TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Robotik, Künstliche Intelligenz und Echtzeitsysteme

Biological-inspired Hierarchical Control of a Snake-like Robot for Autonomous Locomotion

Zhenshan Bing

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

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Thomas Huckle
Prüfer der Dissertation:	1. Prof. Dr.-Ing. habil. Alois Knoll
	2. Prof. Dr. Kai Huang

Die Dissertation wurde am 09.01.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 29.04.2019 angenommen.

Abstract

To acquire similar intelligence and abilities as living creatures and be operated within the real world, robots need to perceive their environment via sensors, make immediate action decisions, and possess excellent locomotion skills to interact with the environment. As their counterparts in nature, animals exhibit distinct abilities and outperform state-of-the-art robots in almost every aspect of life. These characteristics always find their roots in the advanced neural systems of many animals, where everything is sophisticatedly processed according to different function purposes and exhibits a hierarchical organization of the motor system sharing surprising similarities. However, investigating the underlying mechanism of the neural circuit and model it to control locomotion is by no means a trivial problem since it involves the interaction of the brain, body, and their environment in closed perception-action loops.

In order to mimic both structural and functional principles in living creatures, this thesis presents a biological-inspired hierarchical control architecture and takes a snake-like robot as an agent to perform autonomous locomotion tasks. The main point of the control architecture is to offer an efficient way not only to model the principles underlying neural structures devoted to locomotion control, but more importantly to implement them to control autonomous field robots. This control architecture consists of two parts: a high-level SNN (Spiking Neural Network) sub-controller that is responsible for processing sensor information and making decisions; a low-level CPG (Central Pattern Generator) sub-controller that generates joint commands for desired actions. The SNN sub-controller takes spike-encoded information from sensors, learns knowledge based on dopamine modulated STDP (Spiking-Timing-Dependent Plasticity), and generates spike-decoded action commands for the CPG sub-controller. The CPG sub-controller consists of mutual connected neurons that directly receive action decision from the SNN and generate joint commands for motors. Meanwhile in the context of this thesis, snake-like robots are studied to demonstrate the performances of the proposed controller and their slithering gait is utilized to perform three different autonomous locomotion tasks.

Zusammenfassung

Um in der realen Welt zurechtzukommen, müssen Roboter ähnlich intelligent und fähig sein wie Lebewesen. Dies erfordert, dass sie ihre Umwelt mittels Sensoren wahrnehmen, sich in Echtzeit für eine Handlung entscheiden und über hervorragende Fortbewegungsmittel verfügen mit denen sie mit ihrer Umgebung interagieren können. Das Tier, als natürliches Gegenstück des Roboters, weist ausgeprägte Fähigkeiten auf und übertrifft sogar die modernsten Roboter in nahezu jeder Disziplin. Der Urspring hierfür ist immer das hochentwickelte Nervenstystem des Tiers, das alle Informationen - im Bezug zur erforderlichen Funktion - differenziert verarbeitet. Diese Systeme verfügen über eine hierarchische Oranisation des Bewegungsapparats und weisen untereinander erstaunliche Ähnlichkeiten auf. Nichtsdestotrotz ist die Analyse des zugrundeliegenden Mechanismus des neuronalen Netzes, sowie dessen Übetragung in ein Modell für Bewegungskontrolle alles andere als trivial. Der Mechanismus besteht aus der Interkation des Gehirns, des Körpers und der Umgebung in einer geschlossenen Wahrnehmungs-Handlungs-Schleife. In dieser Arbeit wird eine biologisch inspirierte, hierarchische Steuerungsarchitektur vorgestellt mit der ein schlangenähnlicher Roboter autonome Fortbewegungsaufgaben löst. Dies wird durch die Nachahmung von sowohl den strukturellen, sowie den funktionalen Prinzipien von Lebewesen ermöglicht. Der Hauptgrund für diese Kontrollarchitektur ist die daraus resultierende effiziente Modellierung der Prinzipien von neuronalen Strukturen für Bewegungskontrolle, sowie die weitaus bedeutendere Implementierung hiervon in autonomen Robotern. Die Architektur besteht aus zwei Teilen: auf hoher Ebene ist ein SNN (Spiking Neural Network) Sub-controller für die Verarbeitung von Sensordaten und Handlungsentscheidungen verantwortlich, während auf niedriger Ebene ein CPG (Central Pattern Generator) Sub-controller zuständig für die Gelenkkommandos und somit für die Bewegung des Roboters ist. Der SNN Sub-controller nimmt impulskodierte Information von Sensoren auf, lernt mittels Dopamin-moduliertem STDP (Spike-Timing-Dependent-Plasticity), und erzeugt wiederum impulskodierte Kommandos für den CPG Sub-controller. Der CPG Sub-controller besteht aus gegenseitig verbun-

denen Neuronen, die direkt Handlungsentscheidungen von dem SNN bekommen und daraus Gelenkkommandos für die Motoren generieren. Gleichzeitig werden im Rahmen dieser Dissertation schlangenähnliche Roboter untersucht, um die Leistung des vorgestellen Controllers zu demonstrieren. Desweiteren wird die Schlangengangart für drei verschiedene autonome Fortbewegungsaufgaben verwendet.

Acknowledgements

First and foremost, I would like to express my sincere gratitude and appreciation to Prof. Dr. Alois Knoll for offering the opportunity to pursue my doctorate under his supervision. He has been an incredible academic guider that enlightened and encouraged my research topics constantly.

I would like to thank Prof. Dr. Kai Huang for being my mentor and providing me valuable suggestions and solid support about my research.

I would like to thank to Dr. Long Cheng for his help in my research and cooperation in the snake robot project; Xiebing Wang, Mingchuan Zhou, Xiang Gao, Zhuangyi Jiang for their companion; Prof. Dr. Guang Chen, Prof. Feihuang Zhang, Dr. Florian Röhrbein, Dr. Caixia Cai, Dr. Biao Hu and Prof. Gang Chen for their academic advices and collaboration; all the bachelor or master students advised by me for their cooperation during my whole Ph.D. career. Furthermore, my special thanks to Amy Bücherl, Ute Lomp, those students conducting their thesis with me, and all my former and current colleagues of the whole Robotics, Artificial Intelligence and Real-time Systems chair for their help, support and company.

Finally, my dearest thanks go to my parents Xinbing Bing and Meiyu Song for their love, care and support from my childhood. My girlfriend Yonghuan for her persistent companion, understanding and love. Frau Petra Mielich for her care in the two years we shared together.

The work presented in this thesis was partly supported by the China Scholarship Council (grant number: 201506120042). This support is gratefully acknowledged.

Contents

List of Figures

LIST OF FIGURES

List of Tables

Chapter 1

Introduction

Sophisticated robotic intelligence has the great potential to take over more and more tasks that were used to be carried out only by biological intelligence in every aspect of life. As the core and foundation for bionic robots, achieving excellent locomotion ability is always regarded as the most challenging control tasks. Looking at the animal kingdom, the ability to move is essential to all their members and has been deeply rooted in their evolution process of both nervous and physical systems. Therefore, the problem of locomotion control for mobile robots can be regarded as taking inspiration from living creatures and then apply their working principles to mimic similar capabilities.

For all kinds of bionic robots, the locomotion control is never a trivial task when it comes to the perception, decision-making, and interaction with their environment. However, even simple animals appear to manage their locomotion skills exceptional well with extreme adaptabilities in complex and challenging environment. With longtime research and understanding, it is even more surprising to observe how a universal paradigm for neural mechanisms underlying diverse locomotion emerges from studies performed on very different animals with very different forms of motion: The slow aquatic swimming of the a marine mollusca, and the terrestrial perfect running of a cheetah; the amphibious undulation movements of a snake, and the aerial flight of a bird, are all regulated by neural motor systems whose structures share surprising similarities [1]. All these examples exhibit a hierarchical organization of the motor systems that consists of three parts: a high-level brain controller that is responsible for information processing and decision making, a low-level spinal-cord controller to transfer neural activities to motor commands, and the motor actuation components such as fins, legs, and wings (See Figure 1.1). These three factors consist of the main research topics of this work and will be investigated in greater detail.

Typically solutions to the aforementioned aspects refer to the control algorithms and the gait design in robotic domain. For robotic control algorithms, traditional model-based control via the

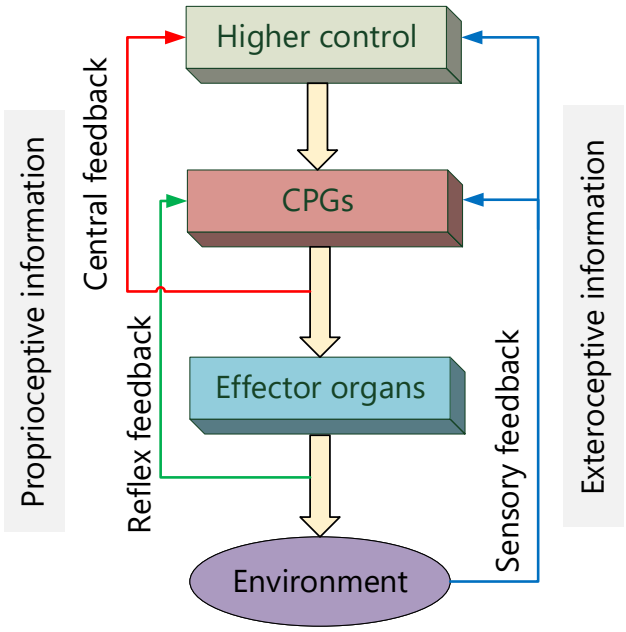


Figure 1.1: The main functional features of the motor system reduced from many animals. Adapted from [2].

kinematics and dynamics of a robot and the geometrics of the environment has always been limited to a situation that all the key information is known and all the unpredictable constrains are overly simplified. Therefore, this kind of control strategy is not well suited to solve complex locomotion tasks where the robot is too complicated to be modeled and the environment conditions are hard to forecast, let along maintain high stability and adaptability. To deal with these challenges, artificial neural networks (ANNs) emerged and have achieved amazing intelligence that can process natural language [3], analysis and recognize images [4], and outperform human intelligence in complicated mind sports [5]. However, they basically learn specific tasks relying on massive computing resources, lengthy time, and regardless power consumption and meanwhile bring significant latency [6]. In self-driving cars for example, the overall computation consumes a few thousand watts compared to the human brain, which only needs around 20 watts of power [7]. These are considerable disadvantages, especially in mobile applications where real-time responses are important and energy supply is limited.

Furthermore, robots have to maintain some vital activities with no need for taking descending high-level control signal, such as balance ability and some rhythmic actions. In most cases, this issue has been neglected but essential for wild applications where the stability is particularly required. On the one hand, this can ensure the robot stability and remain at a satisfied stats once there is

some fatal problems of the brain level controller. On the other hand, this function can lead to power efficiency, since there is no need to keep running heavyweight energy-intensive controllers all the time if there is no further information to be processed or the robot is just required to sustain its action forms once the locomotion is initiated by the signal from higher control.

In order to execute specific movement and obtain good locomotion skills, robots are normally designed to mimic the structural functions of their counterparts in nature. However, robots usually lack of enough degrees of freedom and as flexible as biological muscles. Besides, robots also have to consider how to incorporate essential sensors to their locomotion skills, which are necessary to acquire inner and outer information and this usually bring more difficulties to control them.

To sum up, this thesis attempts to propose a hierarchically universal control paradigm that reveal the underlying neural mechanisms of locomotion and use it to perform autonomous locomotion tasks with bionic mobile robots.

1.1 Motivations

Considering all the aforementioned perspectives involved to perform autonomous locomotion tasks with a bionic mobile robot, our motivations in this thesis are to present how it can be generally solved and give out some implementations. Specially, the motivations are three-fold as follows.

1.1.1 Why Spiking Neural Networks?

In this thesis, spiking neural networks (SNNs) are chosen as our high-level controller to process information and make a sequence of action decisions accordingly. For living creatures, biological neurons use impulses or spikes to process information and communicate, making seemingly simple organisms able to perceive and act in the real world exceptionally well and outperform state-of-the-art robots in almost every aspect of life. While computations within conventional ANNs can be understood as approximation of the impulse rate over time, event-based networks or spiking neural networks perform computations based on individual spikes taking the precise timing into account as well. Therefore, SNNs are always been introduced as the third generation of ANNs [8].

Due to their functional similarity to the brain, SNNs have the capabilities for processing information and learning in a much better fashion, both in terms of energy and data, e.g. building large-scale brain model [9] or using neurally inspired hardware such as the SpiNNaker board [10] or Dynamic Vision Sensors (DVS) [11]. Moreover, SNNs have offered solutions to a broad range of specific implementations, such as fast signal-processing [12], speech recognition [13], robot navigation [14],

and other problems solved by non-spiking neural networks. A comprehensive review on controlling robots based on spiking neural networks will be discussed in the following chapter.

1.1.2 Why Central Pattern Generators?

In this thesis, central pattern generators (CPGs) are chosen as our low-level controller that receive descending command from SNNs and carry out the joint position commands for desired action. CPGs, as the key component of the motor system that usually located in the spinal cord of vertebrates or in relevant ganglia in invertebrates, is a neural circuit that can produce a rhythmic motor pattern even without need for sensory feedback or high-level descending control [2]. In fact, many biology experiments have demonstrated that CPGs receive essential signals from high-level neurons to initiate the locomotion and directly generate the commands to control effector organs [2].

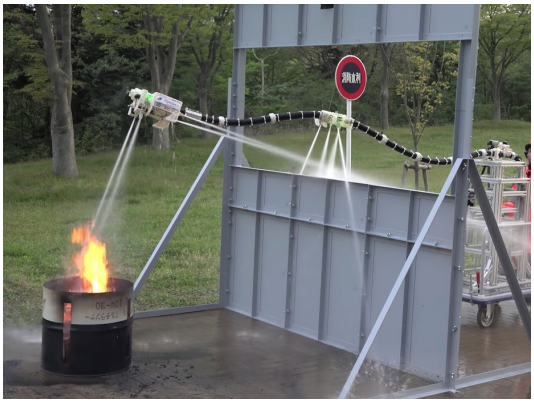
In the domain of robotics control, due to the capability of self modulation, CPGs offer an ideal candidate for practical engineering solutions to for locomotion control of robots, especially those with multiple joints or DOFs and even of hyper-redundant robots. More deep review about the CPG-inspired robotic control methods can be found in [15, 16].

1.1.3 Why Snake-like Robots?

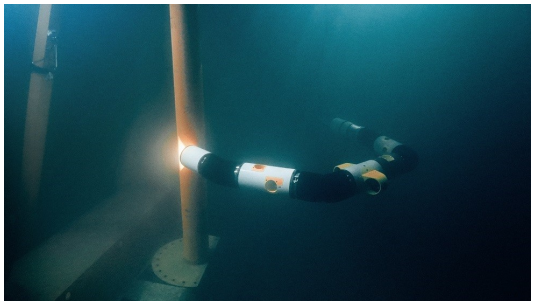
In order to examine the performance of the control architecture, snake-like robots are taken as an example to carry out different autonomous locomotion tasks in this work. Inspired by their skillful locomotion abilities of real snakes, snake-like robots carry the potential of being one kind of promising mobile robotic applications capable of traveling and performing in diverse and challenging environments, such as disaster rescue, underwater exploration, and industrial inspection [17].

Nowadays, snake-like robots are being studied for practical implementations in many fields as shown in Figure 1.2. For example, the snake robot designed from Carnegie Mellon University (CMU) was tentatively implemented in the Mexican earthquake ruins and tired to rescue survivors in 2017 [18]. A flying snake from Tohoku University can use steerable jets of water like rockets and fly into burning buildings to extinguish fires [19]. Like other snake-like robots, this one has the potential to be able to wiggle its way into windows or other gaps in a structure, with the benefit of carrying and directing water as it goes. The snake-like robots from Norwegian University of Science and Technology (NTNU) were designed for underwater manipulation and inspection [20]. Snake-like robots are also deployed for military and public safety tasks, such as inspection, surveillance, and disabling bombs. Anyway, their inherent redundant bodies provide them excellent traversability in irregular terrain that surpasses the mobility of conventional mobile robots such as wheeled, tracked, and

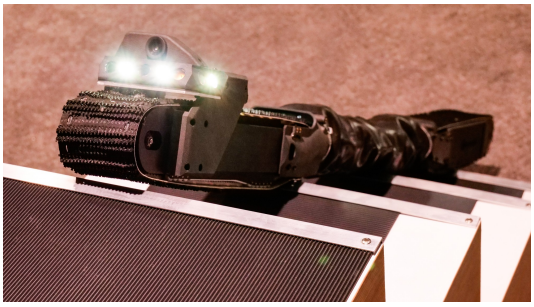
(a) Earthquake scene.



(b) Firefight scene



(c) Underwater inspection scene



(d) Counter-terrorism scene

Figure 1.2: (a). Mexican Red Cross worker holds the snake robot in preparation for a search mission at a collapsed building, image courtesy of Carnegie Mellon University. **(b)**. Fire fighting snake-like robot, image courtesy of Tohoku University/National Institute of Technology. **(c)**. Underwater snake-like robot for inspection scene, image courtesy of Norwegian University of Science and Technology. **(d)**. Sarcos Guardian snake-like robot for anti-terrorist scene, image courtesy of Sarcos Robotics.

legged robots, but meanwhile make them difficult to be modeled or controlled precisely.

1.2 Challenges

Although state-of-the-art locomotion control tasks cover diverse methods according to different kinds of bio-robotics, there still lacks of a universal paradigm to perform autonomous locomotion from the sensory end to the action end. To mimic the structures and functionalities of the motor system in the nervous system, this paradigm does not only need to assemble all the existing solutions properly, but also new challenges have to be settled. We identify a few new challenges as follows:

- **Spiking neural networks:** Since SNNs use non-differential spikes to communicate, the well-known error back-propagation mechanism is no longer applicable for training SNNs. Also for

the same reason, there has been a void of practical learning rules for mobile applications [21]. On the one hand, it is intricate to encode sensory information into spikes or decode motor commands from spikes based on different strategies. On the other hand, using efficient SNN learning rules for practical mobile applications is still being explored. The questions are:

How to effectively encode sensory information from spikes and decode motor commands from spikes?
How to generally train an SNN for mobile robotic applications?

