISSN: 16625218. DOI: 10.3389/fnbot.2017.00002.

[3] Jacques Kaiser, Alexander Friedrich, J. Camilo Vasquez Tieck, Daniel Reichard, Arne Roennau, Emre Neftci, and Rüdiger Dillmann. *Embodied Event-Driven Random Backpropagation*. 2019. URL: https://arxiv.org/abs/1904.04805.

[4] Jacques Kaiser, Alexander Friedrich, J. Camilo Vasquez Tieck, Daniel Reichard, Arne Roennau, Emre Neftci, and Rüdiger Dillmann. "Embodied Neuromorphic Vision with Continuous Random Backpropagation". In: *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*. 2020, pp. 1202–1209.

[5] Jacques Kaiser, Alexander Friedrich, J. Camilo Vasquez Tieck, Daniel Reichard, Arne Roennau, Emre Neftci, and Rüdiger Dillmann. *Embodied Neuromorphic Vision with Event-Driven Random Backpropagation*. 2019.

[6] Jacques Kaiser, Michael Hoff, Andreas Konle, J. Camilo Vasquez Tieck, David Kappel, Daniel Reichard, Anand Subramoney, Robert Legenstein, Arne Roennau, Wolfgang Maass, and Rüediger Dillmann. "Embodied Synaptic Plasticity with Online Reinforcement Learning". In: *Frontiers in Neurorobotics* (2019).

[7] Jacques Kaiser, Gerd Lindner, J. Camilo Vasquez Tieck, Martin Schulze, Michael Hoff, Arne Roennau, and Rüdiger Dillmann. "Microsaccades for Asynchronous Feature Extraction with Spiking Networks". In: *International Conference on Development and Learning and Epigenetic Robotics (ICDL-EPIROB)*. 2018.

[8] Jacques Kaiser, Svenja Melbaum, J. Camilo Vasquez Tieck, Arne Roennau, Martin V Butz, and Rüdiger Dillmann. "Learning to Reproduce Visually Similar Movements by Minimizing Event-Based Prediction Error". In: *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (BIOROB)*. 2018, pp. 260–267.

[9] Jacques Kaiser, J. Camilo Vasquez Tieck, Christian Hubschneider, Peter Wolf, Michael Weber, Michael Hoff, Alexander Friedrich, Konrad Wojtasik, Arne Roennau, Ralf Kohlhaas, Rüdiger Dillmann, and J. Marius Zollner. "Towards a Framework for End-to-End Control of a Simulated Vehicle with Spiking Neural Networks". In: *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots, SIMPAR 2016*. 2016, pp. 127–134. ISBN: 978-1-5090-4616-4. DOI: 10.1109/SIMPAR.2016.7862386.

[10] Jacques Kaiser, Jakob Weinland, Philip Keller, Lea Steffen, J. Camilo Vasquez Tieck, Daniel Reichard, Arne Roennau, Jörg Conradt, and Rüdiger Dillmann. "Microsaccades for Neuromorphic Stereo Vision". In: *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (BIOROB)*. 2018, pp. 244–252.

[11] Jacques Kaiser, David Zimmerer, J. Camilo Vasquez Tieck, Stefan Ulbrich, Arne Roennau, and Rüdiger Dillmann. "Spiking Convolutional Deep Belief Networks". In: *International Conference on Artificial Neural Networks ICANN 2017*. Vol. 10614 LNCS. 2017, pp. 3–11. ISBN: 978-3-319-68611-0. DOI: 10.1007/978-3-319-68612-7_1.

[12] Lea Steffen, Rafael Kübler da Silva, Stefan Ulbrich, J. Camilo Vasquez Tieck, Arne Roennau, and Rüdiger Dillmann. "Networks of Place Cells for Representing 3D Environments and Path Planning". In: *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*. 2020, pp. 1158–1165.

[13] Lea Steffen, J. Camilo Vasquez Tieck, and Rüdiger Dillman. "A Framework for Robot Control with Multi-Modal Motion Activation Using Spiking Neurons (Abstract)". In: *2nd HBP Student Conference : Transdisciplinary Research Linking Neuroscience, Brain Medicine and Computer Science : Book of Abstracts*. Ed. by Andrea Santuy. Ljubjana, Slovenia: Frontiers Event Abstracts, 2019, pp. 53–55.

[14] J. Camilo Vasquez Tieck, Pascal Becker, Jacques Kaiser, Igor Peric, Mahmoud Akl, Daniel Reichard, Arne Roennau, and Rüdiger Dillmann. "Learning Target Reaching Motions with a Robotic Arm Using Brain-Inspired Dopamine Modulated STDP". In: *2019 IEEE 19th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. 2019.

[15] J. Camilo Vasquez Tieck, Heiko Donat, Jacques Kaiser, Igor Peric, Stefan Ulbrich, Arne Roennau, Marius Zoellner, and Rüdiger Dillman. "Towards Grasping with Spiking Neural Networks for Anthropomorphic Robot Hands (Abstract)". In: *2nd HBP Student Conference : Transdisciplinary Research Linking Neuroscience, Brain Medicine and Computer Science : Book of Abstracts*. Ed. by Andrea Santuy. Ljubjana, Slovenia: Frontiers Event Abstracts, 2019, pp. 73–75.

[16] J. Camilo Vasquez Tieck, Heiko Donat, Jacques Kaiser, Igor Peric, Stefan Ulbrich, Arne Roennau, Marius Zöllner, and Rüdiger Dillmann. "Towards Grasping with Spiking Neural Networks for Anthropomorphic Robot Hands". In: *International Conference on Artificial Neural Networks ICANN*. Vol. 10613 LNCS. 2017, pp. 43–51. ISBN: 9783319685991. DOI: 10.1007/978-3-319-68600-4_6.

[17] J. Camilo Vasquez Tieck, Jacques Kaiser, Lea Steffen, Martin Schulze, Axel Von Arnim, Daniel Reichard, Arne Roennau, and Rüdiger Dillmann. "The Neurorobotics Platform for Teaching – Embodiment Experiments with Spiking Neural Networks and Virtual Robots". In: *2019 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. 2019.

[18] J. Camilo Vasquez Tieck, Marin Vlastelica Pogančić, Jacques Kaiser, Arne Roennau, Marc-Oliver Gewaltig, and Rüdiger Dillmann. "Learning Continuous Muscle Control for a Multi-Joint Arm by Extending Proximal Policy Optimization with a Liquid State Machine". In: *International Conference on Artificial Neural Networks ICANN 2018*. Vol. 11139 LNCS. 2018, pp. 211–221. ISBN: 978-3-030-01417-9. DOI: 10.1007/978-3-030-01418-6_21.

[19] J. Camilo Vasquez Tieck, Jacqueline Rutschke, Jacques Kaiser, Martin Schulze, Timothee Buettner, Daniel Reichard, Arne Roennau, and Rüdiger Dillmann. "Combining Spiking Motor Primitives with a Behavior-Based Architecture to

Model Locomotion for Six-Legged Robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019.

[20] J. Camilo Vasquez Tieck, Tristan Schnell, Jacques Kaiser, Felix Mauch, Arne Roennau, and Rüdiger Dillmann. "Generating Pointing Motions for a Humanoid Robot by Combining Motor Primitives". In: *Frontiers in Neurorobotics* 13 (2019), p. 77.

[21] J. Camilo Vasquez Tieck, Katharina Secker, Jacques Kaiser, Arne Roennau, and Rüdiger Dillmann. "Soft-Grasping with an Anthropomorphic Robotic Hand Using Spiking Neurons". In: *IEEE Robotics and Automation Letters* (Oct. 26, 2020). DOI: 10.1109/LRA.2020.3034067.

[22] J. Camilo Vasquez Tieck, Lea Steffen, Jacques Kaiser, Daniel Reichard, Arne Roennau, and Rüdiger Dillmann. "Combining Motor Primitives for Perception Driven Target Reaching with Spiking Neurons". In: *Cognitive Informatics and Natural Intelligence (IJCINI)* 13.1 (2019), p. 12. ISSN: 15573966. DOI: 10.4018/IJCINI.2019010101.

[23] J. Camilo Vasquez Tieck, Lea Steffen, Jacques Kaiser, Daniel Reichard, Arne Roennau, and Rüdiger Dillmann. "Controlling a Robot Arm for Target Reaching without Planning Using Spiking Neurons". In: *2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. 2018, pp. 111–116. ISBN: 978-1-5386-3360-1. DOI: 10.1109/ICCI-CC.2018.8482049.