- **Central pattern generators:** Even CPGs have been investigated and applied to control the locomotion of robotics, a lightweight CPG model is still in need to run on microprocessors with limited computing power [22]. On the basis of the CPG modulation functionalities, a smooth gait transition process need to be explored to ensure minimal undesired movement and reduce the torque produced by sudden jump of joint positions [23]. Therefore, the questions are:

  How to design a CPG model that is fast-oriented but meanwhile still maintains satisfied accuracy?
  Can such a CPG controller contribute to smooth the gait transition process?

- **Locomotion of snake-like robots:** In this work, the snake-like robot is taken as the agent to perform autonomous locomotion tasks. However, the serpentine locomotion of a snake-like robot is rather non-trivial due to its redundant degree of freedoms, especially in terms of steering and the inevitable swing of its head module [24], which make it even more difficult to perform autonomous locomotion tasks [17]. The questions hereby are:

  How to agilely steer the locomotion of a snake-like robot?
  How to somehow effectively utilize the sensors mounted on the head module to acquire environmental information?

This thesis aims to give partial answers to these challenges and propose a biological-inspired hierarchical control architecture for performing autonomous and take a snake-like robot as the agent for demonstration. The detailed state-of-the-art related work is given in the next Chapter.

1.3 Thesis Outline and Contributions

Driven by the aforementioned motivations and challenges, a biological-inspired hierarchical control architecture is proposed in this thesis to mimic both functional and structural principles of the motor neural systems in living creatures, which involves an SNN-based sub-controller as the decision-

making layer, a CPG-based sub-controller as the locomotion generation layer. In order to apply the control architecture into practical implementations, snake-like robots and their locomotions are studied as well. The thesis is constructed with three parts after the introduction chapter: **Survey**, **Theory**, and **Applications** as depicted in Figure 1.3. The first part (Chapter 2) briefly gives us a survey on the robotics control based on learning-inspired spiking neural networks and a short background about the snake-like robots. We first highlight the primary impetuses of SNN-based robotics tasks in terms of speed, energy efficiency, and computation capabilities. We then classify those SNN-based robotic applications according to different learning rules and explicate those learning rules with their corresponding robotic applications. We also briefly present some existing platforms that offer an interaction between SNNs and robotics simulations for exploration and exploitation. Afterwards, the development of snake-like robots are introduced with several well-know examples, as well as their typical locomotion control methods. The details of the subsequent chapters are listed below and their contributions are summarized as well.

- **Chapter 3: Slithering gait.**

  In Chapter 3, we design a slithering gait for a planar snake-like robot to perform autonomous locomotion tasks. In order to make the robot agile, maneuverable, and be capable to efficiently acquire environmental information, we propose three methods aiming at improving its locomotion skills. Meanwhile, the superiorities of the proposed slithering gait is demonstrated by simulation. The detailed contributions are listed in the following:

  - Inspired by real snakes and on the basis of *serpentine curve* [25], the slithering gait is steered by shifting the serpentine curve center of each joint and shaped like a frustum. The relationship between the kinematic characteristics and the gait parameters are analyzed in detail, including the stability, forward velocity, and steering radius.
  - We propose an orientation compensation algorithm for decreasing the flapping of its head module when the robot slithers forward. The first joint is constantly used to adjust the head position to face the forward direction of the locomotion. To prove the effectiveness of the algorithm, a group of simulations are conducted by measuring the orientation angles of the head. As a result, the yaw angle is decreased by 90% compared with the unadjusted locomotion.

- **Chapter 4: Central pattern generator.**

  In this chapter, a lightweight CPG model with fast computing time is designed to control the locomotion of a snake-like robot. Its computing advantages are presented by comparing with

other state-of-the-art CPG models. Meanwhile, the function that smooth the gait transition process is investigated as well. Specifically, the detailed contributions are listed below:

  - Based on the gradient theory, we proposed a lightweight CPG model with fast computing time ability based on the gradient theory. Our CPG model is at least three times faster than the other widely adopted CPG models introduced in the literature, demonstrated by running different converge behaviors on MCU (Micro Control Unit).
  - Our CPG model is used to generate the joint commands for controlling the snake-like robot moving under different gait transition processes. Extensive simulations and prototype measurements are conducted on the robot trajectory and output torque for the change of both body shape and locomotion speed. The results prove that the CPG-based method can effectively avoid undesirable movement and abnormal torque.

- **Chapter 5: Spiking neural network.**
  In this chapter, a general paradigm for modeling spiking neural networks is introduced. It contains the neuron and synapse model, the R-STDP learning rule, the information encoding and decoding strategies, and its development environment. The contributions of this chapter are as follows:

  - We demonstrate the sensory information encoding strategies for three different sensors, i.e. the RGB vision sensor, the proximity sensor, and the dynamic vision sensor. All these sensors are used for implementing autonomous locomotion tasks.
  - With the existing motor decoding method [26], we propose a general SNN architecture with fully connected neurons using R-STDP synapses. By injecting different sensory information and giving proper rewards, the SNN is directly to learn the synaptic weights all by itself according to different tasks.

- **Chapter 6: Autonomous locomotion.**
  In this chapter, we first look back all the key components of the the biological-inspired hierarchal controller and then give the architecture by combing them together. Then, three different autonomous locomotion tasks are given, namely, target tracking, wall following, and obstacle avoidance. In autonomous locomotion applications, the snake-like robot must be adaptive to different situation. In target tracking as an example, the robot has to change its velocity and moving direction to follow the moving target correspondingly. According to the proposed paradigm, different sensors are used to acquire environmental information and drive

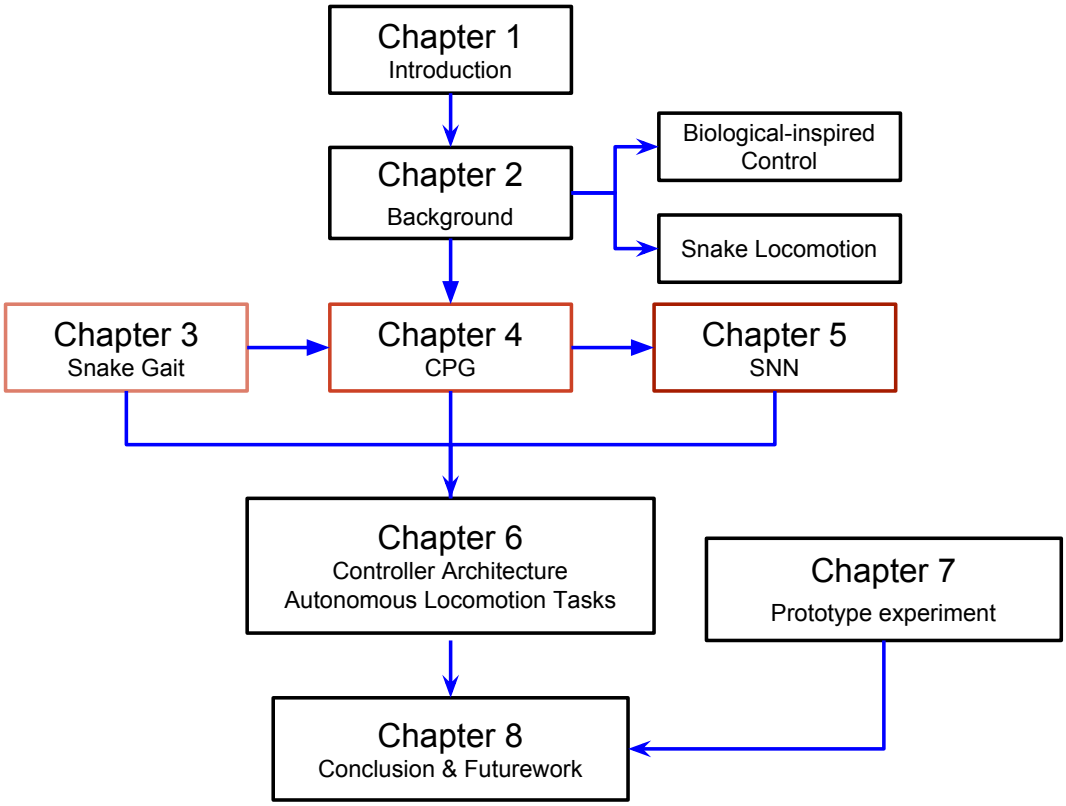


Figure 1.3: The outline of thesis.

the snake-like robot to perform each task. The training details and testing tasks are also presented with detailed analysis.

- **Chapter 7: Prototype experiment.**
  In this chapter, a snake-like robot for 3D locomotion is designed and used for prototype experiments. The total current is measured to represent the torque generated by its joints and analyzed to demonstrate the effectiveness of our proposed CPG controller.

- **Chapter 8: Conclusion and future work.** In this chapter, this thesis is concluded with its advantages compared with other learning-based control algorithms and existing possible future work.

Chapter 2

Background

In this chapter, we will mainly present a survey of the robotics control based on learning-inspired spiking neural networks and briefly introduce the state-of-the-art of snake-like robots, especially focusing on the locomotion control algorithms.

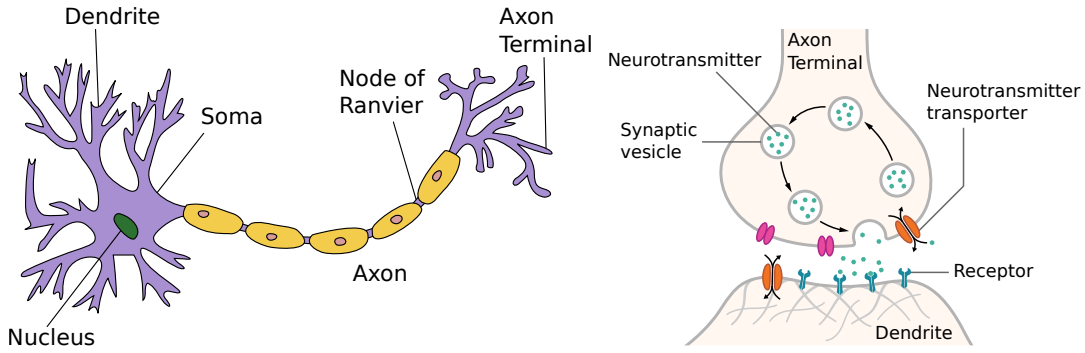
2.1 SNN-inspired Learning

2.1.1 Theoretical Background

Before studying in deep of robotics control based SNNs, it is worth briefly summarizing the biological mechanisms taking place in human nervous system. Therefore, this section serves as a short summary of the theoretical foundations as well as the vocabulary that is used in the following sections. An in-depth introduction of SNNs can be found in [27], [28], [29] and [30].

2.1.1.1 Biological Background

Even today's understanding of the human brain remains rather incomplete and challenging, some insights into our neural structure have been made over the past couple of decades. Since the initial discovery of neurons as the basic structure of the nervous system at the beginning of the twentieth century, a rough concept of how neurons might work has been developed. At the very basis, neurons can be understood as simple building blocks processing incoming information in the form of short pulses of electrical energy into output signals. By connecting neurons to huge networks, complex dynamics emerge that can process information and make sense of our world. This basic concept can be found all over nature, ranging from simpler organisms like jellyfish with a couple of thousand neurons to humans with an estimated number of 86 billion neurons on average in our nervous system [33, 34].



(a) Neuron [31] **(b)** Synapse [32]

Figure 2.1: Structure of the nervous system. **(a)** This figure is attributed to Quasar Jarosz at English Wikipedia, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=7616130`.. **(b)** The figure is attributed to Thomas Splettstoesser (`https://www.scistyle.com`) - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=41349545`

The structure of a typical neuron of the human brain embedded in a salty extra-cellular fluid is shown in Figure 2.1(a). Incoming signals from multiple dendrites alter the voltage of the neuronal membrane. When a threshold is reached, the cell body or soma sends out an action potential - also called spike or pulse - itself. This process of generating a short ($1ms$) and sudden increase in voltage is usually referred to as spiking or firing of a neuron. After firing, it is followed by a short inactive period called the refractory period, in which the neuron cannot send out other spikes regardless of any incoming signals.

Once the membrane potential threshold has been reached and the neuron fires, the generated output spike is transmitted via the axon of a neuron. These can grow quite long and branch out to a multitude of other nervous cells at the end.

At the end of an axon, the axon terminal, incoming signals are transmitted to other nervous cells, such as other neurons or muscular cells. There is now overwhelming evidence that synapses are in fact one of the most complicated part of a neuron. On top of transmitting information, they work as a signal pre-processor and play a crucial part in learning and adaption for many neuroscientific models. When a traversing spike reaches an axon terminal, it can cause a synaptic vesicle to migrate towards the presynaptic membrane, as shown in Figure 2.1(b). At the presynaptic membrane, the triggered vesicle will fuse with the membrane and release its stored neurotransmitters into the synaptic cleft filled with the extra-cellular fluid. After diffusing into this gap, neurotransmitter molecules have to reach a matching receptor at the postsynaptic side of the gap and bind with them. Directly or indirectly, this causes postsynaptic ion-channels to open or close. The resulting ion flux initiates a cascade that traverses the dendritic tree down to the trigger zone of the soma, chang-

ing the membrane potential of the postsynaptic cell. Therefore, different neurotransmitters can have opposing effects on the excitability of postsynaptic neurons, thus mediating the information transfer. These effects that make postsynaptic cells either more or less likely to fire action potentials are called excitatory postsynaptic potential or inhibitory postsynaptic potential, respectively. The dependence of postsynaptic potentials on different amounts and types of neurotransmitters released and the resulting number of ion-channels activated is often referred to as synaptic efficacy in short. After a while, neurotransmitter molecules are released again from their receptors into the synaptic cleft and either reabsorbed into the presynaptic axon terminal or decomposed by enzymes in the extra-cellular fluid.

The properties, characteristics, and capacities of synapses as signal pre-processors, e.g. chances of vesicle deployment or regeneration and amount of receptors, are not fixed, but always changing depending on the short and long-term history of its own and outside influences. Neuro-hormones in the extra-cellular fluid can influence both the pre and postsynaptic terminals temporarily, i.e. by enhancing vesicle regeneration or blocking neurotransmitters from activating ion-gate receptors. All these effects that change the influence of incoming spikes on the postsynaptic membrane potential are usually referred to as synaptic plasticity and form the basis of most models of learning in neuro and computer-sciences.

2.1.1.2 From McCulloch-Pitts to Backpropagation

In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a theoretical paper on how neurons might work describing a simple neural network model using electrical circuits [35]. Capable of performing mathematical operations with boolean outputs, these first generation neural networks fire binary signals once a threshold of summed incoming signals is reached in a neuron. They have been successfully applied in powerful artificial neural networks such as multilayer perceptrons and Hopfield nets [36].

With the advent of more powerful computing, this concept was later extended by introducing continuous activation functions, e.g. sigmoid [37] or hyperbolic tangent functions, to handle analog inputs and outputs as well. In fact, it has been proven that sufficiently large neural networks with continuous activation functions can approximate any analog function arbitrarily well by altering the network information flow through their synaptic weights [38]. The most commonly used and powerful supervised learning algorithm that takes advantage of continuous activation functions by using gradient-descent on an error function is called backpropagation [39].

However, second generation neurons do not model electrical pulses that have been described in

their biological counterparts, their analog information signals can actually be interpreted as an abstracted rate coding. Over a certain period of time, an averaging window mechanism can be used to code pulse frequencies into analog signals giving these models a more biologically plausible meaning.

2.1.1.3 Spiking Neural Networks

Following its biological counterpart, a third generation of neural networks [40, 41] has been introduced that directly communicates by individual sequences of spikes. Instead of using abstracted information signals, they use pulse-coding mechanisms that allow for the incorporation of spatial-temporal information that would otherwise be lost by averaging over pulse frequencies. It becomes clear that these neural network models, referred to as Spiking-Neural-Networks (SNNs), can be understood as an extension to second generation neural networks and can, in fact, be applied to all problems solvable by non-spiking neural networks [42]. In theory, it has been shown that these models are even more computationally powerful than perceptrons and sigmoidal gates [40].

Due to their functional similarity to biological neurons [43], SNNs have become a scientific tool for analyzing brain processes, e.g. helping to explain how the human brain can process visual information in an incredibly short amount of time [44]. Moreover, SNNs promise solutions for problems in applied engineering as well as power efficient, low-latency alternatives to second generation neural networks, e.g. for applications in robotics [45].

2.1.2 Primary Motivation and Framework

In this section, we will briefly establish the research and application impetuses of SNN-based robotics control from multiple aspects. Core points and a major framework for generally organizing an SNN for robotic implementation are introduced as well.

2.1.2.1 Primary Impetuses

As the third generation of the neural network model, SNNs have attracted more and more attention and gradually become an interdisciplinary research hotspot for neuroscience as well as robotics. For clarity and simplicity, the fascinating features of SNNs, which apply well to robotic controllers, can be summarized as follows.

Biological Plausibility From the perspective of neuroscience, SNNs once again raise the level of biological realism by directly using individual sequences of spikes in communication and computation, like real neurons do [46]. Experimental evidence accumulated during the last few years has

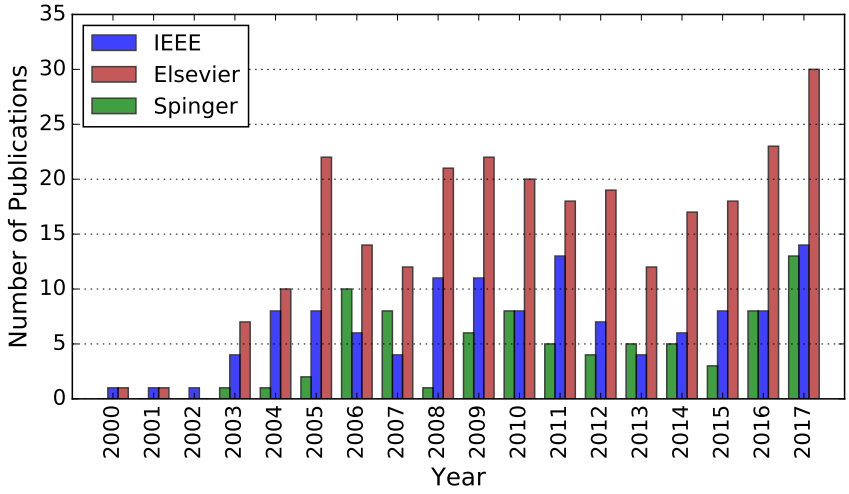


Figure 2.2: Number of publications whose abstract contains the terms "robot" and "spiking neural network" in the IEEE Explore and Elsevier Scopus database, respectively. Number of publications whose title contains the terms "robot" and its main text contains the term "spiking neural network" in the Springer database. All the data is from 2000 to 2016.

indicated that many biological neural systems use the timing of single-action potentials (or "spikes") to encode information [40], rather than the traditional rate-based models. show how the exact modeling of time in spiking neural networks serves as an important basis for In [47] It is explained that how the exact modeling of time in spiking neural networks serves as an important basis for powerful computation based on neurobiological principles.