[24] J. Camilo Vasquez Tieck, Lea Steffen, Jacques Kaiser, Arne Roennau, and Rüdiger Dillmann. "Multi-Modal Motion Activation for Robot Control Using Spiking Neurons". In: *2018 IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. Vol. 2018-August. 2018, pp. 291–298. ISBN: 9781538681831. DOI: 10.1109/BIOROB.2018.8487786.

[25] J. Camilo Vasquez Tieck, Sandro Weber, Terrence C Stewart, Arne Roennau, and Rüdiger Dillmann. "Triggering Robot Hand Reflexes with Human EMG Data Using Spiking Neurons". In: *International Conference on Intelligent Autonomous Systems IAS-15*. Vol. 867. Springer, Cham, 2018, pp. 902–916. ISBN: 978-3-030-01369-1. DOI: 10.1007/978-3-030-01370-7_70.

[26] J. Camilo Vasquez Tieck, Sandro Weber, Terrence C. Stewart, Jacques Kaiser, Arne Roennau, and Rüdiger Dillmann. "A Spiking Network Classifies Human sEMG Signals and Triggers Finger Reflexes on a Robotic Hand". In: *Robotics and Autonomous Systems* 131 (2020), p. 103566. ISSN: 09218890. DOI: 10.1016/j.robot.2020.103566.

Student works

This section lists all student theses that were developed and supervised by the author of this dissertation during his research. This includes the main idea and key details for the specification of the problem, discussion of the work, as well as boundary requirements for solution, visualization and experimental evaluation.

[27] Heiko Donat and J. Camilo Vasquez Tieck. "Towards Grasping with Spiking Neural Networks for an Anthropomorphic Robot Hand". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, July 31, 2016.

[28] Michael Hoff, Jacques Kaiser, and J. Camilo Vasquez Tieck. "Learning Closed-Loop Robot Control with Spiking Neurons and Event-Based Vision". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, July 31, 2017.

[29] Sebastian Jahr and J. Camilo Vasquez Tieck. "A Biological Inspired Compliant Controller for a Robotic Arm Driven by Spiking Neurons". Bachelor Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, May 31, 2018.

[30] Benjamin Kaufmann, J. Camilo Vasquez Tieck, and Jacques Kaiser. "A Framework for Closed-Loop Sensori-Motor Robot Ex- Periments in the Neurorobotics Platform". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, Feb. 28, 2018.

[31] Marin Vlastelica Pogančić and J. Camilo Vasquez Tieck. "Learning Multi-Joint Continuous Control with Spiking Neural Networks". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, July 31, 2017.

[32] Jacqueline Rutschke and J. Camilo Vasquez Tieck. "Controlling LAURON with Spiking Neural Networks". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, Oct. 31, 2018.

[33] Tristan Schnell and J. Camilo Vasquez Tieck. "A Spiking Neural Network for Arm Motion Adaptation Using Motor Primitives". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, Sept. 30, 2017.

[34] Katharina Secker and J. Camilo Vasquez Tieck. ""Soft Grasping" einer Fünffinger Roboterhand mittels Gepulsten Neuronalen Netzen". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, Sept. 11, 2019.

[35] Lea Steffen and J. Camilo Vasquez Tieck. "Multi-Modal Motion Activation for Robot Control Using Spiking Neurons". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, July 31, 2017.

[36] Atanas Tanev and J. Camilo Vasquez Tieck. "A Spiking Network for Adaptive Control of a Robotic Arm Using a Model of the Cerebellum". Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, Mar. 26, 2017.

[37] Christian Telpl, J. Camilo Vasquez TIeck, and Lea Steffen. “Controlling A Real Robot Arm With Spiking Neurons Using Motor Primitives For Target Reaching”. Master Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, Feb. 1, 2020.

[38] Konrad Wojtasik, Jacques Kaiser, and J. Camilo Vasquez Tieck. “Spiking Neural Networks for Arm Pose Estimation Using a Dynamic Vision Sensor”. Bachelor Thesis. Karlsruhe, Germany: KIT Karlsruhe Institute of Technology, Apr. 30, 2017.

General references

[39] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: A System for Large-Scale Machine Learning". In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[40] Larry F Abbott. "Lapicque's Introduction of the Integrate-and-Fire Model Neuron (1907)". In: *Brain Research Bulletin* 50.5-6 (1999), pp. 303–304. ISSN: 03619230. DOI: 10.1016/S0361-9230(99)00161-6.

[41] Birgit Abler, Alard Roebroeck, Rainer Goebel, Anett Höse, Carlos Schönfeldt-Lecuona, Günter Hole, and Henrik Walter. "Investigating Directed Influences between Activated Brain Areas in a Motor-Response Task Using fMRI". In: *Magnetic Resonance Imaging* 24.2 (Feb. 2006), pp. 181–185. ISSN: 0730725X. DOI: 10.1016/j.mri.2005.10.022.

[42] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, Brian Taba, Michael Beakes, Bernard Brezzo, Jente B. Kuang, Rajit Manohar, William P. Risk, Bryan Jackson, and Dharmendra S. Modha. "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pp. 1537–1557. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2015.2474396.

[43] Jan Albiez. *Verhaltensnetzwerke Zur Adaptiven Steuerung Biologisch Motivierter Laufmaschinen*. Forschen Und Wissen - Robotik. GCA-Verl., Waabs, 2007. ISBN: 978-3-89863-229-4.

[44] James S Albus. "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)". In: (1975).

[45] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. "Learning Dexterous In-Hand Manipulation". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364919887447.

[46] Alberto Antonietti, Dario Martina, Claudia Casellato, Egidio D'Angelo, and Alessandra Pedrocchi. "Control of a Humanoid NAO Robot by an Adaptive Bioinspired Cerebellar Module in 3D Motion Tasks". In: *Computational Intelligence and Neuroscience* 2019 (2019), pp. 1–15. ISSN: 1687-5265, 1687-5273. DOI: 10.1155/2019/4862157.

[47] Michael A. Arbib, ed. *The Handbook of Brain Theory and Neural Networks*. 2nd ed. Cambridge, Mass: MIT Press, 2003. 1290 pp. ISBN: 978-0-262-01197-6.

[48] Tamim Asfour, Julian Schill, Heiner Peters, Cornelius Klas, Jens Bucker, Christian Sander, Stefan Schulz, Artem Kargov, Tino Werner, and Volker Bartenbach. “ARMAR-4: A 63 DOF Torque Controlled Humanoid Robot”. In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2013). Atlanta, GA: IEEE, 2013, pp. 390–396. ISBN: 978-1-4799-2617-6 978-1-4799-2619-0. DOI: `10.1109/HUMANOIDS.2013.7030004`.

[49] Ola Ayaso. “A Model for Command Generation in Motor Cortex”. Massachusetts Institute of Technology, 2001.

[50] Ulrich Bässler and Ansgar Büschges. “Pattern Generation for Stick Insect Walking Movements—Multisensory Control of a Locomotor Program”. In: *Brain Research Reviews* 27.1 (1998), pp. 65–88. ISSN: 01650173. DOI: `10.1016/S0165-0173(98)00006-X`.

[51] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C. Stewart, Daniel Rasmussen, Xuan Choo, Aaron Voelker, and Chris Eliasmith. “Nengo: A Python Tool for Building Large-Scale Functional Brain Models”. In: *Frontiers in Neuroinformatics* 7.48 (2014), pp. 1–13. ISSN: 1662-5196. DOI: `10.3389/fninf.2013.00048`.

[52] Trevor Bekolay, Carter Kolbeck, and Chris Eliasmith. “Simultaneous Unsupervised and Supervised Learning of Cognitive Functions in Biologically Plausible Spiking Neural Networks”. In: *Proceedings of the Annual Meeting of the Cognitive Science Society (CogSci)*. Vol. 35. 35. 2013.

[53] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. *A Solution to the Learning Dilemma for Recurrent Networks of Spiking Neurons*. preprint. Neuroscience, 2019. DOI: `10.1101/738385`.

[54] Nikolai Bernstein. *The Co-Ordination and Regulation of Movements*. Pergamon Press Ltd., 1967. 196 pp. DOI: `0022510X68901664`.

[55] E. Bizzi, V.C.K. Cheung, A. d’Avella, P. Saltiel, and M. Tresch. “Combining Modules for Movement”. In: *Brain Research Reviews* 57.1 (2008), pp. 125–133. ISSN: 01650173. DOI: `10.1016/j.brainresrev.2007.08.004`.

[56] M. Bonilla, E. Farnioli, C. Piazza, M. Catalano, G. Grioli, M. Garabini, M. Gabiccini, and A. Bicchi. “Grasping with Soft Hands”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. 2014 IEEE-RAS 14th International Conference on Humanoid Robots (Humanoids 2014). Madrid, Spain: IEEE, 2014, pp. 581–587. ISBN: 978-1-4799-7174-9. DOI: `10.1109/HUMANOIDS.2014.7041421`.

[57] 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 International Joint Conference on Neural Networks (IJCNN). Barcelona, Spain: IEEE, 2010, pp. 18–23. ISBN: 9781424469178. DOI: 10.1109/IJCNN.2010.5596525.

[58] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. Vol. 95. 1. MIT press, 1986. ISBN: 0-262-02208-7 0-262-52112-1. DOI: 10.2307/2185146.

[59] Johanni Brea, Alexisz Tamás Gaál, Robert Urbanczik, and Walter Senn. “Prospective Coding by Spiking Neurons”. In: *PLOS Computational Biology* 12.6 (June 24, 2016). Ed. by Peter E. Latham, e1005003. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005003.

[60] T Graham Brown. “On the Nature of the Fundamental Activity of the Nervous Centres; Together with an Analysis of the Conditioning of Rhythmic Activity in Progression, and a Theory of the Evolution of Function in the Nervous System”. In: *The Journal of physiology* 48.1 (1914), p. 18.

[61] Julian Büchel, Dmitrii Zendrikov, Sergio Solinas, Giacomo Indiveri, and Dylan R. Muir. *Supervised Training of Spiking Neural Networks for Robust Deployment on Mixed-Signal Neuromorphic Processors*. Feb. 17, 2021. URL: http://arxiv.org/abs/2102.06408 (visited on 04/27/2021).

[62] Timothee Buettner, Georg Heppner, Arne Roennau, and Rüdiger Dillmann. “Nimble Limbs - Make Anything Walk with Intelligent Attachable Legs”. In: *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM2019*. 2019.

[63] Daniel Bullock, Stephen Grossberg, and Prank H Guenther. “A Self-Oorganizing Neural Model of Motor Equivalent Reaching and Tool Use by a Multijoint Arm”. In: *Journal of Cognitive Neuroscience* 5.4 (1993), pp. 408–435.

[64] JH Byrne and N Dafny. *Neuroscience Online: An Electronic Textbook for the Neurosciences*. Department of Neurobiology and Anatomy, The University of Texas Medical School at Houston, 1997. URL: http://nba.uth.tmc.edu/neuroscience/.

[65] D.G. Caldwell and N. Tsagarakis. “"Soft" Grasping Using a Dextrous Hand”. In: *Industrial Robot* 27.3 (2000), pp. 194–199. ISSN: 0143-991X. DOI: 10.1108/01439910010323941.

[66] Marie Claire Capolei, Emmanouil Angelidis, Egidio Falotico, Henrik Hautop Lund, and Silvia Tolu. “A Biomimetic Control Method Increases the Adaptability of a Humanoid Robot Acting in a Dynamic Environment”. In: *Frontiers in neurorobotics* 13 (2019), p. 70.

[67] Claudio Castellini and Patrick van der Smagt. “Evidence of Muscle Synergies during Human Grasping”. In: *Biological Cybernetics* 107.2 (2013), pp. 233–245. ISSN: 0340-1200, 1432-0770. DOI: 10.1007/s00422-013-0548-4.

[68] E. Chinellato and A.P. Pobil. *The Visual Neuroscience of Robotic Grasping*. Cognitive Systems Monographs. Springer International Publishing, 2016. ISBN: 978-3-319-20303-4. DOI: 10.1007/978-3-319-20303-4.

[69] Rubana H Chowdhury, Mamun BI Reaz, Mohd Alauddin Bin Mohd Ali, Ashrif AA Bakar, Kalaivani Chellappan, and Tae G Chang. "Surface Electromyography Signal Processing and Classification Techniques". In: *Sensors* (2013).

[70] Mark M. Churchland, John P. Cunningham, Matthew T. Kaufman, Justin D. Foster, Paul Nuyujukian, Stephen I. Ryu, Krishna V. Shenoy, and Krishna V. Shenoy. "Neural Population Dynamics during Reaching". In: *Nature* 487.7405 (2012), pp. 51–56. ISSN: 00280836. DOI: 10.1038/nature11129.

[71] Matei Ciocarlie, Corey Goldfeder, and Peter Allen. "Dexterous Grasping via Eigengrasps: A Low-Dimensional Approach to a High-Complexity Problem". In: *Robotics: Science and Systems Manipulation Workshop-Sensing and Adapting to the Real World*. 2007.

[72] Matei Th Ciocarlie. "Low-Dimensional Robotic Grasping: Eigengrasp Subspaces and Optimized Underactuation". In: *ProQuest Dissertations and Theses* (2010).

[73] S Cobos, M Ferre, and R Aracil. "Simplified Human Hand Models Based on Grasping Analysis". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010). Taipei: IEEE, 2010, pp. 610–615. ISBN: 978-1-4244-6674-0. DOI: 10.1109/IROS.2010.5651479.

[74] A.H. Cohen and P. Wallén. "The Neuronal Correlate of Locomotion in Fish: "Fictive Swimming" Induced in an in Vitro Preparation of the Lamprey Spinal Cord". In: *Experimental Brain Research* 41.1 (1980). ISSN: 0014-4819, 1432-1106. DOI: 10.1007/BF00236674.

[75] Patrick E Crago, Richard J Nakai, and Howard J Chizeck. "Feedback Regulation of Hand Grasp Opening and Contact Force during Stimulation of Paralyzed Muscle". In: *IEEE Transactions on Biomedical Engineering* 38.1 (1991), pp. 17–28.

[76] H. Cruse, V. Dürr, M. Schilling, and J. Schmitz. "Principles of Insect Locomotion". In: *Spatial Temporal Patterns for Action-Oriented Perception in Roving Robots*. Ed. by Paolo Arena and Luca Patanè. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 43–96. ISBN: 978-3-540-88464-4. DOI: 10.1007/978-3-540-88464-4âĆĆ.

[77] Holk Cruse, Thomas Kindermann, Michael Schumm, Jeffrey Dean, and Josef Schmitz. "Walknet - A Biologically Inspired Network to Control Six-Legged Walking". In: *Neural Networks* 11.7-8 (1998), pp. 1435–1447. ISSN: 08936080. DOI: 10.1016/S0893-6080(98)00067-7.

[78] Mark R Cutkosky. "On Grasp Choice, Grasp Models, and the Design of Hands for Manufacturing Tasks". In: *IEEE Transactions on Robotics and Automation* 5.3 (1989), pp. 269–279.

[79] Egidio D'Angelo, Alberto Antonietti, Stefano Casali, Claudia Casellato, Jesus A. Garrido, Niceto Rafael Luque, Lisa Mapelli, Stefano Masoli, Alessandra Pedrocchi, Francesca Prestori, Martina Francesca Rizza, and Eduardo Ros. "Modeling the Cerebellar Microcircuit: New Strategies for a Long-Standing Issue". In: *Frontiers in Cellular Neuroscience* 10 (July 2016), pp. 1–29. ISSN: 1662-5102. DOI: 10.3389/fncel.2016.00176.

[80] Egidio D'Angelo and Stefano Casali. "Seeking a Unified Framework for Cerebellar Function and Dysfunction: From Circuit Operations to Cognition". In: *Frontiers in Neural Circuits* 6 (2013). ISSN: 1662-5110. DOI: 10.3389/fncir.2012.00116.

[81] Andrea d'Avella, Martin Giese, Yuri P. Ivanenko, Thomas Schack, and Tamar Flash. "Editorial: Modularity in Motor Control: From Muscle Synergies to Cognitive Action Representation". In: *Frontiers in Computational Neuroscience* 9 (2015). ISSN: 1662-5188. DOI: 10.3389/fncom.2015.00126.

[82] Andrea d'Avella and Francesco Lacquaniti. "Control of Reaching Movements by Muscle Synergy Combinations". In: *Frontiers in Computational Neuroscience* 7 (April 2013), pp. 1–7. ISSN: 1662-5188. DOI: 10.3389/fncom.2013.00042.