Speed and Energy Efficiency Despite the hardware upgrades that make large neural networks applicable to real-world problems, it usually does not apply to robotics platforms with limited energy and computing resources. Since SNNs are able to transmit and receive large volumes of data encoded by the relative timing of only a few spikes, this leads to the possibility of very fast and efficient implementations. For example, experiments have demonstrated that visual pattern analysis and pattern classification can be carried out by humans in just 100 ms, in spite of the fact that it involves a minimum of 10 synaptic stages from the retina to the temporal lobe [48]. On the other hand, in terms of energy efficiency, maintaining sufficient working of the nervous system to perform various tasks requires a continuous energy supply [49]. Yet, the human brain only needs remarkably low power consumption, which is around 20 Watts of Power [7].

Computational Capabilities Recently, established experiments *in vivo* have indicated that SNNs are capable of processing the information sufficiently using a relatively small number of spikes to drive learning and behavior [50, 51, 52, 53]; meanwhile, they can also handle a large-scale network containing up to a trillion neurons like elephants [54]. Furthermore, SNNs are superior to non-spiking ones for utilizing the temporal information, referring to the precise timing of events to acquire the exact information with incredible precision and accuracy. For instance, the auditory system of the barn owl is able to locate sources of sound in the horizontal plane with a precision of 1 to 2 degrees, which equates to a temporal difference of only a few microseconds ($5\mu s$) between the arrival of sound waves at the left and right ears [55].

Information Processing Instead of using abstracted information signals, SNNs use pulse coding mechanisms that allow incorporating spatial-temporal information that would otherwise be lost by only averaging over pulse frequencies. This ability to learn and act in a dynamic environment, rich with temporal information, is a necessary quality for biological systems and for artificial systems that seek to perform similar tasks. Neurobiologists used weekly electric fish as a model to study the processing from stimulus encoding to feature extraction [56, 57]. They found that although pyramidal cells do not accurately convey detailed information about the time course of the stimulus, they reliably encode up- and down-strokes of random modulations by bursts of spikes. In addition, a problem referred to as "dynamic binding", has at best remained elusive to implement in neural networks, referring to different types of sensor information together in an assembly. SNNs are able to efficiently detect conjunctions of primitives (features) anywhere on a-large-input grid in an efficient, position-invariant manner. Examples such as data classification and image recognition tasks can be found in [58, 59] and [60, 61, 62]

In conclusion, these fascinating features make SNNs suitable for pursuing autonomy for robotics implementations. However, there is an implicit knowledge gap since SNNs are just investigated at the theoretical level, rather than widespreadly adopted to practical robotics applications. Even so, the growing knowledge of spiking neural networks and their increasingly popularity consistently draw more research attention and have led to more and more SNN-based implementations, as illustrated in Figure 2.2.

2.1.2.2 Research Orientations

The major research orientations of SNNs have focused on three aspects: SNN modeling, training, and implementations, which will be detailedly discussed or briefly introduced with other reference

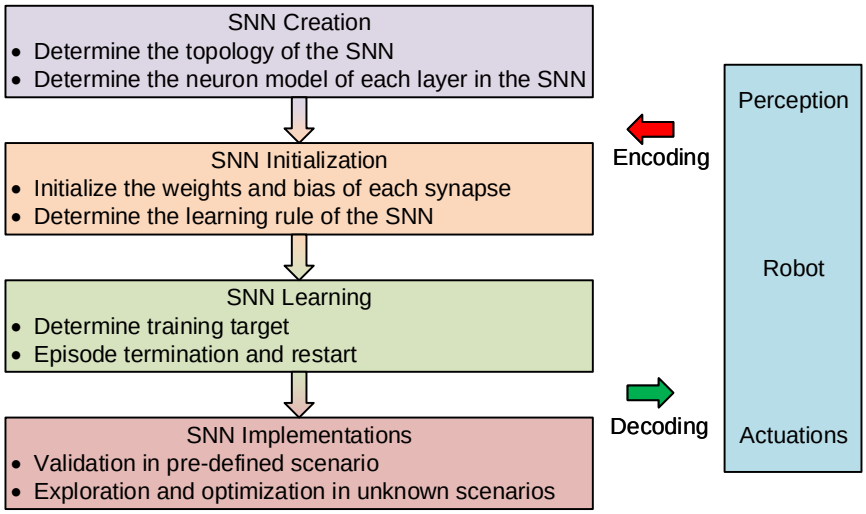


Figure 2.3: General design framework for SNN-based robot control.

readings in the following sections. A general design framework for SNN-based robot control is shown in Figure 2.3. Most robot control tasks chasing autonomy could be described as a cycle including three steps, namely, perception, decision, execution [63]. Robots usually use their sensors and actuators to sense and interact with the environment. However, the SNNs can be regarded as the brains to make decision, which build up a bridge between perception and execution, by taking encoded information from environment and outputting decoded motor commands for robots. To be specific, first, the architecture and mathematical model of an SNN should be determined including the neuron and synapse. Neurons are known to be a major signaling unit of the nervous system, and synapses can be seen as signal transmitters that communicate among neurons. For this reason, modeling of an SNN is of great importance to characterize its properties. Then, the SNN should be initialized and trained with specific parameters and learning rules, as conventional neural networks. Choosing an appropriate learning rule directly impact the performance of the networks. For an SNN, the most common learning rule is the Hebbian rule, which will be detailed explained in the following section. Finally, after training the SNN successfully, it should be validated in other scenarios and be optimized if necessary.

2.1.3 Modeling of Spiking Neural Networks

At the very beginning of the construction of an SNN for robot control, an appropriate SNN control model should be decided on. The basic task is to determine the general topological structure of the SNN, as well as the neuron models in each layer of the SNN.

2.1.3.1 Neuron Models

Generally, neuron models can be expressed in the form of ordinary differential equations. In the literature, many different mathematical descriptions of spiking neural models have been proposed, processing excitatory and inhibitory inputs using internal state variables. The most influential models used for SNNs are the Hodgkin-Huxley model [64] as well as the Integrate-and-Fire model and its variants [65]. To find an appropriate one among existing diverse neuron models, there is usually a trade-off to be balanced between the biological plausibility and complexity (Figure 2.4). A detailed comparison of the neuro-computational properties of spiking and bursting models can be found in [66].

Hodgkin-Huxley Model As the starting point for detailed neuron models which describe the dynamics of biological neurons, Hodgkin-Huxley is defined as:

One of the most widely used models is the so-called Leaky-Integrate-and-Fire (LIF) model [67] that can be easily explained by the principles of electronics. These models are based on the assumption that the timing of spikes, rather than the specific shape, carries neural information [68]. The sequences of firing times are called spike trains and can be described as

$$S(t) = \sum_f \delta(t - t^f), \tag{2.1}$$

where $f = 1, 2, \ldots$ is the label of a spike and $\delta(\cdot)$ is a Dirac function defined as

$$\delta(x) = \begin{cases} +\infty & \text{if } x = 0 \\ 0 & \text{if } x \neq 0 \end{cases} \tag{2.2}$$

$$\int_{-\infty}^{\infty} \delta(t)\, dt = 1 \tag{2.3}$$

. Passing a simplified synapse model, the incoming spike train will trigger a synaptic electric current into the postsynaptic neuron. This input signal $i(t)$ induced by a presynaptic spike train $S_j(t)$ can, in a simple form, be described by the exponential function [27]:

$$i(t) = \int_0^{\infty} S_j(s - t) \exp(-s/\tau_s)\, ds. \tag{2.4}$$

Here, τ_s denotes the synaptic time constant. This synaptic transmission can be modeled by low-pass filter dynamics.

The postsynaptic current then charges the LIF neuron model increasing the membrane potential u according to

$$\tau_m \frac{du}{dt}(t) = u_{rest} - u(t) + R\left(i_0(t) + \sum w_j i_j(t)\right). \tag{2.5}$$

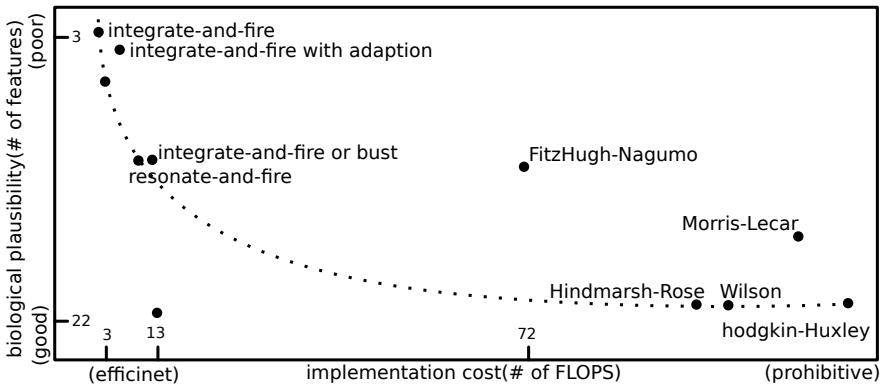


Figure 2.4: Ranks different neuron models by biological plausibility, which is defined by the number of different spiking behaviors or features a model can reproduce and by the computational complexity needed to simulate a model. Adapted from [66].

where $\tau_m = RC$ is the time constant of the neuron membrane, modeling the voltage leakage, depending on the resistance R. u_{rest} is the potential value after each reset. $i_0(t)$ denotes an external current driving the neural state, $i_j(t)$ is the input current from the j-th synaptic input and w_j represents the strength of the j-th synapse. Once the membrane potential u reaches a certain firing threshold ϑ, the neuron fires a single spike and its membrane potential is set back to u_{rest}. Usually, this spiking event is followed by a refractory period in which the neuron stays inactive and can't be charged again.

It is worth pointing out that biological studies highlight the presence of another operational unit *cell assemblies* [69] in the brain, which are defined as a group of neurons with strong mutual excitatory connections and tend to be activated as a whole. A deeper review of spiking neuron models can be found in [70].

2.1.4 Information Encoding and Decoding

The term neural encoding refers to representing information from the physical world (such as direction of a moving stimulus) in the activity of a neuron (such as its firing rate). On the other hand, information decoding is a reverse process to interpret from neuron activity to electrical signal for actuators (such as muscle or motor). How the brain encodes information is to think of two spaces: the physical space and neural space. The physical space can be the physical properties of objects, such as color, speed, and temperature. Neural space consists of properties of a neuron, such as firing rate in most cases.

A number of neural information encoding methods have been proposed, such as binary coding [71], population coding [72], temporal coding, and the most commonly used rate coding [73, 74].

For binary coding, neurons are only modeled to take two values *on*/*off*, but it ignores the timed nature and multiplicity of spikes altogether. Due to is simplicity, this coding mechanism was used in early-stage implementations. Besides, binary coding is also used to represent pixel value of an image. For rate coding, it is inspired by the observation that neurons tend to fire more often for stronger (sensory or artificial) stimulus. Scientists usually use a concept in probability theory known as the Poisson process to simulate spike trains that have characteristics close to real neurons. As the most intuitive and simple coding strategy, rate-coding has been adopted by most robotic implementations. For temporal coding, it is motivated by the evidence founded in neuroscience that spike-timing can be remarkably precise and reproducible [75]. With this encoding strategy, information is represented with the timing when the spike occurs. However, the underlying mechanism is still not so clear. The aforementioned coding solutions is usually for one single neuron. However, sometime a population of neurons is used as a whole to encode information. This is strongly supported by the brain of living creature, where functions are controlled by one area of neuron populations.

The goal of neural decoding is to characterize how the electrical activity of neurons elicit activity and responses in the brain. The most common used scheme for decoding is rate-based, where stronger neuron activity usually means higher motor speed or force. In [26], a steering wheel model based on an agonist-antagonist muscle system was proposed according to the spike numbers of output neuron.

2.1.4.1 Synaptic Plasticity Models

Once the neuron model is decided on, the synapse model should be carefully chosen to connect those neurons inside and among the layers of SNNs. By influencing the membrane potentials of each connected neuron, synaptic plasticity was first proposed as a mechanism for learning and memory on the basis of theoretical analysis (Hebb, 1949). Up to this day, the synaptic plasticity models used for practical implementations are typically very simple. Based on an input-output relationship between neuronal activity and synaptic plasticity, they are roughly classified into two types, which are rate-based and spike based, that differ in the type of their input variables.

Rate-based The first and most commonly used definition of a firing rate refers to a spike-count average over time [68]. The rate-based model is a popular approach for converting conventional ANNs into a spiking neural network that can still be trained by backpropagation. It has been successfully used in many aspects, especially in experiments on the sensory or motor system [76, 77, 78, 79].

Spike-based Spike-based learning rules were developed in [80], [81], [82], [83], and [84]. Experiments showed that the synaptic plasticity is influenced by the exact timing of individual spikes, in particular, by their order [85][86]. If a presynaptic spike preceded a postsynaptic spike, a potentiation of the synaptic strength could be observed, while the reversed order caused a depression. This phenomenon has been termed as Spike-Timing-Dependent-Plasticity (STDP) or anti-STDP for the exact opposite impact and explains the activity-dependent development of nervous systems. In other words, neural inputs that are likely to have contributed to the neurons' excitation are strengthened, while inputs that are less likely to have contributed are weakened. As for neuro-engineering, STDP has demonstrated to be successfully implemented as the underlying neural learning mechanism in robots and other autonomous systems in both simulated and real environments.

In the past, different mathematical models of STDP have been proposed, e.g. by [87]. For this work, the weight update rule under STDP as a function of the time difference between pre and postsynaptic spikes was defined as

$$\Delta t = t_{post} - t_{pre} \tag{2.6}$$

$$STDP(\Delta t) = \begin{cases} A_+ e^{-\Delta t/\tau_+}, & \text{if } \Delta t \geq 0 \\ -A_- e^{\Delta t/\tau_-}, & \text{if } \Delta t < 0 \end{cases}, \tag{2.7}$$

with A_+ and A_- representing positive constants scaling the strength of potentiation and depression, respectively. τ_+ and τ_- are positive time constants defining the width of the positive and negative learning window. For deeper insights into the influence of the STDP mechanism, readers could refer to [88, 89, 90].

A comparison of rate-based and spike-based spiking neural networks used for MNIST classification is shown in [91].

2.1.4.2 Network Models

The SNN network model resembles the synapse model in that it simulates synaptic interactions among neurons. Typical examples of neural networks consisting of neurons of these types are classified into two general categories:

Feed-forward Networks As the first and simplest type of network topology, information in feed-forward networks always travels from the input nodes, through hidden nodes (if any), to the output nodes and never goes backwards. In the biological nervous system, feed-forward networks are mainly found to acquire and transmit external information. Therefore, similarly, networks of this type are usually adopted for low-level sensory acquisition in robotic systems, such as vision [92],

tactile sensing [93], and olfaction [94].

Taking the work from [21] as an example, a two-layer feed-forward SNN was trained for a lane keeping vehicle. The control scheme is shown in Figure 2.5. In this work, the dynamic vision sensors (DVS) was used to detect the land markers by generating a sequence of events. The input layer consisted of 8×4 Poisson neurons and connected to the two LIF output motor neurons with R-STDP synapses in an “all to all" fashion. The learning phase was conducted by repeatedly training and switching the robot from the start positions in the inner and outer lanes. In comparison with other three learning methods, namely, the deep Q-learning (DQN), DQN-SNN, and Braitenberg Vehicle, the R-STDP SNN exhibited the best accuracy and adaptability in different lane scenarios.

Recurrent networks Different from the feed-forward networks, recurrent neural networks (RNNs) transmit their information with a directed cycle and exhibit dynamic temporal behaviors. It is worth pointing out that recurrent neural networks are recursive neural networks [95] with a certain structure such as a linear chain. Living organisms seem to use this mechanism to process arbitrary sequences of inputs with their internal memory stored inside RNNs. As for robotics implementations, RNNs are widely used for vision [96], planning [97, 98], and dynamic control [72].

In [97], a recurrent SNN is proposed for solving planning tasks, which consists of two populations of neurons, namely, the state neuron population and the content neuron population. (See Figure 2.6) The state neuron population consists of K state neurons, which control all the state of a freely moving target. In their finite horizon planning task, the agent spatial position is controlled by $nine$ state neurons. These state neurons are wired to each other and the content neuron populations by R-STDP synapse. The context neurons produce spatiotemporal spike patterns that represent high-level goals and context information. In this case, its average firing rate represents the target spatial position at different time step. A final reward is only received if the agent passes through two obstacles, one at time $T/2$ and one at time T. They show that the optimal planning policy can be learned using the reward modulated update rule in a network where the state neurons follow winner-take-all (WTA) dynamics. Due to the probability, in each time step exactly one state neuron is active and encodes the current position of the agent. Their results demonstrated a successful planner trajectory planning task using a recurrent SNN.