[83] Andrea d'Avella, Alessandro Portone, and Francesco Lacquaniti. "Superposition and Modulation of Muscle Synergies for Reaching in Response to a Change in Target Location". In: *Journal of Neurophysiology* 106.6 (2011), pp. 2796–2812. ISSN: 0022-3077, 1522-1598. DOI: 10.1152/jn.00675.2010.

[84] Andrea d'Avella, Philippe Saltiel, and Emilio Bizzi. "Combinations of Muscle Synergies in the Construction of a Natural Motor Behavior". In: *Nature Neuroscience* 6.3 (2003), pp. 300–308.

[85] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99. ISSN: 0272-1732. DOI: 10.1109/MM.2018.112130359.

[86] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A. Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R. Risbud. "Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook". In: *Proceedings of the IEEE* (2021), pp. 1–24. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2021.3067593.

[87] A. P. Davison and Y. Frégnac. "Learning Cross-Modal Spatial Transformations through Spike Timing-Dependent Plasticity". In: *Journal of Neuroscience* 26.21 (2006), pp. 5604–5615. ISSN: 0270-6474, 1529-2401. DOI: 10.1523/JNEUROSCI.5263-05.2006.

[88] Andrew P Davison. "PyNN: A Common Interface for Neuronal Network Simulators". In: *Frontiers in Neuroinformatics* 2 (January 2008), p. 11. ISSN: 16625196. DOI: 10.3389/neuro.11.011.2008.

[89] Shoubhik Debnath, John Nassour, and Gordon Cheng. "Learning Diverse Motor Patterns with a Single Multi-Layered Multi-Pattern CPG for a Humanoid Robot". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. 2014 IEEE-RAS 14th International Conference on Humanoid Robots (Humanoids 2014). Madrid, Spain: IEEE, Nov. 2014, pp. 1016–1021. ISBN: 978-1-4799-7174-9. DOI: 10.1109/HUMANOIDS.2014.7041489.

[90] T DeWolf and C Eliasmith. "Trajectory Generation Using a Spiking Neuron Implementation of Dynamic Movement Primitives". In: *27th Annual Meeting for the Society for the Neural Control of Movement*. 2017.

[91] Travis DeWolf, Pawel Jaworski, and Chris Eliasmith. "Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics". In: *Frontiers in Neurorobotics* 14 (Oct. 9, 2020), p. 568359. ISSN: 1662-5218. DOI: 10.3389/fnbot.2020.568359.

[92] Travis DeWolf, Terrence C. Stewart, Jean-Jacques Slotine, and Chris Eliasmith. "A Spiking Neural Model of Adaptive Arm Control". In: *Proceedings of the Royal Society B: Biological Sciences* 283.1843 (2016), p. 20162134. ISSN: 0962-8452. DOI: 10.1098/rspb.2016.2134.

[93] Markus Diesmann and Marc-Oliver Gewaltig. "NEST: An Environment for Neural Systems Simulations". In: *Forschung und wissenschaftliches Rechnen. Beiträge zum Heinz-Billing-Preis 2001* 58 (2002), pp. 43–70.

[94] R Dillmann and PM Frank. "A New Approach to Pulse Frequency Modulated Control". In: *IFAC Proceedings Volumes* 14.2 (1981), pp. 405–410.

[95] Rüdiger Dillmann. "Ein Geschlossenes Mathematisches Modell Für Regelkreise Mit Pulsfrequenzmodulation". Dissertation. Karlsruhe, Germany: University of Karlsruhe, 1981.

[96] Elisa Donati, Fernando Perez-Pena, Chiara Bartolozzi, Giacomo Indiveri, and Elisabetta Chicca. "Open-Loop Neuromorphic Controller Implemented on VLSI Devices". In: *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*. 2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob). Enschede: IEEE, 2018, pp. 827–832. ISBN: 978-1-5386-8183-1. DOI: 10.1109/BIOROB.2018.8487937.

[97] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen. "A Large-Scale Model of the Functioning Brain". In: *Science* 338.6111 (2012), pp. 1202–1205. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1225266.

[98] Chris Eliasmith. *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford University Press, 2013.

[99] Chris Eliasmith and Charles H. Anderson. *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT Press, 2003.

[100] Andrew H Fagg and Michael A Arbib. "Modeling Parietal–Premotor Interactions in Primate Control of Grasping". In: *Neural Networks* 11.7-8 (1998), pp. 1277–1303. ISSN: 08936080. DOI: 10.1016/S0893-6080(98)00047-1.

[101] Fanny Ficuciello, Alba Federico, Vincenzo Lippiello, and Bruno Siciliano. "Synergies Evaluation of the SCHUNK S5FH for Grasping Control". In: *Advances in Robot Kinematics 2016*. Springer, 2018, pp. 225–233.

[102] Flavia Filimon, Jonathan D Nelson, Ruey-Song Huang, and Martin I Sereno. "Multiple Parietal Reach Regions in Humans: Cortical Representations for Visual and Proprioceptive Feedback during On-Line Reaching". In: *Journal of Neuroscience* 29.9 (2009), pp. 2961–2971. DOI: 10.1523/JNEUROSCI.3211-08.2009.

[103] Tamar Flash and Binyamin Hochner. "Motor Primitives in Vertebrates and Invertebrates". In: *Current Opinion in Neurobiology* 15.6 (2005), pp. 660–666. ISSN: 09594388. DOI: 10.1016/j.conb.2005.10.011.

[104] Steve B. Furber, David R. Lester, Luis A. Plana, Jim D. Garside, Eustace Painkras, Steve Temple, and Andrew D. Brown. "Overview of the SpiNNaker System Architecture". In: *IEEE Transactions on Computers* 62.12 (2013), pp. 2454–2467. ISSN: 0018-9340. DOI: 10.1109/TC.2012.142.

[105] Forschungszentrum Informatik FZI. *HoLLiE – The Assistance Robot*. 2020. URL: https://www.fzi.de/en/research/projekt-details/hollie/ (visited on 04/21/2020).

[106] Forschungszentrum Informatik FZI. *LAURON*. 2020. URL: https://www.fzi.de/en/research/projekt-details/lauron/ (visited on 04/20/2020).

[107] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. "Event-Based Vision: A Survey". In: *CoRR* abs/1904.08405 (2019). URL: http://arxiv.org/abs/1904.08405.

[108] Wulfram Gerstner. "Time Structure of the Activity in Neural Network Models". In: *Physical review E* 51.1 (1995), p. 738.

[109] Wulfram. Gerstner, Werner M. Kistler, Richard. Naud, and Liam. Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press, 2014, p. 577. ISBN: 1-107-63519-5. DOI: 10.1017/CBO9781107447615.

[110] Marc-Oliver Gewaltig and Markus Diesmann. "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4 (2007), p. 1430.

[111] Aditya Gilra and Wulfram Gerstner. "Non-Linear Motor Control by Local Learning in Spiking Neural Networks". In: *arXiv preprint* (2017). URL: http://arxiv.org/abs/1712.10158.

[112] Aditya Gilra and Wulfram Gerstner. "Predicting Non-Linear Dynamics by Stable Local Learning in a Recurrent Spiking Neural Network". In: *eLife* 6 (2017), e28295. ISSN: 2050-084X. DOI: 10.7554/eLife.28295.

[113] Dan Goodman. "Brian: A Simulator for Spiking Neural Networks in Python". In: *Frontiers in Neuroinformatics* 2 (2008). ISSN: 16625196. DOI: 10.3389/neuro.11.005.2008.

[114] Henry Gray. *Anatomy of the Human Body*. 20th ed., thoroughly rev. and re-edited by Warren H. Lewis. Philadelphia: Lea & Febiger, 1918. 1396 pp. ISBN: 1-58734-102-6. URL: https://www.bartleby.com/107/.

[115] André Grüning and Sander M Bohte. "Spiking Neural Networks: Principles and Challenges". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning – ESANN*. Vol. 44. April. 2014, pp. 23–25. ISBN: 9782874190957. URL: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2014-13.pdf.

[116] R Gütig et al. "Learning Input Correlations through Nonlinear Temporally Asymmetric Hebbian Plasticity". In: *Journal of Neuroscience* 23.9 (2003), pp. 3697–3714. ISSN: 1529-2401.

[117] Germain Haessig, Moritz B. Milde, Pau Vilimelis Aceituno, Omar Oubari, James C. Knight, André van Schaik, Ryad B. Benosman, and Giacomo Indiveri. "Event-Based Computation for Touch Localization Based on Precise Spike Timing". In: *Frontiers in Neuroscience* 14 (2020), p. 420. ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00420.

[118] C. B. Hart and S. F. Giszter. "A Neural Basis for Motor Primitives in the Spinal Cord". In: *Journal of Neuroscience* 30.4 (2010), pp. 1322–1336. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.5894-08.2010.

[119] Guillaume Hennequin, Tim P. Vogels, and Wulfram Gerstner. "Optimal Control of Transient Dynamics in Balanced Networks Supports Generation of Complex Movements". In: *Neuron* 82.6 (2014), pp. 1394–1406. ISSN: 08966273. DOI: 10.1016/j.neuron.2014.04.045.

[120] Georg Heppner. *Schunk_canopen_driver - ROS Wiki*. 2016. URL: http://wiki.ros.org/schunk_canopen_driver (visited on 04/24/2020).

[121] Georg Heppner. *Schunk_svh_driver - ROS Wiki*. 2017. URL: http://wiki.ros.org/schunk_svh_driver (visited on 04/24/2020).

[122] Georg Heppner, Arne Roennau, Jan Oberländer, Sebastian Klemm, and Rüdiger Dillman. "LAUROPE - Six Legged Walking Robot for Planetary Exploration Participating in the SpaceBot Cup". In: *13th Symposium on Advanced Space Technologies in Robotics and Automation* (2015). URL: http://robotics.estec.esa.int/ASTRA/Astra2015/Papers/Session%205B/96035%5C%7B_%5C%7DHeppner.pdf.

[123] A. Hermann, J. Sun, Z. Xue, S. W. Ruehl, J. Oberlaender, A. Roennau, J. M. Zoellner, and R. Dillmann. "Hardware and Software Architecture of the Bimanual Mobile Manipulation Robot HoLLiE and Its Actuated Upper Body". In: *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. 2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). Wollongong, NSW: IEEE, 2013, pp. 286–292. ISBN: 978-1-4673-5320-5 978-1-4673-5319-9. DOI: 10.1109/AIM.2013.6584106.

[124] M. L. Hines and N. T. Carnevale. "The NEURON Simulation Environment". In: *Neural Computation* 9.6 (1997), pp. 1179–1209. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.1997.9.6.1179.

[125] Alan L Hodgkin and Andrew 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.

[126] Philip Holmes, Robert J. Full, Dan Koditschek, and John Guckenheimer. "The Dynamics of Legged Locomotion: Models, Analyses, and Challenges". In: *SIAM Review* 48.2 (2006), pp. 207–304. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/S0036144504445133.

[127] Sebastian Höppner, Yexin Yan, Andreas Dixius, Stefan Scholze, Johannes Partzsch, Marco Stolba, Florian Kelber, Bernhard Vogginger, Felix Neumärker, Georg Ellguth, Stephan Hartmann, Stefan Schiefer, Thomas Hocker, Dennis Walter, Genting Liu, Jim Garside, Steve Furber, and Christian Mayr. *The SpiNNaker 2 Processing Element Architecture for Hybrid Digital Neuromorphic Computing*. Mar. 15, 2021. URL: http://arxiv.org/abs/2103.08392 (visited on 04/27/2021).

[128] Auke Jan Ijspeert. "Central Pattern Generators for Locomotion Control in Animals and Robots: A Review". In: *Neural Networks* 21.4 (2008), pp. 642–653. ISSN: 08936080. DOI: 10.1016/j.neunet.2008.03.014.

[129] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors". In: *Neural Computation* 25.2 (2013), pp. 328–373. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00393.

[130] E.M. Izhikevich. “Simple Model of Spiking Neurons”. In: *IEEE Transactions on Neural Networks* 14.6 (2003), pp. 1569–1572. ISSN: 1045-9227. DOI: `10.1109/TNN.2003.820440`.

[131] E. Jankowska. “Spinal Interneuronal Systems: Identification, Multifunctional Character and Reconfigurations in Mammals”. In: *The Journal of Physiology* 533.1 (2001), pp. 31–40. ISSN: 00223751. DOI: `10.1111/j.1469-7793.2001.0031b.x`.

[132] Matthew S Johannes, John D Bigelow, James M Burck, Stuart D Harshbarger, Matthew V Kozlowski, and Thomas Van Doren. “An Overview of the Developmental Process for the Modular Prosthetic Limb”. In: *Johns Hopkins APL Technical Digest* (2011).

[133] Jacques Kaiser, Rainer Stal, Anand Subramoney, Arne Roennau, and Rüdiger Dillmann. “Scaling up Liquid State Machines to Predict over Address Events from Dynamic Vision Sensors”. In: *Bioinspiration & Biomimetics* 12.5 (2017), p. 055001. ISSN: 1748-3190. DOI: `10.1088/1748-3190/aa7663`.

[134] John F Kalaska. “From Intention to Action : Motor Cortex and the Control of Reaching Movements”. In: *Progress in Motor Control*. Springer, 2009, pp. 139–178. ISBN: 978-0-387-77064-2. DOI: `10.1007/978-0-387-77064-2`.

[135] Mitsuo Kawato. “Internal Models for Motor Control and Trajectory Planning”. In: *Current opinion in neurobiology* 9.6 (1999), pp. 718–727.

[136] T. Kerscher, A. Roennau, M. Ziegenmeyer, B. Gassmann, J. M. Zoellner, and R. Dillmann. “Behaviour-Based Control of the Six-Legged Walking Machine Lauron IVc”. In: *11th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*. 11th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines. Coimbra, Portugal: WORLD SCIENTIFIC, 2008, pp. 736–743. ISBN: 978-981-283-576-5 978-981-283-577-2. DOI: `10.1142/9789812835772_0089`.

[137] Seong-Min Kim, Sung-Yong Hyun, Jeong-woo Sohn, Soyong Chae, and Sung-Phil Kim. “Neural Response to Grasp of Robot Hand from M1 Area of Rhesus Monkey”. In: *2019 7th International Winter Conference on Brain-Computer Interface (BCI)*. 2019, pp. 1–4.

[138] Florence I. Kleberg, Tomoki Fukai, and Matthieu Gilson. “Excitatory and Inhibitory STDP Jointly Tune Feedforward Neural Circuits to Selectively Propagate Correlated Spiking Activity”. In: *Frontiers in Computational Neuroscience* 8 (2014). ISSN: 1662-5188. DOI: `10.3389/fncom.2014.00053`.

[139] 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`.

[140] Jens Kober and Jan Peters. “Learning Motor Primitives for Robotics”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009 IEEE International Conference on Robotics and Automation (ICRA). Kobe: IEEE, May 2009, pp. 2112–2118. ISBN: 978-1-4244-2788-8. DOI: 10.1109/ROBOT.2009.5152577.

[141] Jürgen Konczak. “On the Notion of Motor Primitives in Humans and Robots”. In: *Fifth International Workshop on Epigenetic Robotics*. Fifth International Workshop on Epigenetic Robotics. Lund University Cognitive Studies, 2005, p. 7.

[142] T. Kroger and F.M. Wahl. “Online Trajectory Generation: Basic Concepts for Instantaneous Reactions to Unforeseen Events”. In: *IEEE Transactions on Robotics* 26.1 (2010), pp. 94–111. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2009.2035744.

[143] Torsten Kröger. “Opening the Door to New Sensor-Based Robot Applications — The Reflexxes Motion Libraries”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011 IEEE International Conference on Robotics and Automation (ICRA). Shanghai, China: IEEE, 2011, pp. 1–4. ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5980578.

[144] Ioan Doré Landau, Rogelio Lozano, Mohammed M’Saad, and Alireza Karimi. *Adaptive Control: Algorithms, Analysis and Applications*. Springer Science & Business Media, 2011.

[145] Jean-Claude Latombe. *Robot Motion Planning*. Springer Science & Business Media, 2012.

[146] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. “Efficient Backprop”. In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 9–48.

[147] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. “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 (2018), pp. 421–436. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364917710318.

[148] Rafael Lorente de No. “Cerebral Cortex: Architecture, Intracortical Connections, Motor Projections”. In: *Physiology of the nervous system* (1938), pp. 288–313.

[149] N. R. Luque, J. A. Garrido, R. R. Carrillo, S. Tolu, and E. Ros. “ADAPTIVE CEREBELLAR SPIKING MODEL EMBEDDED IN THE CONTROL LOOP: CONTEXT SWITCHING AND ROBUSTNESS AGAINST NOISE”. In: *International Journal of Neural Systems* 21.05 (2011), pp. 385–401. ISSN: 0129-0657, 1793-6462. DOI: 10.1142/S0129065711002900.