2.1.5 Learning and Robotics Applications

Changes in the strength of synaptic connections between neurons are thought to be the physiological basis of learning [99]. These changes can either be gated by neuromodulators that encode the

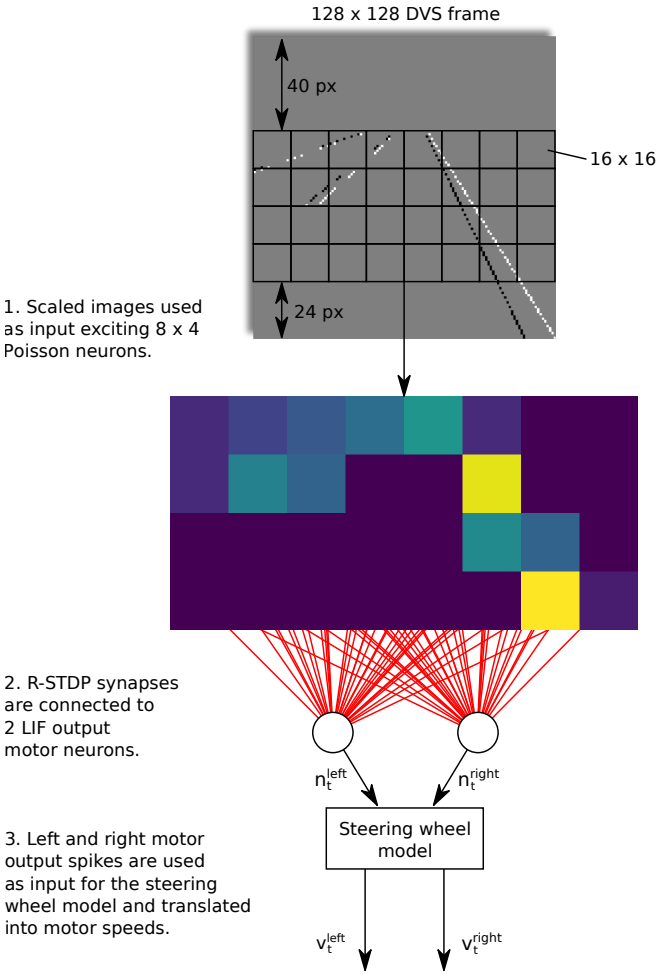


Figure 2.5: Control architecture of feed-forward SNN. A R-STDP SNN is used to achieve lane-keeping task. The sensor input is the event sequence from DVS and the two LIF output neurons are used to decode motor speed. All these neurons are connected with R-STDP synapse in an “all to all" fashion.

presence of reward or inner co-activation among neurons and synapses. In control tasks presented in this section, the network is supposed to learn a function that maps some state input to a control or action output. When successfully learned, the network is able to perform simple tasks such as wall following, obstacle avoidance, target reaching, lane following, taxi behavior or food foraging. In most cases, the network input directly comes from the robot's sensors, which range from binary sensors, e.g. olfactory, to multi-dimensional continuous sensors, such as cameras. In other cases, the input can be pre-processed data, e.g. coming from electroencephalography (EEG) data. Similarly, the output can range from one-dimensional, binary behavior control to multi-dimensional continuous output values, e.g. for motor control, as well.

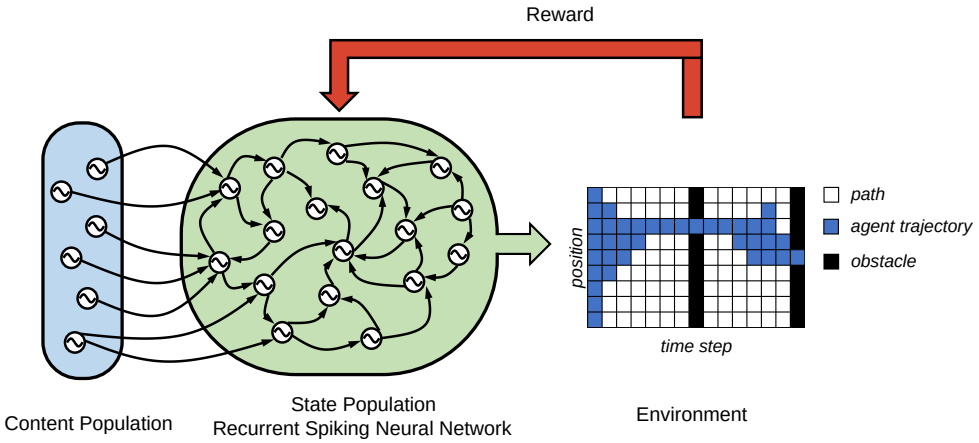


Figure 2.6: Control architecture of recurrent-SNN. A recurrent layer of state neurons is used to control the state of the agent and receives signals from the content population, which decides the target position according to different time step.

Initially, solving simulated control tasks was done by manually setting network weights, e.g. in [100] and [101]. However, this approach is limited to solving simple behavioral tasks such as wall following [102] or lane following [26], it is usually only feasible for very small network architectures with few weights.

Therefore, a variety of training methods for SNNs in control tasks has been researched and published. Instead of focusing on criteria such as field of research, biological plausibility or the specific task, this section is meant to serve as a classification of published algorithms into the basic underlying training mechanisms from a robotics and machine learning perspective. In the first part of this section, some implementations of SNN control are introduced that use some form of Hebbian-based learning. In the second part, publications are shown that try to bridge the gap between classical reinforcement learning and spiking neural networks. Finally, some alternative methods on how to train and implement spiking neural networks are discussed.

2.1.5.1 Hebbian-Based Learning

One of the earliest theories in neuroscience explaining the adaption of synaptic efficacies in the brain during the learning process was introduced by Donald Hebb in his 1949 book *The Organization of Behavior* [103]. Often summarized by the phrase *"Cells that fire together, wire together"*, his idea is usually expressed in mathematical terms as

$$\Delta w_{ij} \propto v_i v_j, \tag{2.8}$$

where w_{ij} refers to the change of synaptic weight between the presynaptic neuron i and the postsynaptic cell j; and v represents the activities of those neurons, respectively.

Hebbina-based learning rule that rely on the precise timing of pre and post-synaptic spikes play a crucial part in the emergence of highly non-linear functions in SNNs. Learning based on Hebbs rule has been successfully applied to problems such as input clustering, pattern recognition, source separation, dimensionality reduction, formation of associative memories, or formation of self-organizing maps [104]. Furthermore, different biologically plausible learning rules have been used for using Spiking Neural Networks in robot control tasks. However, as the basic underlying mechanism stays the same, training these networks can be achieved in different ways as follows (See Table 2.1). In the table, the two-wheel vehicle means a vehicle with two active wheels.

Unsupervised Learning According to STDP, if a presynaptic spike preceded a postsynaptic spike, a potentiation of the synaptic strength could be observed (Long Term Potentiation (LTP)), while the reversed order caused a depression (Long Term Depression (LTD)). Because of the absence of direct goals, correction functions or a knowledgeable supervisor, this kind of learning is usually categorized as unsupervised learning [104]. Learning based on STDP rule has been successfully applied to many problems such as input clustering, pattern recognition, and spatial navigation and mental exploration of the environment.

[105] used this approach to train a behavior controller based on SNN to achieve obstacle avoidance using ultrasonic sensory signals with a mobile robot. Compared with other classical NNs, they demonstrated that SNN needs fewer neurons and is relatively simple. Afterwards, they [106] extended their navigation controllers with with wall-following and goal-approaching abilities. In a similar research, [107] presented an SNN based on an unsupervised learning paradigm to allow the robot to autonomously learn how to navigate in an unknown environment. Their controller allowed the robot to learn high-level sensor features, based on a set of basic reflexes, depending on some low-level sensor inputs.

Supervised Learning In non-spiking neural networks, many successes in recent years can be summarized as finding ways to efficiently learn from labeled data. This type of learning, where a neural network mimics a known outcome from given data is called supervised learning. A variety of different neuroscientific studies has shown that this type of learning can also be found in the human brain [108], e.g. in motor control and motor learning [109], [110]. But despite the extensive exploration of these topics, the exact mechanisms of supervised learning in biological neurons remain

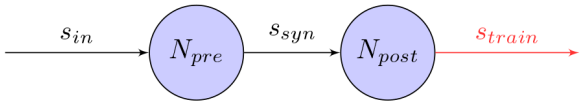


Figure 2.7: Supervised Hebbian training of a synapse: The weight of the synapse between pre and post-synaptic neurons, N_{pre} and N_{post}, is adjusted by the timing of the pre-synaptic spike-train s_{syn} and external post-synaptic training signal s_{train}.

unknown.

Accordingly, a simple way of training SNNs for robot control tasks is by providing an external training signal that adjusts the synapses in a supervised learning setting. As shown in Figure 2.7, when an external signal is induced into the network as a post-synaptic spike-train, the synapses can adjust their weights, for example, using learning rules such as STDP. After an initial training phase, this will cause the network to mimic the training signal with satisfactory precision. Even though this approach provides a simple, straight-forward way for training networks, it is dependent on an external controller. Especially for control tasks involving high-dimensional network inputs, this may not be feasible.

Several models have been proposed on how this might work, either by using activity templates to be reproduced [111] or error signals to be minimized [112], [110]. In the nervous system, these teaching signals might be provided by sensory feedback or other supervisory neural structures [113]. One of these models that is primarily suitable for single-layer networks is called supervised Hebbian learning (SHL). Based on the learing rule derived in 2.8, a teaching signal is used to train the postsynaptic neuron to fire at target times and to remain silent at other times. It can be expressed as

$$w_{ij}^{new} = w_{ij}^{old} + \alpha\, v_i\, t_j, \tag{2.9}$$

where w_{ij} again is the synaptic efficacy between a presynaptic neuron i and a postsynaptic neuron j, α is the learning rate, v_i is the presynaptic neurons activity and t_j represents the postsynaptic teaching signal.

[114] used this basic approach to train a spiking model of the cerebellum to control a robotic arm with 2 degrees of freedom in a target-reaching task taking joint angles and speeds, as well as target position as inputs. In contrast to other STDP learning rules, only long-term depression was externally induced by a training signal, which relied on the motor error, namely the difference between the desired and actual state. In a similar experiment, [115] trained a single-layer network to control a robotic arm with 4 degrees of freedom in 3D space. As inputs, joint angles and the spatial direction of the end-effector were used, while outputs consisted of four motor-command neurons. The training

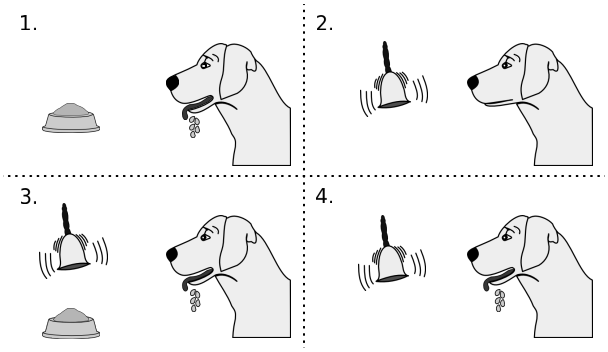


Figure 2.8: Classical Conditioning by Pavlov. (1) Before Conditioning, the dog starts to salivate when it sees food. (2) When a bell rings, the dog doesn't salivate. (3) During Conditioning the bell is rung shortly before the food is presented. (4) After conditioning, the dog will start salivating if the bell is rung. [116]

signal was computed using an inverse kinematics model of the arm, adjusting the synaptic weights with a symmetric STDP learning rule. More examples can be found in Table 2.1 with an order by descending year.

Classical Conditioning Classical conditioning [117] refers to a learning procedure in which a biologically potent stimulus (e.g. food) is paired with a previously neutral stimulus (e.g. a bell). It will result that the neutral stimulus comes to elicit a response (e.g. salivation), which is usually elicited by the potent stimulus. In the famous experiment on classical conditioning [118] (See Figure 2.8), Pavlov's dog learns to associate an unconditioned stimulus (US), in this case food, and a conditioned stimulus (CS), a bell, with each other. While, it is not clear how the high-level stimuli given in his experiment are processed within the brain, the same learning principle can be used for training on a neural level as well. Figure 2.9 shows how a synapse based on the STDP learning rule can associate US and CS provoking a response even in the absence of US.

Following this principle, bio-inspired robots can learn to associate a CS, e.g. sensory information, with a US that functions as an external reinforcer. That way, robots can learn to follow the desired behavior based on sensory inputs. [119], [107], and [120] showed how classical conditioning can be used in an obstacle avoidance and target reaching task. In an SNN with two output motor neurons, distance and vision sensors function as CS, while contact and target sensors work as US causing an unconditioned response. In a similar experiment, [121] carried out different classical conditioning tasks in a controlled virtual environment using infrared, ultrasound and visual neurons as CS and vibration neurons as US. [105][106] constructed a controller that stimulated two motor neurons as US. A single-layer SNN using proximity sensor data as CS input was then trained in tasks such as

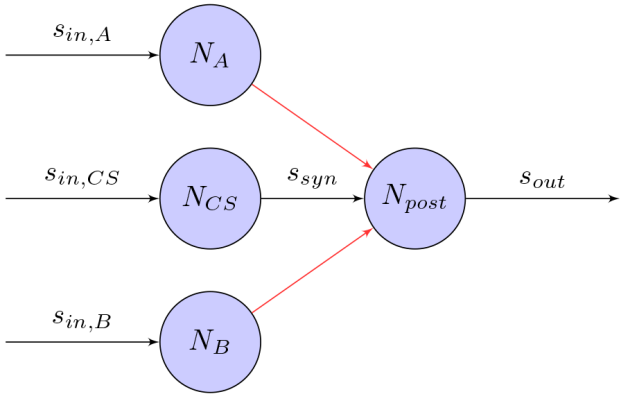


Figure 2.9: Classical Conditioning with STDP synapse between N_{pre} and N_{post}: An unconditioned stimulus (US) A or B causes the post-synaptic neuron N_{post} to fire. The conditioned stimulus (CS) firing shortly before its associated US will adjust its weights so that N_{post} will fire even in the absence of US. Due to the Hebbian learning rule, the synaptic weight is unchanged when the other, unrelated stimulus causes N_{post} to fire.

obstacle avoidance and target reaching. [122] used light sensors in a target-reaching task to punish wrongful behavior. [123], [124], and [125] implemented a virtual ant that learns to associate olfactory sensor input with different behaviors through a single-layer SNN. The robot was able to learn to recognize rewarding and harmful stimuli as well as simple navigation in a simulated environment.

In order to successfully learn such behavioral tasks, some unconditioned stimulus has to be given for every relevant conditioned stimulus that the robot should learn. This also means that the robot will not learn to associate stimuli that are delayed in time. Taken together, using classical conditioning for robot control basically means constructing an external controller that provides unconditioned stimuli for every relevant state input, which may not be feasible in many tasks.

Operant Conditioning While classical conditioning is concerned with passively associating conditioned and unconditioned stimuli with each other, operant conditioning consists of associating stimuli with responses and actively changing behaviors thereafter. Conceptually, operant conditioning is closely related to reinforcement learning and its agent-environment interaction cycle. Instead of developing a formal mathematical model, operant conditioning has been mainly researched in biological and psychological domains. Despite advances in the understanding of operant conditioning, it is still not clear how this type of learning is implemented on a neural level.

In this context, [126] and [127] developed a spiking OC model that consists of an input feeding cue neuron, an action neuron and a predictor neuron that receives rewards or punishments. With

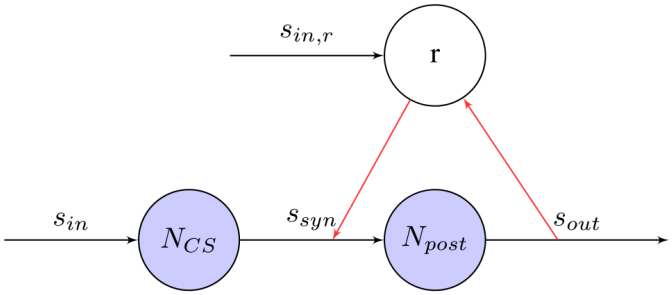


Figure 2.10: Reward-modulated STDP synapse between N_{pre} and N_{post}: Depending on the post-synaptic output spike-train, a reward r is defined that modulates the weight change of the synapse.

this simple basic architecture and learning rules such as habituation and STDP, they were able solve simple OC-related tasks in a simulated environment, such as pushing blocks. In another publication by [128] and [129] a reward-dependent STDP learning rule was implemented on a robot to allow for OC learning and demonstrated in a maze task. Interestingly, the developed learning rule closely resembles previously discussed modulated STDP learning rules, which will be discussed in the next section.

Reward-Modulated Training In Figure 2.10 the learning rule for reward-based training is shown. Using one or more chemicals emitted by a given neuron to regulate diverse populations of neurons is know as neuromodulation [130]. As one of the neuromodulators, dopamine neurons forming the midbrain dopaminergic cell groups are crucial for executive functions, motor control, motivation, reinforcement and rewards. Most types of neurological rewards increase the level of dopamine in the brain, thus stimulating the dopamine neurons [131]. Inspired by dopaminergic neurons in the brain, the effects of STDP events are collected in an eligibility trace and a global reward signal induces synaptic weight changes. In contrast to supervised training as discussed before, rewards can be attributed to stimuli, even if they are delayed in time. This can be a very useful property for robot control, because it might simplify the requirements of an external training signal leading to more complex tasks. A simple learning rule combing models of STDP and a global reward signal was proposed by [132] and [133]. In the R-STDP, the synaptic weight w changes with the reward signal R. The eligibility trace of a synapse can be defined as,

$$\dot{c}(t) = -\frac{c}{\tau_c} + \omega(\Delta t)\delta(t - s_{pre/post})C_1 \tag{2.10}$$

where c is an eligibility trace. $s_{pre/post}$ means the time of a pre- or post-synaptic spikes. C_1 is a constant coefficient. τ_c is a time constant of the eligibility trace. $\delta(\cdot)$ is the Dirac delta function.

$$\dot{w}(t) = R(t) \times c(t) \tag{2.11}$$

where $R(t)$ is the reward signal.

In the literature, a variety of algorithms has been published using this basic learning architecture for training. Even though they are all based on the same mechanism, the rewards can be constructed in different ways.

1. Rewarding Specific Events: The most straight-forward implementation of reward-based learning resembling classical reinforcement learning tasks uses rewards associated to specific events. [134] trained a simple, single-layer SNN in several food foraging tasks consisting of 4 input sensor neurons and 4 output motor neurons. In a separate network, other reward-related sensor neurons stimulated a dopaminergic neuron that in turn modulated the synaptic weight change. With this simulation setup, the robot was able to learn food-attraction behavior and subsequently unlearn this behavior when the environment changed. By shifting the dopamine response from the primary to a secondary stimulus, the robot was able to learn, even with large temporal distance, between correct behavior and reward. [135] showed an SNN model of a fruit fly that is able to execute both first and second order conditioning. In a simple task, the simulated fly learned to avoid getting close to an olfactory target emitting electric shocks. Furthermore, the same behavior can be transferred to a secondary stimulus that is associated to the primary stimulus without emitting electric shocks itself.