[150] Niceto R. Luque, JesÃºs A. Garrido, Richard R. Carrillo, Egidio D'Angelo, and Eduardo Ros. "Fast Convergence of Learning Requires Plasticity between Inferior Olive and Deep Cerebellar Nuclei in a Manipulation Task: A Closed-Loop Robotic Simulation". In: *Frontiers in Computational Neuroscience* 8 (August 2014), pp. 1–16. ISSN: 1662-5188. DOI: `10.3389/fncom.2014.00097`.

[151] Wolfgang Maass. "Networks of Spiking Neurons: The Third Generation of Neural Network Models". In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 08936080. DOI: `10.1016/S0893-6080(97)00011-7`.

[152] Wolfgang Maass and Michael Schmitt. "On the Complexity of Learning for Spiking Neurons with Temporal Coding". In: *Information and Computation* 153.1 (1999), pp. 26–46. ISSN: 08905401. DOI: `10.1006/inco.1999.2806`.

[153] David MacNeil and Chris Eliasmith. "Fine-Tuning and the Stability of Recurrent Neural Networks". In: *PLoS ONE* 6.9 (2011). Ed. by Eleni Vasilaki, e22885. ISSN: 1932-6203. DOI: `10.1371/journal.pone.0022885`.

[154] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. "A History of Spike-Timing-Dependent Plasticity". In: *Frontiers in synaptic neuroscience* 3 (2011), p. 4.

[155] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W. Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, Guy Antoine Atenekeng Kahou, Thomas K. Berger, Ahmet Bilgili, Nenad Buncic, Athanassia Chalimourda, Giuseppe Chindemi, Jean-Denis Courcol, Fabien Delalondre, Vincent Delattre, Shaul Druckmann, Raphael Dumusc, James Dynes, Stefan Eilemann, Eyal Gal, Michael Emiel Gevaert, Jean-Pierre Ghobril, Albert Gidon, Joe W. Graham, Anirudh Gupta, Valentin Haenel, Etay Hay, Thomas Heinis, Juan B. Hernando, Michael Hines, Lida Kanari, Daniel Keller, John Kenyon, Georges Khazen, Yihwa Kim, James G. King, Zoltan Kisvarday, Pramod Kumbhar, Sébastien Lasserre, Jean-Vincent Le Bé, Bruno R.C. Magalhães, Angel Merchán-Pérez, Julie Meystre, Benjamin Roy Morrice, Jeffrey Muller, Alberto Muñoz-Céspedes, Shruti Muralidhar, Keerthan Muthurasa, Daniel Nachbaur, Taylor H. Newton, Max Nolte, Aleksandr Ovcharenko, Juan Palacios, Luis Pastor, Rodrigo Perin, Rajnish Ranjan, Imad Riachi, José-Rodrigo Rodríguez, Juan Luis Riquelme, Christian Rössert, Konstantinos Sfyrakis, Ying Shi, Julian C. Shillcock, Gilad Silberberg, Ricardo Silva, Farhan Tauheed, Martin Telefont, Maria Toledo-Rodriguez, Thomas Tränkler, Werner Van Geit, Jafet Villafranca Díaz, Richard Walker, Yun Wang, Stefano M. Zaninetta, Javier DeFelipe, Sean L. Hill, Idan Segev, and Felix Schürmann. "Reconstruction and Simulation of Neocortical Microcircuitry". In: *Cell* 163.2 (2015), pp. 456–492. ISSN: 00928674. DOI: `10.1016/j.cell.2015.09.029`.

[156] Timothée Masquelier, Rudy Guyonneau, and Simon J. Thorpe. "Spike Timing Dependent Plasticity Finds the Start of Repeating Patterns in Continuous

Spike Trains". In: *PLoS ONE* 3.1 (2008). Ed. by Olaf Sporns, e1377. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0001377.

[157] Amirhossein H Memar and Ehsan T Esfahani. "Modeling and Dynamic Parameter Identification of the Schunk Powerball Robotic Arm". In: *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. 2015.

[158] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes von Hofsten, Kerstin Rosander, Manuel Lopes, José Santos-Victor, Alexandre Bernardino, and Luis Montesano. "The iCub Humanoid Robot: An Open-Systems Platform for Research in Cognitive Development". In: *Neural Networks* 23.8-9 (2010), pp. 1125–1134. ISSN: 08936080. DOI: 10.1016/j.neunet.2010.08.010.

[159] Andrew T Miller and Peter K Allen. "Graspit! A Versatile Simulator for Robotic Grasping". In: *IEEE Robotics & Automation Magazine* 11.4 (2004), pp. 110–122.

[160] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. "A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)". In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1 (Feb. 2018), pp. 106–122. ISSN: 1932-4545, 1940-9990. DOI: 10.1109/TBCAS.2017.2759700.

[161] John Nassour, Patrick Hénaff, Fethi Benouezdou, and Gordon Cheng. "Multi-Layered Multi-Pattern CPG for Adaptive Locomotion of Humanoid Robots". In: *Biological Cybernetics* 108.3 (2014), pp. 291–303. ISSN: 0340-1200, 1432-0770. DOI: 10.1007/s00422-014-0592-8.

[162] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63. ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2019.2931595.

[163] Nengo. *NEF Summary - Nengo.Ai*. 2020. URL: https://www.nengo.ai/nengo/examples/advanced/nef-summary.html (visited on 04/27/2020).

[164] Erhan Oztop and Michael A. Arbib. "Schema Design and Implementation of the Grasp-Related Mirror Neuron System". In: *Biological Cybernetics* 87.2 (2002), pp. 116–140. ISSN: 03401200. DOI: 10.1007/s00422-002-0318-1.

[165] Erhan Oztop, Nina S. Bradley, and Michael A. Arbib. "Infant Grasp Learning: A Computational Model". In: *Experimental Brain Research* 158.4 (2004), pp. 480–503. ISSN: 00144819. DOI: 10.1007/s00221-004-1914-1.

[166] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "Pytorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. 2019, pp. 8026–8037.

[167] Hélène Paugam-Moisy and Sander Bohte. “Computing with Spiking Neuron Networks”. In: *Handbook of Natural Computing*. Ed. by Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 335–376. ISBN: 978-3-540-92909-3. DOI: 10.1007/978-3-540-92910-9_10.

[168] Wilder Penfield and Edwin Boldrey. “SOMATIC MOTOR AND SENSORY REPRESENTATION IN THE CEREBRAL CORTEX OF MAN AS STUDIED BY ELECTRICAL STIMULATION”. In: *Brain* 60.4 (1937), pp. 389–443. ISSN: 0006-8950, 1460-2156. DOI: 10.1093/brain/60.4.389.

[169] Rolf Pfeifer and Josh Bongard. *How the Body Shapes the Way We Think: A New View of Intelligence*. MIT press, 2006.

[170] Rolf Pfeifer, Max Lungarella, and Fumiya Iida. “The Challenges Ahead for Bio-Inspired ’soft’ Robotics”. In: *Communications of the ACM* 55.11 (Nov. 2012), pp. 76–87. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/2366316.2366335.

[171] Thomas Pfeil, Andreas Grübl, Sebastian Jeltsch, Eric Müller, Paul Müller, Mihai A. Petrovici, Michael Schmuker, Daniel Brüderle, Johannes Schemmel, and Karlheinz Meier. “Six Networks on a Universal Neuromorphic Computing Substrate”. In: *Frontiers in Neuroscience* 7 (2013). ISSN: 1662-4548. DOI: 10.3389/fnins.2013.00011.

[172] J.-P. Pfister and Wulfram Gerstner. “Triplets of Spikes in a Model of Spike Timing-Dependent Plasticity”. In: *Journal of Neuroscience* (2006). DOI: 10.1523/JNEUROSCI.1425-06.2006.

[173] Filip Ponulak and Andrzej Kasinski. “Introduction to Spiking Neural Networks: Information Processing, Learning and Applications.” In: *Acta neurobiologiae experimentalis* 71.4 (2011), pp. 409–433.

[174] Isabella Pozzi, Sander M Bohté, and Pieter R Roelfsema. “Attention-Gated Brain Propagation: How the Brain Can Implement Reward-Based Error Backpropagation”. In: (2020), p. 11.

[175] Dimitri Probst, Wolfgang Maass, Henry Markram, and Marc-Oliver Gewaltig. “Liquid Computing in a Simplified Model of Cortical Layer IV: Learning to Balance a Ball”. In: *International Conference on Artificial Neural Networks ICANN*. 2012, pp. 209–216.

[176] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. “ROS: An Open-Source Robot Operating System”. In: *ICRA Workshop on Open Source Software*. Vol. 3. 3.2. 2009, p. 5.

[177] Joel Reithler, Hanneke I. van Mier, Judith C. Peters, and Rainer Goebel. “Nonvisual Motor Learning Influences Abstract Action Observation”. In: *Current Biology* 17.14 (July 2007), pp. 1201–1207. ISSN: 09609822. DOI: 10.1016/j.cub.2007.06.019.

[178] Oliver Rhodes, Luca Peres, Andrew GD Rowley, Andrew Gait, Luis A Plana, Christian Brenninkmeijer, and Steve B Furber. “Real-Time Cortical Simulation on Neuromorphic Hardware”. In: *Philosophical Transactions of the Royal Society A* 378.2164 (2020), p. 20190160.

[179] Mathis Richter, Yulia Sandamirskaya, and Gregor Schoner. “A Robotic Architecture for Action Selection and Behavioral Organization Inspired by Human Cognition”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012). Vilamoura-Algarve, Portugal: IEEE, 2012, pp. 2457–2464. ISBN: 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. DOI: 10.1109/IROS.2012.6386153.

[180] A. Roennau, G. Heppner, M. Nowicki, and R. Dillmann. “LAURON V: A Versatile Six-Legged Walking Robot with Advanced Maneuverability”. In: *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. 2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). Besacon: IEEE, 2014, pp. 82–87. ISBN: 978-1-4799-5736-1. DOI: 10.1109/AIM.2014.6878051.

[181] A. Roennau, G. Heppner, L. Pfotzer, and R. Dillmann. “Lauron v: Optimized Leg Configuration for the Design of a Bio-Inspired Walking Robot”. In: *Nature-Inspired Mobile Robotics*. World Scientific, 2013, pp. 563–570. DOI: 10.1142/9789814525534_0071.

[182] Elmar Rückert and Andrea D’Avella. “Learned Parametrized Dynamic Movement Primitives with Shared Synergies for Controlling Robotic and Musculoskeletal Systems”. In: *Frontiers in Computational Neuroscience* 7 (October 2013), pp. 1–18. ISSN: 1662-5188. DOI: 10.3389/fncom.2013.00138.

[183] Steffen W. Ruehl, Christoper Parlitz, Georg Heppner, Andreas Hermann, Arne Roennau, and Ruediger Dillmann. “Experimental Evaluation of the Schunk 5-Finger Gripping Hand for Grasping Tasks”. In: *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO). Bali, Indonesia: IEEE, 2014, pp. 2465–2470. ISBN: 978-1-4799-7397-2. DOI: 10.1109/ROBIO.2014.7090710.

[184] Abigail A. Russo, Sean R. Bittner, Sean M. Perkins, Jeffrey S. Seely, Brian M. London, Antonio H. Lara, Andrew Miri, Najja J. Marshall, Adam Kohn, Thomas M. Jessell, Laurence F. Abbott, John P. Cunningham, and Mark M. Churchland. “Motor Cortex Embeds Muscle-like Commands in an Untangled Population Response”. In: *Neuron* 97.4 (2018), pp. 953–966. ISSN: 08966273. DOI: 10.1016/j.neuron.2018.01.004.

[185] Philippe Saltiel, Matthew C. Tresch, and Emilio Bizzi. “Spinal Cord Modular Organization and Rhythm Generation: An NMDA Iontophoretic Study in the Frog”. In: *Journal of Neurophysiology* 80.5 (1998), pp. 2323–2339. ISSN: 0022-3077, 1522-1598. DOI: 10.1152/jn.1998.80.5.2323.

[186] M Santello, M Flanders, and J F Soechting. “Postural Hand Synergies for Tool Use”. In: *Journal of Neuroscience* 18.23 (1998), pp. 10105–10115. ISSN: 0270-6474. DOI: citeulike-article-id:423192.

[187] Cosimo Della Santina, Visar Arapi, Giuseppe Averta, Francesca Damiani, Gaia Fiore, Alessandro Settimi, Manuel G. Catalano, Davide Bacciu, Antonio Bicchi, and Matteo Bianchi. “Learning From Humans How to Grasp: A Data-Driven Architecture for Autonomous Grasping With Anthropomorphic Soft Hands”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1533–1540. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2019.2896485.

[188] Jeffrey A. Saunders and David C. Knill. “Humans Use Continuous Visual Feedback from the Hand to Control Fast Reaching Movements”. In: *Experimental Brain Research* 152.3 (2003), pp. 341–352. ISSN: 0014-4819, 1432-1106. DOI: 10.1007/s00221-003-1525-2.

[189] Andreea I Sburlea and Gernot R Müller-Putz. “Exploring Representations of Human Grasping in Neural, Muscle and Kinematic Signals”. In: *Scientific reports* 8.1 (2018), pp. 1–14.

[190] Alessandro Scano, Andrea Chiavenna, Lorenzo Molinari Tosatti, Henning Müller, and Manfredo Atzori. “Muscle Synergy Analysis of a Hand-Grasp Dataset: A Limited Subset of Motor Modules May Underlie a Large Variety of Grasps”. In: *Frontiers in neurorobotics* 12 (2018), p. 57.

[191] Stefan Schaal. “Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics”. In: *Adaptive Motion of Animals and Machines*. Ed. by Hiroshi Kimura, Kazuo Tsuchiya, Akio Ishiguro, and Hartmut Witte. Tokyo: Springer-Verlag, 2006, pp. 261–280. ISBN: 978-4-431-24164-5. DOI: 10.1007/4-431-31381-8_23.

[192] Stefan Scherzinger, Arne Roennau, and Rudiger Dillmann. “Forward Dynamics Compliance Control (FDCC): A New Approach to Cartesian Compliance for Robotic Manipulators”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Vancouver, BC: IEEE, 2017, pp. 4568–4575. ISBN: 9781538626818. DOI: 10.1109/IROS.2017.8206325.

[193] Mathieu Schiess, Robert Urbanczik, and Walter Senn. “Somato-Dendritic Synaptic Plasticity and Error-Backpropagation in Active Dendrites”. In: *PLOS Computational Biology* 12.2 (2016). Ed. by Boris S. Gutkin, e1004638. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1004638.

[194] Schunk. *Schunk SVH 5-Finger Anthropomorphic Hand*. 2020. URL: https://schunk.com/de_en/gripping-systems/series/svh/.

[195] Stephen H. Scott. “The Computational and Neural Basis of Voluntary Motor Control and Planning”. In: *Trends in Cognitive Sciences* 16.11 (2012), pp. 541–549. ISSN: 13646613. DOI: 10.1016/j.tics.2012.09.008.

[196] Mario Senden, Judith Peters, Florian Röhrbein, Gustavo Deco, and Rainer Goebel. "Editorial: The Embodied Brain: Computational Mechanisms of Integrated Sensorimotor Interactions With a Dynamic Environment". In: *Frontiers in Computational Neuroscience* 14 (June 18, 2020), p. 53. ISSN: 1662-5188. DOI: 10.3389/fncom.2020.00053.

[197] ShadowRobot. *ShadowRobot Dextereous Hand*. 2020. URL: https://www.shadowrobot.com/products/dexterous-hand/.

[198] Krishna V. Shenoy, Maneesh Sahani, and Mark M. Churchland. "Cortical Control of Arm Movements: A Dynamical Systems Perspective". In: *Annual Review of Neuroscience* 36.1 (2013), pp. 337–359. ISSN: 0147-006X, 1545-4126. DOI: 10.1146/annurev-neuro-062111-150509.

[199] Charles Scott Sherrington. "Flexion-Reflex of the Limb, Crossed Extension-Reflex, and Reflex Stepping and Standing". In: *The Journal of physiology* 40.1-2 (1910), p. 28.

[200] Satoshi Shigemi. "ASIMO and Humanoid Robot Research at Honda". In: *Humanoid Robotics: A Reference*. Ed. by Ambarish Goswami and Prahlad Vadakkepat. Dordrecht: Springer Netherlands, 2019, pp. 55–90. ISBN: 978-94-007-6046-2. DOI: 10.1007/978-94-007-6046-2_9.