2. Control Error Minimization: As opposed to rewarding specific events, dopamine-modulated learning can also be used in an optimization task to minimize an objective function. This is usually achieved by strengthening or weakening the connections that lead to changes in the objective function based on their eligibility traces. [136] used this basic architecture to train an SNN to follow a trajectory. The network consisted of lateral state variables as inputs, a hidden layer and an output layer population decoding the lateral control output. Learning is achieved by minimizing the error between decoded actual and desired output, which is provided by an external linear controller.

3. Indirect Control Error Minimization: For some potential applications of SNNs, e.g. neuroprosthetic devices implanted in the brain, direct manipulation of synaptic weights might not be possible. Therefore, an indirect approach to training SNNs was shown by [137] that induces

changes in the synaptic efficacy through input spikes generated by a separate critic SNN. This external network was provided with control input as well as feedback signals and trained using a reward-based STDP learning rule. By minimizing the error between control output and optimal control law, it was able to learn adaptive control of an aircraft. Similar ideas were presented by [138], [139], [140], and [141] who trained a simple, virtual insect in a target reaching and obstacle avoidance task.

4. Metric Minimization: The same principle can also be applied to minimize a global metric that might be easier to construct and calculate than an external controller. [142] proposed a spiking-neuron model of the motor cortex which controlled a single-joint arm in a target-reaching task. The model consisted of 144 excitatory and 64 inhibitory neurons with proprioceptive inputs cells and output cells controlling the flexor and extensor muscles. A global reward or punishment signal was given depending on the change of hand-target distance. [143] and [144] extended this architecture later. Similarly, [145] used a biomimetic cortical spiking model composed of several hundred spiking model-neurons to control a two-joint arm. With proprioceptive sensory input (muscle lengths) and muscle excitation output, the network was trained by minimizing the hand-target distance. [146] extended the same basic architecture in order to develop a brain-machine interface. Extracellularly recorded motor cortical neurons provide the network inputs used for prosthetic control. By pressing a button, the user can reward desired movements and guide the prosthetic arm towards a target. Using a miniaturized microprocessor with resistive crossbar memories implemented on a two-wheeled differential drive robot, [147] and [148] showed how an STDP-based learning rule could lead to target approaching and obstacle avoidance behavior. Although, in this case, learning was implemented using if-then rules that relied on distance changes from target and obstacles, it is conceptually identical to reward-modulated learning. This can easily be seen by exchanging the if-rules with a reward of $+1$ or -1.

5. Reinforcing Associations: [149] introduced a tactile robot that uses a network architecture inspired by the insular cortex. As in classical conditioning, a dopamine-modulated synaptic plasticity rule was used to reinforce associations between conditioned and unconditioned stimuli.

2.1.5.2 Reinforcement Learning

In the previous subsection, a variety of approaches was presented for training SNNs based on Hebbian learning rules. This was done either by providing a supervised training signal through an external controller or by using a reward-based learning rule with different ways of constructing the reward. The latter type of learning, however, was shown to successfully train SNNs in simple tasks solely based on delayed rewards. In general, all of these approaches have been trained in tasks that don't require looking very far ahead, as reinforcement learning theories usually do.

In classical reinforcement learning theory, on the other hand, learning to look at multiple steps in advance in a Markov Decision Process (MDP) is one of the main concerns. Therefore, several algorithms have been published combining SNNs with classical reinforcement learning algorithms.

Temporal Difference The learning rule in which one looks at one or more steps forward in time was introduced as temporal difference learning. Hereby, [150] and [151] used place cells to represent the state space in an MDP and single-layer SNNs for state evaluations and policies. Both algorithms were able to learn to navigate in a simple grid-world after some training. With a similar approach, [152] presented a robot controller inspired by the control structures of biological systems. In a self-organizing, multi-layered network structure, sensory data coming from distance and orientation sensors was gradually fused into state neurons representing distinct combinations of sensory inputs. On top, each individual state neuron was connected to 3 output motor neurons. By fusing the sensory input into distinct state neurons and connecting them to action neurons, a simplified TD learning rule could be used to set each synaptic weight in the last layer individually. Performance of this controller was demonstrated in a wall-following task.

While these state representations work very well for relatively small state spaces, they are usually bound to fail for larger, high-dimensional state spaces. In fact, these approaches can conceptually be seen as an SNN implementation of table-based Q-learning.

Model-based Although for robot control tasks, such as those shown in this paper, model-free reinforcement learning methods seem favorable, two recent publications are at least worth mentioning that presented SNN implementations of model-based reinforcement learning algorithms. [97] presented a recurrent spiking neural network for planning tasks that was demonstrated on a real robot in an obstacle avoidance task. [153] implemented a biologically realistic network of spiking neurons for decision making. The network uses local plasticity rules to solve one-step as well as sequential decision making tasks, which mimics the neural responses recorded in frontal cortices during the

execution of such similar tasks. Their model reproduced behavioral and neuro-physiological data on tasks ranging from simple binary choice to multi-step sequential decision making. They took a two-step maze navigation task as an illustration. During each state, the rat was rewarded with different values according to its actions. The reward was modeled as an external stimuli. The SNN learned a stable policy within 10 ms.

Table 2.1: Learning Rules based on STDP/Hebbian Learning

Learning Rule	Robot	Sensor	Methodology	Reference
unsupervised	Two-wheel vehicle	5 Proximity Sensors	Implementing an SNN on a resistive memory device and apply it to navigation tasks	[148, 147]
	Mobile vehicle Casis-I	16 Ultrasonic Sensors	A behavior-based target-approaching navigation controller composed of three sub-controllers: the obstacle-avoidance, wall-following, and goal-approaching SNN controllers.	[154, 105]
	TriBot Robot	Distance Sensor, Contact Sensor,	Using SNN to make robot navigate in an unknown environment and avoid obstacles	[107]
Supervised	Two-wheel insect	4 Proximity Sensors	Implementing an indirect training SNN in digital CMOS to navigate with obstacles	[141, 140]
	Two-wheel insect	2 Terrain, 2 Target	Indirectly train an SNN by RBFs to determine precise spike timings and minimize a desired objective function	[139]
	Aircraft	IMU	Indirectly training an SNN to approximate an optimal flight controller	[137]
	4-DoF robotic arm	4 Joint Encoder, 3 Spatial direction of end-effector	Using supervised learning to train a single-layer network to control a robotic arm with 4 degrees of freedom in 3D space	[115]
	2-DoF robotic arm	Sensorimotor	Using supervised learning to train a spiking model of the cerebellum to control a robotic arm	[114]
Conditioning	Simulated fly	Olfactory Receptor	Implementing an SNN inspired by Drosophila olfactory system to simulate flight	[135]
	Lego EV3 robotic platform	Camera, Infrared sensor Colour/light sensor	Learning and unlearning autonomously locomotion based on visual-input with reinforced/aversive reflex-response	[124]
	Two-wheel robot	3 Proximity Sensors, 1 RGB Sensor	Using Reward-dependent STDP learning rule to allow OC and CC learning	[128, 129]
	Foraging Ants	Olfactory Sensors, Nociceptor	Learns to associate olfactory sensor input with different behaviors through a single-layer SNN	[123, 125]

	Lego NXT 2.0	Color sensor, Touch Sensor	Using SNN to sustain OC in multiple learning scenarios	[126, 127]
	Two-wheel Vehicle	Light Sensors	Using light sensors in a target-reaching task to punish wrongful behavior	[122]
	Two-wheel Vehicle	5 Proximity Sensors, 9 IR Sensors, Vibration Sensor	Using infrared, ultrasound and visual neurons as CS and vibration neurons as US	[121]
	Mobile vehicle Casis-I	16 Ultrasonic Sensors	A learning algorithm combining operant conditioning and a shunting neural dynamics model is applied to the path planning	[155]
	TriBot Robot	Distance Sensor, Camera, Contact Sensor,	Using target distance as CS, while contact sensors work as US causing an unconditioned response	[120, 119]
R-STDP	Flapping Insect	GPS and IMU	Indirectly training an SNN-based controller for adaptive flight control	[136]
	1-DoF robotics arm	5 Proximity Sensors	Using an SNN trained by a global reward and punishment signal to reach arbitrary targets	[144]
	Musculoskeletal arm, WAM robot	Encoders	Using a cortical spiking model composed of several hundred spiking model-neurons to control a two-joint arm	[145]
	CARL-SJR	Tactile Sensors	Using SNN to provide feedback to users by displaying bright colors on its surface.	[149]
	Two-wheel vehicle	2 Proximity Sensors	Implement a version of DA-modulated STDP on a food foraging task	[134]
	Foraging Simulator	Visual Sensors	Using reward-STDP based SNN to solve a grid-based foraging task	[156]
	DfRobotShop Rover	Camera,, Light Sensor	Using an SNN and external flash to reinforce the goal-directed and adaptive behaviors	[157]
	2-DoF robotics arm	Sensorimotor	Using an SNN based on R-STDP to control a two-joint virtual arm to reach to a fixed target	[143]
	1-DoF robotics arm	Encoder	Using an SNN to control a single-joint arm for target reaching	[142]

2.1.5.3 Others

Except for the two aforementioned major methods, there are also other training methods for SNNs in robot control tasks as follows (See Table 2.2).

Evolutionary Algorithms In nature, evolution has produced a multitude of organisms in all kinds of shapes with survival strategies optimally aligned to environmental conditions. Based on these ideas, a class of algorithms has been developed for finding problem solutions by mimicking elementary natural processes called evolutionary algorithms [158]. Generally, evolutionary processes can be understood as some form of gradient-descent optimization. Therefore, a typical problem using these algorithms is getting stuck in local minima. In applications in robot control, evolving SNNs have been shown to work well in mostly static environments. Due to the training principle of trial and error, there are usually difficulties in dynamically changing environments.

[159] showed a vision-based controller in an irregularly textured environment that navigated without hitting obstacles. The predefined SNN consisted of 18 sensory-input receptors connected to 10 fully-connected hidden neurons and 2 motor-output neurons. Using static synaptic weight values, the algorithm was used to search the space of connectivity by genetically evolving only signs of weights. With a population of 60 individuals, fitness was evaluated by summing up over motor speeds at every time step, and new generations were created using one-point crossover, bit mutation and elitism. [160] later extended this approach to evolving SNN weights as well using adaptive crossover and mutation probabilities. They were able to evolve good SNN controllers in a small number of generations in a wall-following scenario. [161] presented a quadrotor neurocontroller that performed a hovering task in challenging wind conditions. With a feed-forward network taking the differences between current position and target position as input and pitch, roll and thrust as output, weights and topology were evolved to minimize the spatial error. In a target-reaching and obstacle-avoidance task using binocular light sensors and proximity sensors, [162] evolved an SNN by minimizing the control error in order to mimic an external controller signal. [163] used a feed-forward network architecture of predefined size to control a toy car. Based on speed, localization and road boarder input signals, the network controlled speed regulation and turn direction, and evolved its weights using a genetic algorithm.

Self-Organizing Algorithms [164] formulated a synaptic learning rule that enforced connections between neurons depending on their activities. With this self-organization algorithm that resembles other Hebbian-based learning methods, they were able to learn obstacle avoidance and simple

navigation behavior.

Liquid State Machine As a particular kind of SNN, an LSM usually consists of a large assemblage of neurons that receives time-varying input from external sources as well as from other neural units [165]. All mixed and disorderly neuron units are randomly generated and then arranged under the activations of recurrent spatio-temporal patterns of the connections obtained from the time-varying input. Hence, the LSM is regarded as a large variety of nonlinear functions which is able to compute the output as linear combinations of the input. LSMs seem to be a potential and promising theory to explain brain operation mainly because neuron activities are not hard coded and limited for specific tasks. [166], [72] and [167] showed how liquid state machines can be trained for robot control tasks.

Table 2.2: Other learning rules.

Learning Rule	Robot	Sensor	Methodology	Reference
Evolutionary Algorithms	*Neural Racing* game	Speedometer, Proximity Sensors	Using evolutionary algorithm to train SNN and compare results with multi-layer perceptron	[163]
	Quadrotor	GPS	Using evolutionary algorithm to generate high utility topology/weight combinations in the SNN	[161]
	Two-wheel Vehicle	5 IR Sensors	Using SNN to mimic the behaviors captured under control of a heuristic rule program	[162]
	Khepera Robot (Two-wheel Vehicle)	Linear Camera	Using evolution to rapidly generate SNN capable of navigating in a textual environment	[168, 159]
	Two-wheel Vehicle	4 IR Sensors	A use-dependent synaptic modification algorithm of SNN for obstacle-avoidance vehicle behavior	[164]
	Two-wheel Vehicle	9 Ultrasonic Sensors, 4 Bump Sensors	Using an adaptive GA to evolve the SNN online through interaction with the real environment	[160]
Fuzzy Logical	Two-wheel Vehicle	7 Ultrasonic Sensors (5 in front, 2 at back)	Using SNN to mimic the knowledge of a fuzzy controller	[169]
Liquid State Machine	Hexapod Robot	Visual Sensor (Distance, Height)	Mushroom bodies in drosophila are modeled as a recurrent SNN under LSM paradigm	[167]
	2-Dof Ball Balance Platform	Position and Velocity	Using a cortical network (LSM) to learn under a supervised learning rule for position control	[72]
	Khepera Robot	8 IR Sensors	Using Randomly generated recurrent SNN to operate real-time obstacle avoidance	[166]

2.1.6 Simulators and Platforms

With the fast development of neurology and chip industry, large-scale neural simulators using spiking neural networks have been studied to achieve the same capabilities as animal brains in terms of speed, efficiency, and mechanism. For examples, SpiNNaker [170] is a million-core system for modeling large-scale SNNs in real time. TruthNorth [171] contains 1 million programmable spiking neurons and only consumes less than one hundred milliwatts. Other neuromorphic computing platforms such as Neural Grid [172], NeuroFlow [173] can be found and introduced in [174]. Meanwhile, a growing number of dynamic simulators has been developed to assist robotic research [175], such as Gazebo [176], ODE [177], and V-Rep [178]. Those simulators greatly facilitate the research process that involving mechanical design, virtual sensors simulation, and control architecture.

Although adequate tools exist to simulate either spiking neural networks [179, 180], or robots and their environments [181, 182], tools that offer researchers joint interaction, including a realistic brain model, robot, and sensory-rich environment, are in need. Some existing platforms are listed in Table 2.3.

iSpike [183], as the first attempt to combine spiking neural networks and robots, is a C++ library that provides an interface between SNN simulators and the iCub humanoid robot. It uses a biologically inspired approach to convert the robot's sensory information into spikes that are passed to the neural network simulator, and it decodes output spikes from the network into motor signals that are sent to control the robot. CLONES [184] communicates between the Brian neural simulator [185] and SOFA [186] and is also an interface used for shared memory and semaphores. A more generic system which permits dealing with simulated robotic platforms is AnimatLab [187], which provides functionalities such as robot modeling, two neural models, and plugins for importing other models.

Recently, the first release of the HBP Neurorobotics Platform (NRP) [188, 189] was presented, which was developed within the EU Flagship Human Brain Project. For the first time, it provides scientists with an integrated toolchain to connect pre-defined and customized brain models to detailed simulations of robot bodies and environments in in-silico experiments. In particular, NRP consists of six key components, which are essential to construct neurorobotics experiments from scratch. It can be seen that the NRP provides a complete framework for the coupled simulation of robots and brain models. The Brain Simulator simulates the brain by bio-inspired learning algorithms such as a spiking neural network to control the robot in a silico neurorobotics experiment. The World Simulator simulates the robots and their interacting environment. The Brain Interface and Body Integrator (BIBI) builds a communication channel between brain models and robot models. The Closed Loop Engine (CLE) is responsible for the control logic of experiments as well as for the data commu-

nication between different components. The *Backend* receives requests from the frontend for the neurorobotics experiment and distributes them to the corresponding component, mainly via ROS. The *Frontend* is a web-based user interface for neurorobotics experiments. Users are able to design a new experiment or edit existing template experiments.

Table 2.3: Taxonomy of Platforms for Robotics Control based on SNNs

Platform	Name	Methodology	Reference
Platform	Neurorobotics Platform	Design, import, and simulate different robot bodies and diverse brain models in rich environments	[188]
	Musculoskeletal Robots	Combining Myorobotics with SpiNNaker the proof of principle of a system that can scale to dozens of neurally controlled, physically compliant joints.	[190]
	Retina simulation	The retina simulation platform is integrated in the NRP.	[101]
	Neural self-driving vehicle simulation framework	A visual encoder from camera images to spikes inspired by the silicon retina, and a steering-wheel decoder based on an agonist antagonist muscle model.	[26]
	iSpike	Interface between SNN simulators and the iCub humanoid robot	[183]
	AnimatLab	Provide functions, such as robot modeling, two neural models, and plugins for importing other models.	[187]

2.2 Snake-like Robots

Snake robots, as a class of hyper-redundant mechanisms, can use their many degrees of freedom to achieve a variety of locomotion capabilities [191]. There are several review articles expounding the mechanism, modeling, and locomotion control research results. A general overview can be found in [17, 192, 193]. Perception-driven obstacle-aided locomotion of snake-like robot is presented in [194]. A preliminary review on metrics for snake-like robots locomotion can be found in [195].

2.2.1 Mechanism of Snake-like Robots

Based on their mechanical structures, snake-like robots can be roughly divided into two types: planner snake-like robot with passive wheels [196, 197] and 3D snake-like robots with orthogonal modular joints [198, 199, 22, 200]. This type of configuration is also extended into the fish-like robots [201, 202] and reptile-like robots [203, 204]. In this thesis, we name the first kind as the wheeled snake-like robot and the second kind as the modular snake-like robot.

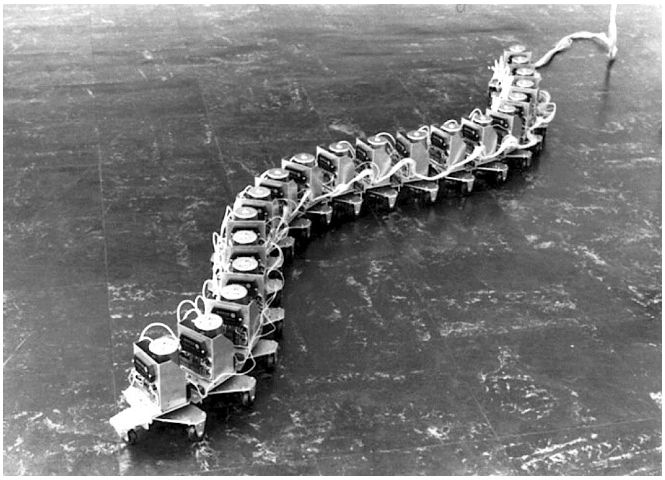


Figure 2.11: The first snake-like robot, *Active Cord Mechanisms* known as *ACM III*, was developed by Hirose in 1972 [205].

2.2.1.1 Wheeled Snake-like Robots

The first snake-like robot, *ACM III* (See Figure 2.11), was developed by Hirose in 1972 [205]. This robot was a successful prototype to demonstrate the *serpenoid curve*, which was inspired by real snakes and used to model their later undulations. The robot was equipped with passive wheels (two for each body module) to simulate the frictional anisotropy of snake skin [206], which was known as

Figure 2.12: Salamander robot developed by Ijspeert [204].

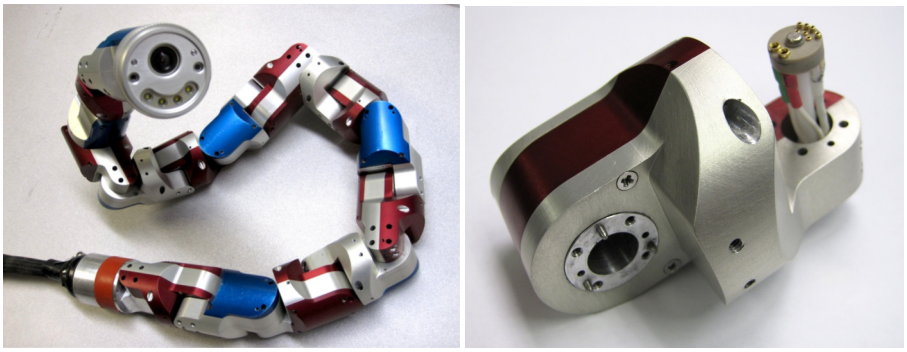


Figure 2.13: The *Unified Snake* robot (left) consists of 14 body modules, one head module, and one tail model. One of its identical body modules (right), developed by Choset [210].

side-slip constraints at that time. Hereafter, Hirose developed a series of *ACM* snake-like robots with similar mechanical configuration, such as *ACM-R3* [207], *ACM-R5* [208], and *ACM-R7* [209]. Generally speaking, *ACM III* defines the wheeled snake-like robots, where a string of body modules are connected by parallel joints and equipped with passive wheels. Many other snake-like robots with this type of configuration were designed afterwards and a good review of this can be found in [17]. This kind of snake-like robots mainly move on flat surfaces in 2D space, therefore the traversability in challenging terrain is great limited.

It is worth mentioning that this type of mechanical design has also been widely used for fish-like robot or reptile-like robots, for example, the salamander robot developed by Ijspeert [204] (See Figure 2.12).

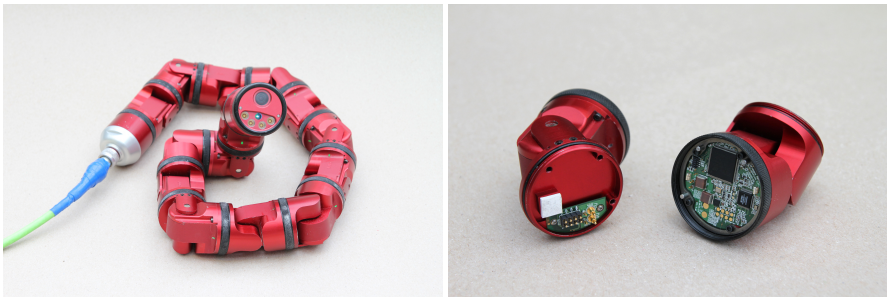


Figure 2.14: The SEA snake-like robot (left) consists of 13 body modules, one head module, and one tail model. One of its identical body modules (right), developed by Choset [211].

2.2.1.2 Modular Snake-like Robots

To increase their 3D locomotion abilities, another kind of snake-like robots without passive wheels was investigated. Inspired by [212, 213], Howie Choset designed one kind of snake-like robot that rely solely on their internal shape changes to locomote [214, 210]. As we can see in Figure 2.13, this kind of snake-like robot consists of identical modules, except for the head and tail module. All the modules are orthogonally connected to the adjacent modules and allows a full 180° rotation.

Afterwards, Choset redesigned their body module by changing its driving mechanism [211]. An series elastic actuator (SEA) was added to the output shaft of each module, where the motor position is transferred into torque (See Figure 2.14). As the snake-like robots are required to travel through narrow environment or with obstacles, the SEA unit is used to change the body stiffness under torque control. However, this kind of modular snake-like robots sacrifice their speed and stability to some degree due to their inherent locomotion skills. Especially under the slithering gait, their head module usually wiggle from side to side and hence the sensors mounted in the head module can not acquire stable information to aid autonomous locomotion.

2.2.2 Locomotion Control of Snake-like Robots

On the basis of the two kinds of snake-like robots, the locomotion control methods differ as well. For the wheeled snake-like robots, the control methods generally aim at generating the serpentine curve by calculating each joints angle. And the locomotion control methods hereby can be classified into three types: sinusoid-based, CPG-based, and model-based. The sinusoid-based method [196] is achieved to control the snake-like robot, by observing the morphology of real snakes, imitating the serpenoid curves with simple sine-like signals. It was firstly presented by Hirose [25] as serpenoid curve, which is a mathematical description of lateral undulation. Ma proposed another model *serpentine curve* to describe the snakes locomotion by modeling their muscle characteristics

and proved a higher efficiency than the *serpenoid curve* by running simulations [196].

The CPG-based method is similar to sinusoid-based method, where the target wave is generated by neural modulated method instead of defining a sinusoid wave directly. When a snake-like robot has to change its motion state, such as speed, body shape, and moving direction, there will be a sudden change in the joint positions that either potentially damaging the robot or affecting the locomotion by mistake [22]. In the effect of CPG, this kind of sudden change can be smoothly transferred than the sinusoid-based method. And then a smooth gait transition can be further achieved and implemented in autonomous tasks [23].

Different from the first two kind control methods, the model-based method is to analyze the locomotion using its kinematics and dynamics. Since the interaction between the ground and the modular snake-like robot is too complicated, the model-based method is mainly used for wheeled snake-like robots. Prautsch and Mita proposed a dynamic model of a wheeled snake-like robot based on Lagrange's equations of motion [215]. Afterwards, similar methods are also used to different locomotion, such as lifting the head module in the air [216] and travel through ascending and descending Steps [217]. While, several crucial bottlenecks of this method limit its use, where the redundant degrees of freedom of a snake-like robot really complicate its dynamics and the interaction against the uneven ground is almost unpredictable for calculation.

For the modular snake-like robot, the locomotion is also driven by sinusoid signals, but propagate along the joints in the lateral and dorsal respectively. The *gait equation* [199] is an abstract expression of gaits of a snake-like robot by describing joint angles as parameterized sinusoidal functions. It allows for the emergence of complex behaviors from low-dimensional representations with only few key parameters, greatly expanding their maneuverability and simplifying user control [218]. Inspired by this, researchers developed several biological gaits for snake-like robots to move inside and outside door environment [200].

2.2.3 Gaits of Snake-like Robots

At first glance snakes look rather sluggishly in the locomotion, since they degenerate limbs and lack other appendages as well, with which to propel their cylindrical bodies. In fact, many snakes are fast movers, efficient climbers, excellent swimmers, adept burrowers, and some can even fly. Normally, there are four typical and common kinds of gaits, namely, lateral undulation, concertina locomotion, rectilinear crawling, and sidewinding.

As the most widely adopted gait, slithering, also called serpentine crawling or lateral undulation, is shown in Figure 2.15. With this gait, the snakes constantly contract their bodies from side to side

Figure 2.15: Typical serpentine locomotion is invariably exhibited by *Tropidonotus* when the animal is moving through grass or over a substratum of sufficiently irregular surface [219].

to create a *S*-shape body wave propagating from head to tail. By pushing against the ground where the bodies bend, the snakes propel themselves forward Serpentine movement works efficiently on rough land or in water, but fails on slippery surfaces. One important fact is that, even the bodies undulate heavily, their head can remain steady and focus on the moving direction stably.

Like their biological counterparts, snake-like robots locomote using cyclic motions to generate propulsion. For wheeled snake-like robot, the most common gait is slithering, which is inspired by serpentine locomotion. Since one important factor for locomotion of real snakes is the anisotropic friction property of their skin, the passive wheels are used to imitate this property by offer very low friction in direction of rotation and high friction in lateral direction. On the basis of slithering gait, Ma [220] presented a locomotion control based on CPG of a wheeled snake-like robot. By smoothly changing the body shape, the robot could adapt to different space widths and also exhibit obstacle avoiding behaviors. Pfotzer *et al.* [216] proposed a navigation approach for reconfigurable snake-like robots to autonomously overcome unknown and challenging obstacles like stairs or large steps.

For modular snake-like robots that rely solely on their internal body shape change, the most common gaits are rolling, sidewinding, slithering, and helix locomotion [199]. Since missing those passive wheels, the slithering gait is fundamentally different. There are two wave propagating along the lateral and dorsal plane of the snake body. The dorsal wave propels the robot forward like a worm and the lateral wave forms the body into an *S* shape to supply balancing pivots. After giving solutions to decrease the shaking of the snake robot, Bing first presented the model of slithering gait and achieved a target tracking task [24]. Helix is a gait crawl inside of pipes and climb outside

of poles. This gait is mainly used for pipeline inspection and surveillance purposes [221]. Modular snake-like robots can move to the sideways quickly under the rolling and sidewinding gait, since they remove the passive wheels as the constraint. By using the *gait equation*, most complex modes of snake locomotion can be described by the superposition of waves: horizontal and vertical body waves propagating with different phase difference. Zhen *et al.* modeled the rolling gait by changing its rolling shape to cross obstacles [222]. Sidewinding is another efficient gait for locomotion over at ground. Hatton *et al.* identified the stability of sidewinding and its conditions for sidewinding on the slope [223]. Furthermore, Gong *et al.* modified the gait for steering purpose by transforming the amplitude of the gait equation into a function depending on the element index. Although these 3D gaits enhanced the locomotion ability of snake-like robots, it is still far from autonomous locomotion [224, 225, 226].

Chapter 3

Locomotion Foundation: Slithering Gait Design

In this work, a snake-like robot is taken as the agent to perform a series of autonomous locomotion tasks. In order to dynamically cope with different task situations, it has to be capable of offering great maneuverability such as accelerating or decelerating, steering, and stability. In this chapter, the slithering will be further modeled on the basis of the *serpentine curve* [196] and analyzed in terms of its steering, body shape regulation, and head position control.

In the 1940s, Gary first described and analyzed the four main types of locomotion observed in snakes by using sinusoid curves. Among them, the slithering gait has been observed in nearly all kinds of snakes. Under slithering gait, every part of the snake's body and tail glides forward simultaneously and faithfully follows the path taken by the head. Gary also took a three-segment snake as an model to visualize the fundamental relationship between the sinusoidal form of the body and the ability of the axial muscles to propel the animal in serpentine movement [227]. Inspired by the locomotion of biological snake, Hirose proposed the *serpenoid curve* that greatly simplified the slithering control by neglecting the dynamic interaction within the body and environment [228]. In 1999, Ma [196] employed the muscle characteristics of snakes to model their lateral undulation, named the *serpentine curve*. Ma showed that snake locomotion according to the serpentine curve has a higher locomotive efficiency than locomotion according to the serpenoid curve. We will further investigate the modeling of slithering gait on the basis of the *serpentine curve*.

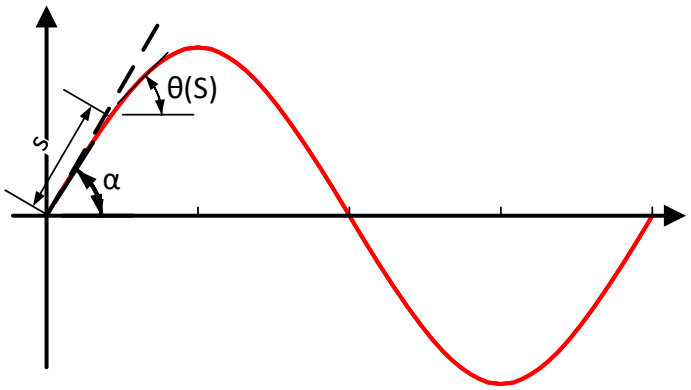


Figure 3.1: Illustration of the serpentine curve in one period.

3.1 Modeling of Slithering Gait

The curvature of the *serpentine curve* ρ is shown in Figure 3.1 and described as

$$\rho = -\alpha b \sin(bs) \tag{3.1}$$

, where α is initial bending angle of the curve, $b = \frac{2\pi}{L}$ is a proportional constant with L as the wave-length of the curve in one cycle, and s is the length of the serpentine curve. After integrating the s in (3.1), we can obtain the angle θ along the serpentine curve as,

$$\theta(s) = \alpha \cos(bs) \tag{3.2}$$

If we discretize the serpentine curve into a series of linkage and joint to represent the body of a snake-like robot, the joint angle ϕ can be calculated as

$$\phi = \theta(s+l) - \theta(s-l) = -2\alpha \sin(bl) \sin(bs) \tag{3.3}$$

, where $2l$ is the length of each module of a snake-like robot. For the i^{th} joint, we can obtain its joint angle ϕ_i as

$$\phi_i = -2\alpha \sin(bl) \sin(b \times 2l(i-1)) \tag{3.4}$$

$$\phi_i = -2\alpha \sin(bl) \sin((i-1)\beta) \tag{3.5}$$

, where $s = 2l(i-1)$ represents the curve length of the i^{th} module. $\beta = 2bl$ is a constant parameter. Then, in order to make each module go through all the serpentine curve, we integrate the time t into (3.3) to propagate the wave. Therefore, we can get the joint angle of each module as a function of time as

$$\phi_i = A \sin(\omega t + (i-1)\beta) \tag{3.6}$$

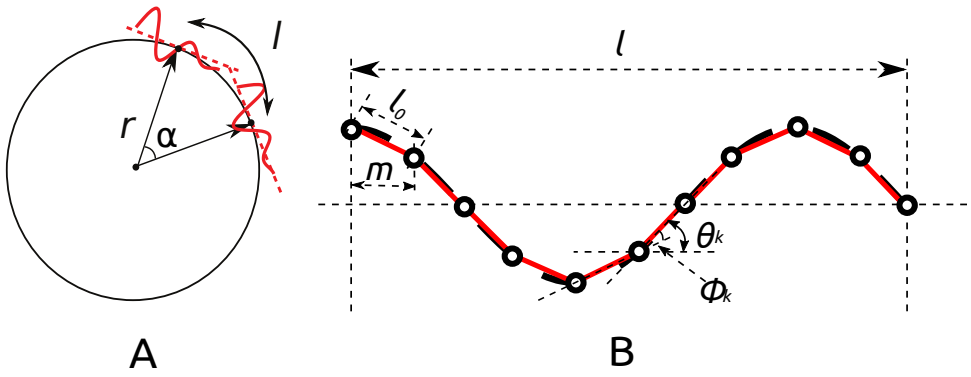


Figure 3.2: (A) The snake robot (red line) with effective traveling length l_e moving along a circle (black line) with radius r. (B) Our snake robot with module length l_0. Illustration of joint angle ϕ and global module angle θ.

, where ω and $A = -2\alpha\sin(bl)$ represent the frequency and the amplitude of the slithering curve, respectively.

The amplitude of the wave domains the body shape when the robot is slithering forward. On the other hand, the frequency determines the speed of the robot. Therefore, with this standard *serpentine curve*, we can control the motion state of the snake-like robot.

For most situations of diverse tasks, a snake-like robot will start to move from an initial position that all body modules are usually aligned as a straight line. Therefore, a damping function $(1-e^{-\lambda t})$, is used to smooth the start-up process, where λ is used to adjust the convergence rate. Then the serpentine wave equations is shown as

$$\phi_i = A(1-e^{-\lambda t})\sin(\omega t + (i-1)\beta). \tag{3.7}$$

For now, this *serpentine curve* is still not sufficient for performing practical tasks, since it only changes the robot's speed or drives it forward. In the following sections, we will further investigate the slithering gait to make it more suitable for autonomous locomotion tasks.

3.2 Radius Control

Steering direction agilely during locomotion is one of the most critical skills for many animals to survive in the wild. Based on the observations during different turning locomotions, snakes usually impose an amplitude modulation to steer their bodies for high maneuverability [218]. Inspired by this, our snake-like robot also steers its moving direction by adding a bias to the amplitude A in (3.7) as

$$\phi_i = A(1-e^{-\lambda t})\sin(\omega t + (i-1)\beta) + C \tag{3.8}$$

, where C is a bias value. This bias slithering gait is further used for investigating the steering control.

A snake-like robot slithers forward based on the assumption that there is no slide in the lateral plane between the body and the ground [227]. Given this assumption, the rear modules of the robot will follow the trajectory of their previous module, which is consistent with the locomotion of real snakes. Figure 3.2A illustrates the relationship between a biased slithering snake robot and a turning radius. r_d is the turning radius and α is the steering angle after traveling one effective body length l_e. Thus, the turning radius can be calculated as,

$$r_d = \frac{l_e}{\alpha} \tag{3.9}$$

Since the rear modules of the snake are following their previous module, the change of the heading direction can be determined by adding up the single module angles with respect to a global frame. Thus, the steering angle α can be computed by summing over the joint angle θ in (3.8) as

$$\alpha = \sum_{1}^{N} \phi = \sum_{1}^{N} [C + A \times \sin(\omega t + (i-1)\beta)] \tag{3.10}$$

After averaging over a period of time, the sum of the sine function part approximates zero. Thus, it leads to a simple relation between the bias C and the heading angle α as

$$\alpha = N \times C \tag{3.11}$$

, where N is the number of joints.