[201] S Song, K D Miller, and L F Abbott. "Competitive Hebbian Learning through Spike-Timing-Dependent Synaptic Plasticity." In: *Nature neuroscience* 3.9 (2000), pp. 919–926. DOI: 10.1038/78829.

[202] Devarajan Sridharan and Eric I. Knudsen. "Selective Disinhibition: A Unified Neural Mechanism for Predictive and Post Hoc Attentional Selection". In: *Vision Research* 116 (2015), pp. 194–209. ISSN: 00426989. DOI: 10.1016/j.visres.2014.12.010.

[203] N. Srinivasa and Youngkwan Cho. "Self-Organizing Spiking Neural Model for Learning Fault-Tolerant Spatio-Motor Transformations". In: *IEEE Transactions on Neural Networks and Learning Systems* 23.10 (2012), pp. 1526–1538. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2012.2207738.

[204] Julia Starke, Christian Eichmann, Simon Ottenhaus, and Tamim Asfour. "Human-Inspired Representation of Object-Specific Grasps for Anthropomorphic Hands". In: *International Journal of Humanoid Robotics* 17.2 (2020), p. 2050008. ISSN: 0219-8436, 1793-6942. DOI: 10.1142/S0219843620500085.

[205] Lea Steffen, Daniel Reichard, Jakob Weinland, Jacques Kaiser, Arne Roennau, and Rüdiger Dillmann. "Neuromorphic Stereo Vision: A Survey of Bio-Inspired Sensors and Algorithms". In: *Frontiers in Neurorobotics* 13 (2019), p. 28. ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00028.

[206] Lea Steffen, Stefan Ulbrich, Arne Roennau, and Rüdiger Dillmann. "Multi-View 3D Reconstruction with Self-Organizing Maps on Event-Based Data". In: *2019 19th International Conference on Advanced Robotics (ICAR)*. 2019, pp. 501–508.

[207] Terrence C Stewart. "A Technical Overview of the Neural Engineering Framework". In: *Centre for Theoretical Neuroscience technical report* (2012).

[208] Emma Strubell, Ananya Ganesh, and Andrew McCallum. *Energy and Policy Considerations for Deep Learning in NLP*. 2019.

[209] Anand Subramoney, Guillaume Bellec, Franz Scherr, Robert Legenstein, and Wolfgang Maass. *Revisiting the Role of Synaptic Plasticity and Network Dynamics for Fast Learning in Spiking Neural Networks*. preprint. Neuroscience, Jan. 27, 2021. DOI: 10.1101/2021.01.25.428153.

[210] Johan AK Suykens, Joos PL Vandewalle, and Bart L de Moor. *Artificial Neural Networks for Modelling and Control of Non-Linear Systems*. Springer Science & Business Media, 2012.

[211] Francesco Tenore, Ander Ramos, Amir Fahmy, Soumyadipta Acharya, Ralph Etienne-Cummings, and Nitish V Thakor. "Towards the Control of Individual Fingers of a Prosthetic Hand Using Surface EMG Signals". In: *EMBC*. 2007.

[212] ThalmicLabs. *Myo Diagnostics*. 2019. URL: http://diagnostics.myo.com/.

[213] Thomas George Thuruthel, Syed Haider Abidi, Matteo Cianchetti, Cecilia Laschi, and Egidio Falotico. *A Bistable Soft Gripper with Mechanically Embedded Sensing and Actuation for Fast Closed-Loop Grasping*. 2019. DOI: 10.13140/RG.2.2.22701.13280.

[214] Silvia Tolu, Mauricio Vanegas, Jesus A. Garrido, Niceto R. Luque, and Eduardo Ros. "Adaptive and Predictive Control of a Simulated Robot Arm". In: *International Journal of Neural Systems* 23.3 (2013). ISSN: 0129-0657. DOI: 10.1142/S012906571350010X.

[215] Silvia Tolu, Mauricio Vanegas, Niceto R. Luque, Jesús A. Garrido, and Eduardo Ros. "Bio-Inspired Adaptive Feedback Error Learning Architecture for Motor Control". In: *Biological Cybernetics* 106.8-9 (2012), pp. 507–522. ISSN: 0340-1200, 1432-0770. DOI: 10.1007/s00422-012-0515-5.

[216] Klaus Uhl and Marco Ziegenmeyer. "Mca2 - an Extensible Modular Framework for Robot Control Applications". In: *Advances in Climbing and Walking Robots*. World Scientific, 2008, pp. 680–689. DOI: 10.1142/9789812770189_0078.

[217] Gabriel Urbain, Victor Barasuol, Claudio Semini, Joni Dambre, et al. *Stance Control Inspired by Cerebellum Stabilizes Reflex-Based Locomotion on HyQ Robot*. 2020.

[218] Sacha J. van Albada, Andrew G. Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B. Stokes, David R. Lester, Markus Diesmann, and Steve B. Furber. “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 (May 23, 2018), p. 291. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00291.

[219] Alexander Vandesompele, Gabriel Urbain, Hossain Mahmud, Joni Dambre, et al. “Body Randomization Reduces the Sim-to-Real Gap for Compliant Quadruped Locomotion”. In: *Frontiers in neurorobotics* 13 (2019), p. 9.

[220] Jilles Vreeken. “Spiking Neural Networks, an Introduction”. In: *Utrecht University: Information and Computing Sciences* 7.3 (2003), pp. 1–5. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.12.7093%5C%7B&%5C%7Drep=rep1%5C%7B&%5C%7Dtype=pdf.

[221] Donald M. Wilson. “Insect Walking”. In: *Annual Review of Entomology* 11.1 (1966), pp. 103–122. ISSN: 0066-4170. DOI: 10.1146/annurev.en.11.010166.000535.

[222] Andreas Wolf and Henrik A Schunk. *Grippers in Motion: The Fascination of Automated Handling Tasks*. Carl Hanser Verlag GmbH Co KG, 2018.

[223] Daniel M Wolpert, R Chris Miall, and Mitsuo Kawato. “Internal Models in the Cerebellum”. In: *Trends in cognitive sciences* 2.9 (1998), pp. 338–347. URL: http://discovery.ucl.ac.uk/189113/.

[224] Si Wu, Shun-ichi Amari, and Hiroyuki Nakahara. “Population Coding and Decoding in a Neural Field: A Computational Study”. In: *Neural Computation* 14.5 (2002), pp. 999–1026. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976602753633367.

[225] Yexin Yan, Terrence Stewart, Xuan Choo, Bernhard Vogginger, Johannes Partzsch, Sebastian Hoeppner, Florian Kelber, Chris Eliasmith, Steve Furber, and Christian Mayr. “Comparing Loihi with a SpiNNaker 2 Prototype on Low-Latency Keyword Spotting and Adaptive Robotic Control”. In: *Neuromorphic Computing and Engineering* (Mar. 24, 2021). ISSN: 2634-4386. DOI: 10.1088/2634-4386/abf150.

[226] Nianfeng Yang, Ming Zhang, Changhua Huang, and Dewen Jin. “Synergic Analysis of Upper Limb Target-Reaching Movements”. In: *Journal of Biomechanics* 35.6 (2002), pp. 739–746. ISSN: 00219290. DOI: 10.1016/S0021-9290(02)00018-0.

[227] Alper Yegenoglu, Detlef Holstein, Long Duc Phan, Michael Denker, Andrew Davison, and Sonja Grün. *Elephant–Open-Source Tool for the Analysis of Electrophysiological Data Sets*. Computational and Systems Neuroscience, 2015.

[228] Davide Zambrano, Roeland Nusselder, H. Steven Scholte, and Sander Bohte. *Efficient Computation in Adaptive Artificial Spiking Neural Networks*. 2017. URL: http://arxiv.org/abs/1710.04838 (visited on 05/31/2020).

[229] Friedemann Zenke, Sander M. Bohté, Claudia Clopath, Iulia M. Comşa, Julian Göltz, Wolfgang Maass, Timothée Masquelier, Richard Naud, Emre O. Neftci, Mihai A. Petrovici, Franz Scherr, and Dan F.M. Goodman. "Visualizing a Joint Future of Neuroscience and Neuromorphic Engineering". In: *Neuron* 109.4 (Feb. 2021), pp. 571–575. ISSN: 08966273. DOI: `10.1016/j.neuron.2021.01.009`.

[230] John G Ziegler and Nathaniel B Nichols. "Optimum Settings for Automatic Controllers". In: *trans. ASME* 64.11 (1942).