On the other hand, the effective body length l_e in one period should be further calculated. Since the bias C only shifts the body curve perpendicularly along the body length direction, it will be sufficient to analyze the average body length l_e without regarding the bias C and the steering angle α. Since the time t only propagates the wave along the body and does not affect the body length l_e, we manually set $t = 0$ for reasons of simplicity. As shown in Figure 3.2B, each module contributes an effective length that is parallel to the forward direction. In this case, the effective boy length l_e is the sum of the effective module length l_k of all modules as:

$$l_e = \sum_{k=1}^{N} l_k \tag{3.12}$$

$$l_k = l_0 \times \cos(\theta_k - \theta_{snake}) \tag{3.13}$$

, where k is the module index and θ_{snake} is the global angle of the snake robot, which is manually set as 0. l_0 is the length of each module.

According to our assumption, the rear modules of the robot will follow the trajectory of their previous module. Then, the joint angle ϕ_k is the result of concatenated local joint angles, which is

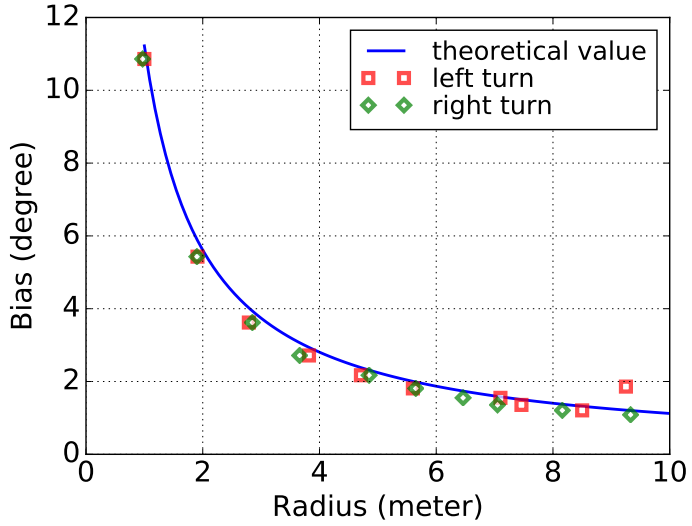


Figure 3.3: The relation between the amplitude bias and the turning radius is depicted. The blue solid line is the theoretical value according to (3.16). The red square and the blue diamond marks are the measured radius when the snake turns left and right, respectively. Meanwhile, other parameters are set as $A = 20°$, $\beta = \frac{2}{9}\pi$, and $\omega = 1.5\pi$.

calculated as:

$$\phi_k = \sum_{p=1}^{k} \phi_p = A \times \sum_{p=1}^{k} \sin(\beta \times p) \tag{3.14}$$

. Now with all quantities being computed, a look back on the introducing (3.9) and (3.11) leads to:

$$C = \frac{l_e}{N r_d} \tag{3.15}$$

With above equations, the turning radius can be calculated as:

$$r_d = \frac{l_0 \times \sum_{k=1}^{N} \cos(A \times \sum_{p=1}^{k} \sin(\beta \times p))}{C} \tag{3.16}$$

, under the condition that the slithering parameters in (3.8) are previously known.

To verify our proposed turning method, we run a series of simulation experiments to compare the ground truth steering radius and the theoretical radius calculated by equations. We adopt an 8-joint planer snake-like robot to run two groups of simulation experiments, in which the robot is required to turn left or right with a range of expected turning radius. Meanwhile, the measured turning radius is extracted by plotting out the trajectory of the snake-like robot over a period of time.

The results are illustrated in Figure 3.3. First, with the amplitude setting as $A = 20°$, the theoretical relationship between the turning radius r and the bias C is depicted with a blue line. Then, by varying the turning radius from $1m$ to $10m$, the snake-like robot turns left with a decreasing bias.

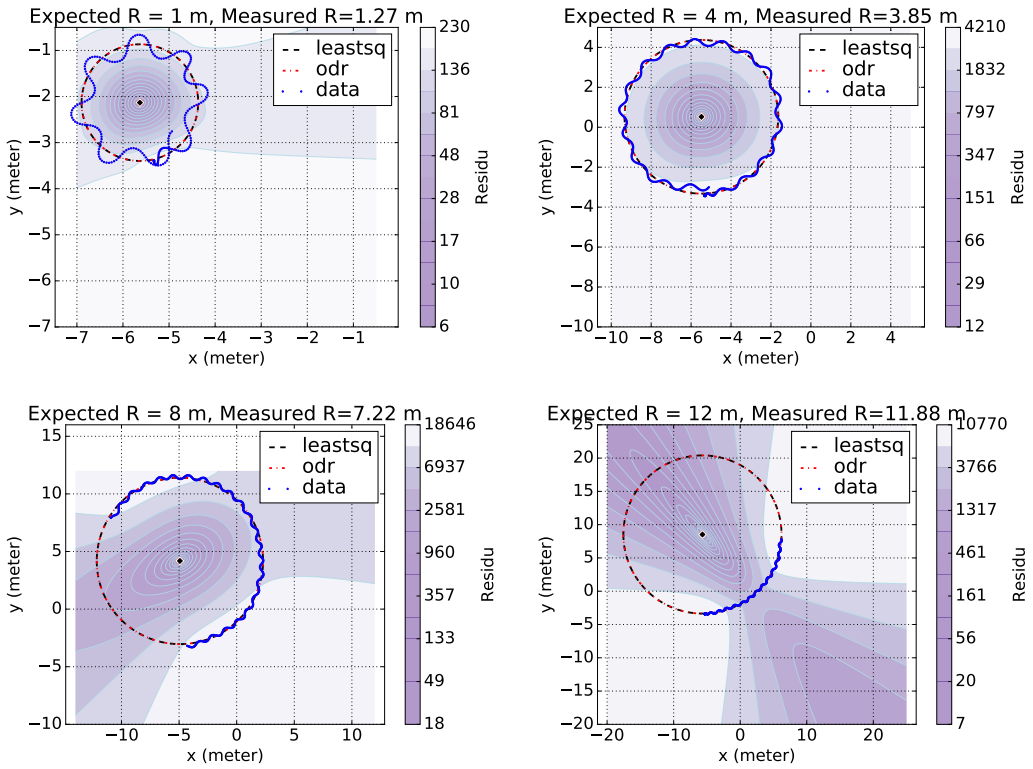


Figure 3.4: The measured turning radius comparison with setting amplitude as 30°. From left to right, the expected radii are 1 m, 4 m, 8 m and 12 m. The measured radius is calculated by two least-squares fitting of circle. The trajectory of the head module of the snake robot is represented by blue solid points. The calculated circle is represented by red dash line. Moreover, the fitting residue is depicted with a color bar, since only part of the trajectory are sampled.

The measured turning radius is marked with red square. By varying the amplitude bias from $-10m$ to $-1m$, the robot turns right with a decreasing bias and marked with green diamond The results clearly exhibit that our method can calculate the inverse relationship between the turning radius and the amplitude bias correctly, no matter turning left or right.

An trajectory example is shown in Figure 3.4 as a case study. We manually set up the amplitude as $A = 30°$, since the robot mostly travel around with this body shape value. Four different turning radii are 1 m, 4 m, 8 m and 12 m. The trajectory of the head module of the snake robot is represented by blue solid points. The calculated circle is represented by red dash line. Moreover, the fitting residue is depicted with a color bar, since only part of the trajectory are sampled. From these results, we can conclude that our proposed method can achieve steer the robot around precisely, no matter with a large or a small turning radius. It is worth mentioning that the robot exhibits larger turning radius error, when we increase the amplitude over 60°. This is because the robot slides much more, due to

the strong body undulation and thus the aforementioned assumption is not valid anymore.

3.3 Body Regulation

With the radius control method, we can navigate the snake-like robot to the direction as we expect. However, the robot has to sense the environment during movement in order to perform autonomous locomotion tasks. Under the slithering gait, the robot shapes its body as the *serpentine curve*, in which the head module is wavering with the rest of the body in the meantime. This side to side swing makes its head very difficult to concentrate on its target while slithering forward. To tackle this problem, we first introduce the body regulation method to decrease the head swing. This method is also inspired by real snakes that they usually shape themselves into a triangle-like shape where the motion range is decreased from the head to the tail (See Figure 2.15).

First, we adopt the linear reduction P to linearize the body curve of the snake-like robot from head to tail, defined as

$$P = (\frac{k}{N} \cdot y + z) \in [0,1], \forall k \in [0,N] \tag{3.17}$$

, where z and y are the linear reduction parameters. Then, the *gait equation* is extend as,

$$\phi_i = P \cdot A(1 - e^{-\lambda t}) \cdot \sin(\omega t + (i-1)\beta) \tag{3.18}$$

. Under the effect of the body regulation, the amplitude for each joint is defined as a linear function dependent on the module number k.

To further validate the effectiveness of our proposed method, we also conduct a series of simulation experiments and the results are shown in Figure 3.5. First, the snake-like robot is controlled to slither forward and the orientation data of the head module is measured in the meantime. As shown in Figure 3.5 *A*, the orientation of the head module with time is presented in the world coordinate, in which the z direction is upward perpendicular to the ground and the x direction is align with the forward direction. Therefore, the alpha angle along the z direction shows a sine-wave curve, while the beta and gamma are zero.

As shown in Figure 3.5 *B*, three groups of linear reduction parameters are used and the alpha angles are plotted, respectively. For $y = 0$ and $z = 1$, the effect of the linear reduction is prohibited as a reference (blue line). For $y = 0.5$ and $z = 0.5$, the alpha orientation is significantly decreased from 60° to 25° (green line). When we further strengthen the linear reduction to $y = 0.8$ and $z = 0.2$, the alpha angle is reduced to 10° (red line).

If the linear reduction goes to $y = 1$ and $z = 0$, the snake-like robot will move like a fish that solely pushes the body with those rear parts. However, there will bring another problem along with

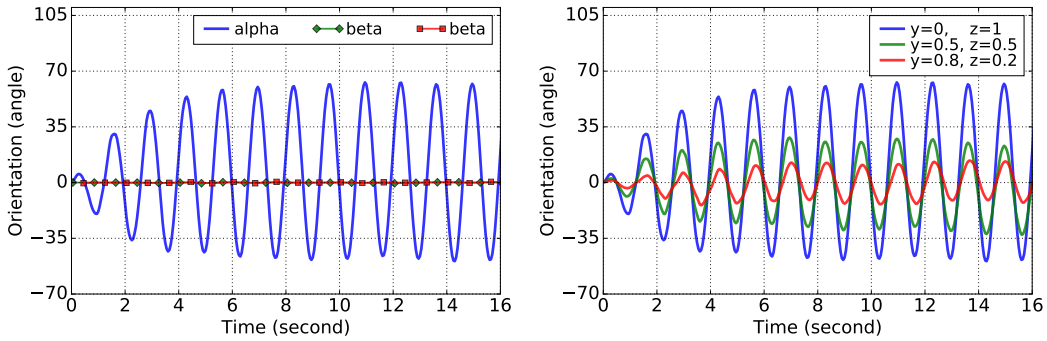


Figure 3.5: Left: the orientation angle of the head module in the world coordinate during slithering locomotion, in which the z direction is upward perpendicular to the ground and the x direction is align with the forward direction. Right: alpha orientation angle performance under three different groups of linear reduction parameters.

the linear reduction method that it will greatly exacerbate the torque output of those rear modules and incur potential damage on those motors. Therefore, we take the linear coefficient starts at $y = 0.3$ for the head module and ends at $z = 0.7$ for the tail module in this thesis.

3.4 Head Orientation Control

In previous sections, we have modeled the slithering gait to drive the locomotion, aiming at the stability, body shape, and steering ability. In order to perceive the environment properly and effectively, we use the body regulation to reduce the head swing movement. However, it is still not sufficient to be adopted for autonomous locomotion tasks, since this undulatory motion inevitably causes the head module swinging from side to side, even it has been limited to a range of $\pm 10^{\circ}$. The head swing will bring negative influences for further advanced locomotion control based on those sensors located at the head module, especially the vision sensor inside. The aforementioned linear reduction method can limit the head swing in some way, but meanwhile the head module will seriously drag down the locomotion speed and aggravate the torque burden of the rear modules. In this section, we will seek solutions inspired by real snakes to eliminate this harmful swing or orientation by using locomotion compensation.

When we observe the real snakes, we can always find out that the snake will lift its head or preaxial body up and aim at the front direction during the locomotion or attack. Even for sidewinders like a rattlesnake, it can adjust its head posture to look at the direction of locomotion under the sidewinding gait that push the body to its sideway [1]. This is important for snakes since they can observe the

[1] `https://youtu.be/B3NbPUTD5qA`

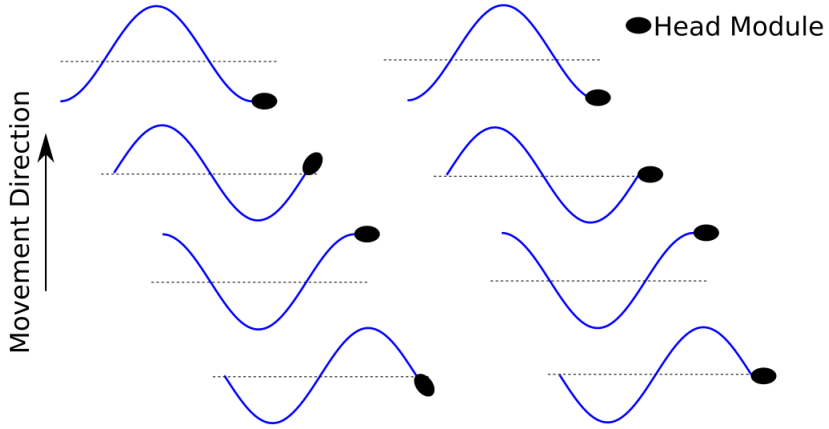


Figure 3.6: Top view of the backbone curve of the snake robot with and without head compensation (left and right).

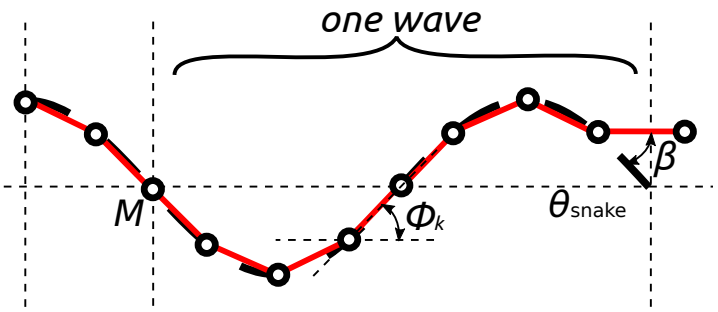


Figure 3.7: This diagram shows how to use the first joint to compensate the head orientation in order to make the robot aim at the front direction all the time during the locomotion.

environment effectively during the locomotion, which is helpful to avoid natural enemies or find the quarry.

As shown in the left of Figure 3.6, the head will swing along with the body periodically under an unadjusted undulation motion. If the body is a continuous sinusoid-like curve, the direction of the head will be align with the tangential direction of the body. However, in the right figure, the head is aiming at the forward direction constantly by compensating the angle at the head joint all the time. For real snakes, they can travel with the rear part of the body and attack with the front part of the body. This agile motion is achieved by using their flexible muscles and ball socket skeletons. For our snake-like robot, we will use the first joints to compensate the yaw orientation angle during the slithering locomotion.

As depicted in Figure 3.7, the compensation angle for the first joint is γ, the joint angle for the

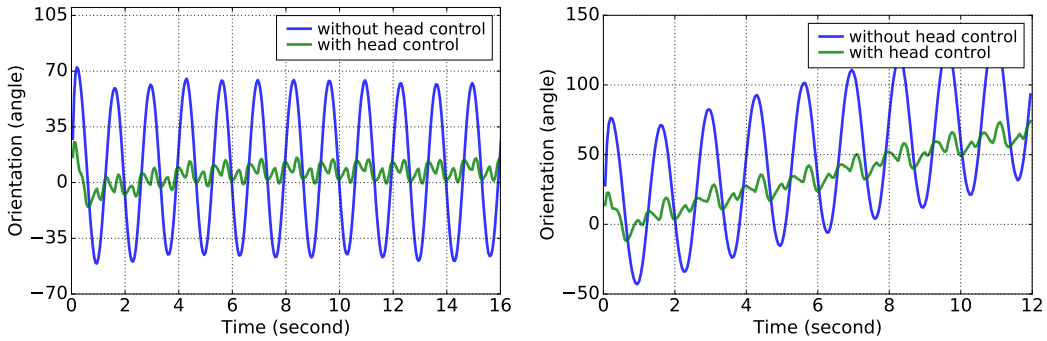


Figure 3.8: The head control results compared with the one without head control. The figure in the left is the straight slithering and the right one shows the results from a left turning with $r = 5\ m$.

k^{th} module is ϕ_k. We still assume that the rear modules of the robot will follow the trajectory of their previous module in an ideal situation that the snake will not slide sideways. Therefore, in the global coordinates, the i^{th} module angle can be calculated as,

$$\phi_i = \gamma + \sum_{j=2}^{i} \phi_j \tag{3.19}$$

Besides, for one whole sinusoid wave, there are M joints. The summation of those evenly distributed module angle should be zero in the coordinate of the snake, which can be described as,

$$\sum_{1}^{M} \phi_i = 0 \tag{3.20}$$

The phase difference summation from the first module to the M_{th} module should be 2π. Therefore, in the global frame, we can calculate the heading direction of the snake θ_{snake} as,

$$\sum_{1}^{M} \phi_i = (M+1) \times \theta_{snake} \tag{3.21}$$

$$\theta_{snake} = \frac{1}{M+1} \times \sum_{1}^{M} \phi_i \tag{3.22}$$

The phase difference summation from the first module to the M_{th} module should be 2π. Therefore, after integrating (3.19) into (3.20), we can get

$$\sum_{i=1}^{M} (\gamma + \sum_{j=2}^{i} \phi_j) = (M+1) \times \theta_{snake} \tag{3.23}$$

. Then we can derive the compensation angle γ as

$$\gamma = \theta_{snake} - \phi_1 \tag{3.24}$$

To further examine our proposed method, we run a serious of simulation experiments. By running simulations, we can obtain the orientation angles of the head module during slithering and the results are shown in Figure 3.8. Compared to Figure 3.5, the alpha (yaw) angle is greatly decreased at 90% from 50° to 5° roughly, with our head compensation method. Besides, we also investigate the practicability and effectiveness of our method when the robot is turning itself. As shown in the right of Figure 3.8, the robot is performing a steering action with $r = 5m$. The orientation angle is shifting around in the world coordinate but still maintains a stable fluctuation. The result with head control (green curve) exhibits clearly small range of orientation compare to the one without head control (blue curve).

3.5 Discussion

In this chapter, we present the model of our slithering gait, which will be adopted to perform autonomous locomotion tasks. First, the slithering gait is modeled based on the *serpentine curve* and formalized with simple interface to control the motion state, such as speed and body shape. Second, we investigate the radius control to steer the snake-like robot accurately. Then, in order to strengthen the sensing ability, we reshape the body curve of the snake-like robot with our linear reduction method. Finally, to further limit the head orientation to an acceptable range on the basis of the body regulation, we use the first joint to compensate the head movement and make it aim at the forward direction roughly.

Chapter 4

Low-Level Control: Central Pattern Generator

As the key component of motor system in living creatures, central pattern generators (CPGs) have many vital features including producing the rhythmic motor pattern, taking action commands from high-level controller or sensory feedback, and smoothing locomotion processes [23]. In this chapter, a low-level central pattern generator (CPG) controller will be modeled and implemented to generate the rhythmic signal for driving the slithering gait. Furthermore, the CPG controller is used for smoothing the gait transition process. Under its effect, the joint torque is greatly decreased and the locomotion is smoothed during gait transition process.

4.1 Mathematical model of CPG network

Likewise the long chain-type body of a snake, our CPG controller also takes the chain-type network structures which is formed by a chain of oscillators coupling the neighbor oscillators. Inspired by Kopell [229], we design an oscillator model for the chain-coupled CPG network based the convergence behavior of the gradient system, which can adjust the output signal's frequency, phase difference, and amplitude as required.

The concept of *Gradient System* comes from the gradient field [230]. For any point M in space region G, there is a certain scalar function $V(M)$ corresponding to point M. Then, $V(M)$ is one certain scalar field in space region G. If there is a gradient function grad$V(M)$ corresponding to the point M, then a gradient field can be determined. The gradient field is generated by the scalar field $V(M)$, which is called the potential of the gradient field. In space region G, the potential of any point decreases against the gradient direction , as shown in (4.1).

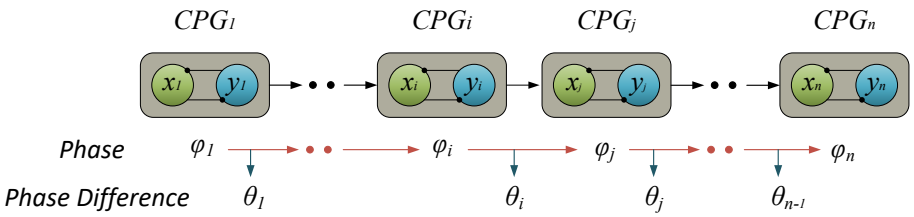


Figure 4.1: Parameters setting of CPG net with chain-type.

$$\frac{dx}{dt} = -\frac{\partial V}{\partial x}, \quad \forall x \in G \tag{4.1}$$

If the gradient system has a minimal value x^* in G, then all the vector x will converge to x^*, which satisfies:

$$\left.\frac{\partial V}{\partial x}\right|_{x=x^*} = 0, \quad \left.\frac{\partial^2 V}{\partial x^2}\right|_{x=x^*} > 0 \tag{4.2}$$

The gradient system has an important property; any initial state x_0, will converge to the minimal value in region G during finite time and remain stable since then. According to (4.2), we can design a chain-type CPG network to realize a fixed phase difference among all the CPG models. The phase differences among all CPGs can be seen as a group of state vectors. A global convergence gradient system is assumed in G, which is composed of those state vectors. The vectors in G will converge to the extreme point of the gradient system in finite time. If the extreme point of the gradient system is different from the target phase vector we set, the CPG network will converge to the desired phase difference from any position in finite time. As shown in Figure 4.1, the chain-type CPG network is composed of n neurons with the same parameters. Suppose that the phase of the i^{th} CPG is $\varphi_i(t)$, and the phase difference between two neighbouring CPGs i^{th}, j^{th} is $\theta_i(t)$, then the desired phase difference decided by the gait generator is $\tilde{\theta}_i$.

Consider the chain-type CPG network as a directed graph. The topological structure of the chain-type CPG network can be described by the incidence matrix. Set the incidence matrix $T = \{a_{i,j}\}_{(n-1)\times n}$ and satisfies:

$$\{a_{i,j}\} = \begin{cases} -1, & \text{from } j \text{ to } i \\ 1, & \text{from } i \text{ to } j \\ 0, & i \text{ and } j \text{ are not adjacent} \end{cases} \tag{4.3}$$

Therefore, the incidence matrix T for chain-type CPG network is

$$T = \begin{bmatrix} 1 & -1 & & & 0 \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ 0 & & & 1 & -1 \end{bmatrix}_{(n-1)\times n} . \tag{4.4}$$

Thus, the relationship between the phase difference and the phase vector is as follows:

$$\Theta = T\Phi \tag{4.5}$$

where phase difference vector is $\Theta = [\theta_1, \theta_2, ..., \theta_{n-1}]^T$, the phase vector is $\Phi = [\varphi_1, \varphi_2, ..., \varphi_{n-1}, \varphi_n]^T$.

In order to design the potential function, Ψ_i is used as the generalized coordinates of the gradient system, which satisfies:

$$\Psi_i = \begin{cases} \varphi_1 - \varphi_2 = \theta_1, & i\text{=1} \\ \varphi_{n-1} - \varphi_n = \theta_n, & i\text{=n-1} \\ \varphi_{i+1} + \varphi_{i-1} - 2\varphi_i = \theta_{i-1} - \theta_i, & \text{otherwise} \end{cases} \tag{4.6}$$

Then the potential function of a parabolic system is as follows:

$$V(\Psi) = \sum_{i=1}^{n} \mu_i (\psi_i - \tilde{\psi}_i)^2 \tag{4.7}$$

where μ_i is the coefficient of the convergence velocity and $\tilde{\psi}_i$ is the generalized coordinates of the desired phase differences. According to (4.1), $V(\Psi)$ can be seen as the gradient system. $\tilde{\psi}_i$ is the desired phase difference, represented in the new coordinate constructed by vector Ψ. Thus, the gradient system described by the new coordinate Ψ is:

$$\frac{dV(\Psi)}{dt} = -\frac{\partial V(\psi_1, \psi_2, ..., \psi_n)}{\partial(\psi_1, \psi_2, ..., \psi_n)} \tag{4.8}$$

Transfer the equation into the original coordinate Θ, we get:

$$\frac{d\theta_i}{dt} = -\frac{\partial V(\Psi)}{\partial \theta_i} = -\frac{\partial V(\Psi)}{\partial(\psi_i)} \frac{\partial \psi_i}{\partial \theta_i} \tag{4.9}$$

Then we can expend (4.9) into:

$$\frac{d\theta_i}{dt} = \begin{cases} -2\mu_1(\theta_1 - \tilde{\theta}_1) - 2\mu_2(\theta_1 - \theta_2 - \tilde{\theta}_1 + \tilde{\theta}_2), & i\text{=1} \\ 2\mu_{n-1}(\theta_{n-1} - \theta_n - \tilde{\theta}_{n-1} + \tilde{\theta}_n) & \\ -2\mu_n(\theta_{n-1} - \tilde{\theta}_{n-1}), & i\text{=n-1} \\ 2\mu_{i-1}(\theta_{i-1} - \theta_i - \tilde{\theta}_{i-1} + \tilde{\theta}_i) & \\ -2\mu_i(\theta_i - \theta_{i+1} - \tilde{\theta}_{i-1} + \tilde{\theta}_{i+1}), & \text{otherwise} \end{cases} \tag{4.10}$$

Finally, the gait generator model for the chain-type CPG-network is obtained as follows:

$$\begin{bmatrix} \dot{\varphi}_1 \\ \dot{\varphi}_2 \\ \vdots \\ \dot{\varphi}_n \end{bmatrix} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{bmatrix} + A \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_n \end{bmatrix} + B \begin{bmatrix} \tilde{\theta_1} \\ \tilde{\theta_2} \\ \vdots \\ \tilde{\theta_{n-1}} \end{bmatrix} \tag{4.11}$$

where A is,

$$A = \begin{bmatrix} -\mu_1 & \mu_2 & & & 0 \\ \mu_2 & -2\mu_2 & \mu_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-1} & -2\mu_{n-1} & \mu_{n-1} \\ 0 & & & \mu_n & -\mu_n \end{bmatrix}_{n\times n} \tag{4.12}$$

and B is,

$$B = \begin{bmatrix} 1 & & & 0 \\ -1 & 1 & & \\ & -1 & \ddots & \\ & & \ddots & 1 \\ 0 & & & -1 \end{bmatrix}_{n\times(n-1)} \tag{4.13}$$

ω_i in (4.11) is the integration constant, which is also the frequency of the CPG signal. To output the same frequency signals, we set $\omega_1 = \omega_2 = ... = \omega_n$. The convergence rate of the system is decided by the matrix A, which increases with the value of μ_i.

Next, two PD controllers are adopted to ensure the convergence of the amplitude [231] and the amplitude bias,

$$\ddot{r}_i = a_i[\frac{a_i}{4}(R_i - r_i) - \dot{r}_i] \tag{4.14}$$

$$\ddot{c}_i = a_i[\frac{a_i}{4}(C_i - c_i) - \dot{c}_i] \tag{4.15}$$

where, the parameter R_i determines the stable amplitude. The parameter C_i dominates the biased value of the stable amplitude. r_i, c_i are the amplitude and the amplitude bias of the i^{th} oscillator, respectively. In summary, the single neuron CPG model can be written as,

$$\begin{cases} \dot{\varphi}_i &= \omega_i + A\{i,:\}\cdot\Phi + B\{i,:\}\cdot\tilde{\Theta} \\ \ddot{r}_i &= a_i[\frac{a_i}{4}(R_i - r_i) - \dot{r}_i] \\ \ddot{c}_i &= a_i[\frac{a_i}{4}(C_i - c_i) - \dot{c}_i] \\ x_i &= c_i + r_i\sin(\varphi_i); \end{cases} \tag{4.16}$$

where $A\{i,:\}$ is the i^{th} row vector in (4.12), $B\{i,:\}$ is the i^{th} row vector in (4.13). Φ is the vector of the CPG neurons' phase, and $\tilde{\Theta}$ is the vector of the phase difference among the CPG neurons. The

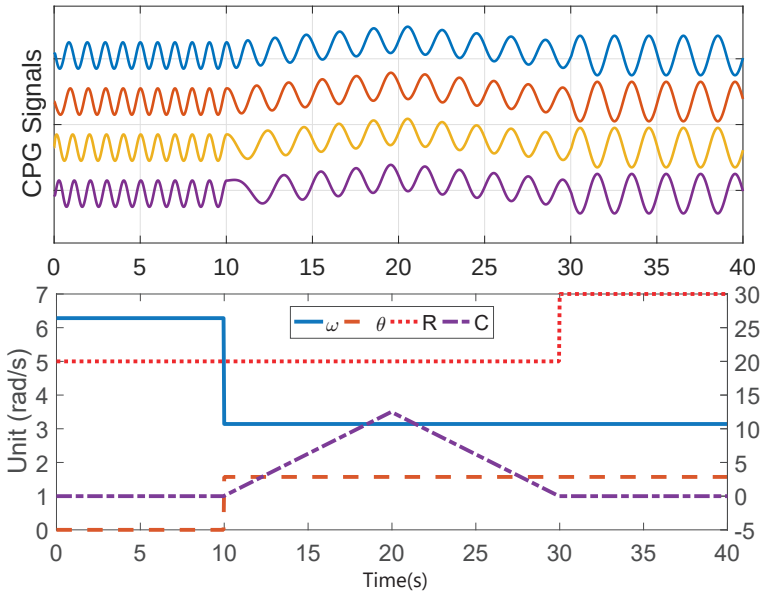


Figure 4.2: CPG output waves with changing parameters. The upper figure shows four CPG output signals. The bottom figure shows different sets of CPG parameters. Phase difference changes from 0 to π, with θ changing from 0 to π, at $t = 10s$. Frequency changes from 1Hz to 0.5Hz, with ω_i changes from 2π to π, at $t = 10s$. Amplitude bias rises up from 0 to 12 at $t = 20s$, and then falls off to 0 at $t = 30s$. Amplitude goes up from 20 to 30 at $t = 30s$

state variable φ_i is the phase of the i^{th} oscillator, respectively. The positive constants a_i and b_i are used to adjust the convergence speed of the amplitude. The variable x_i is the rhythmic output signal integrated by the phase ϕ_i, the amplitude r_i, and the amplitude bias c_i. To evaluate the output signals of the proposed CPG model, we plot four CPG outputs by adjusting the parameters with time, as seen in Figure 4.2. The CPG network changes the phase difference θ from 0 to π, the frequency ω from 0.5Hz to 1Hz, respectively, at $t = 10$, then linearly changes the amplitude bias C from 0° to 15°, between $t = 10$ and $t = 20$. The amplitude R is changed from 20° to 30°, at $t = 30$.

To evaluate the convergence speed of the proposed CPG model, we define the convergence tolerance as (4.17), where n is the number of CPG neurons and the target phase difference among the CPG network is set as 0. Therefore, the signals will reach the peak value at the same pace in finite time. When the tolerance is below 1%, the convergence is treated as finished.

$$tolerance = \sum_{i=1}^{n-1} (\frac{1}{n-1} \mid (x_i - x_j) \mid), \quad j = i+1 \tag{4.17}$$

The parameter μ in (4.12), is related to the speed of the synchronization. Based on the tolerance definition in (4.17), the relationship between the convergence rate and the parameter μ is shown in Figure 4.3. Therefore, we can conclude that the convergence rate increases with μ.

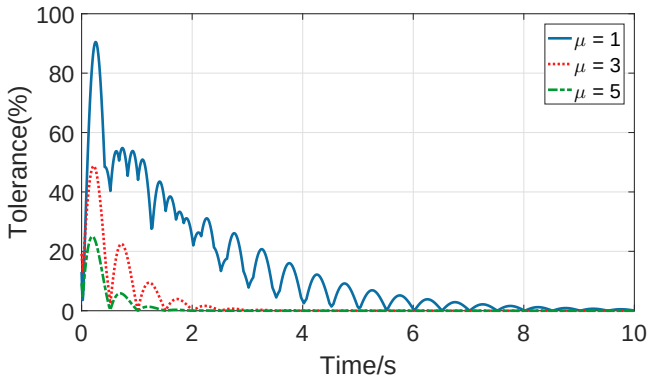


Figure 4.3: The synchronization speed of a three-CPG network changes with the parameter μ. Y axis is the tolerance between adjacent CPG.

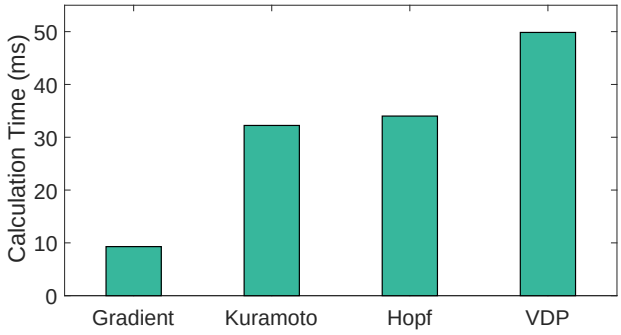
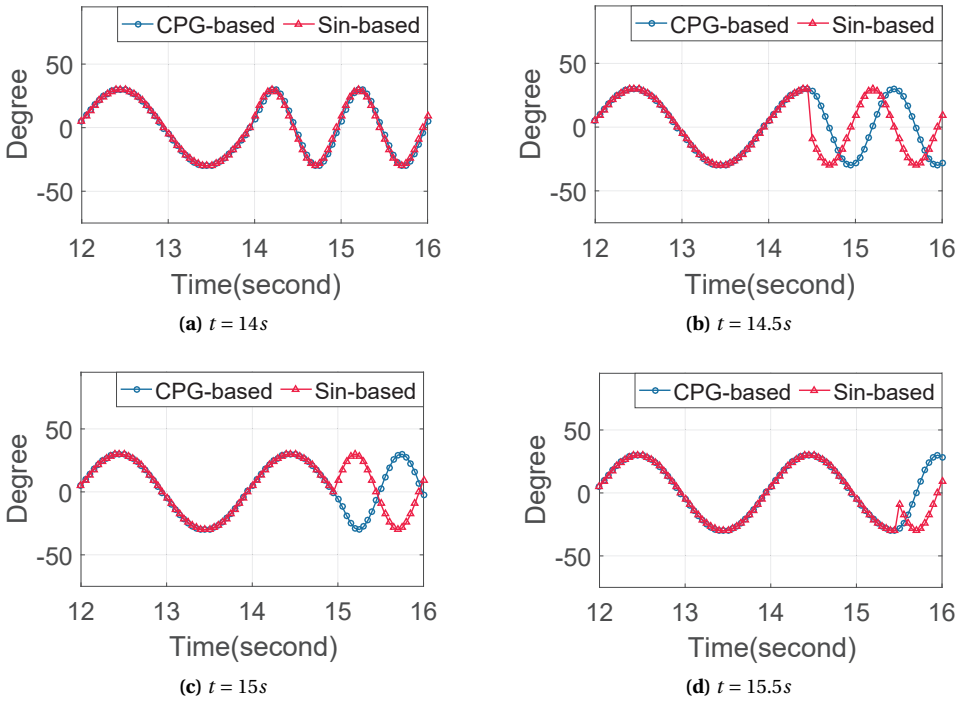


Figure 4.4: The execution time on Atmega 328 for 10 iteration of four kinds of CPG models. Each iteration step is 5ms. Gradient, Kuramoto, Hopf, and VDP are short for the CPG models based on the Kuramoto oscillator, the Hopf oscillator, and the Van der Pol oscillator, respectively.

4.2 Comparison with other CPG models

Those widely adopted CPG models are based on different nonlinear oscillators; for example, the Kuramoto oscillator model was adopted by Ijspeert and Shugen Ma [231] [232], the Hopf oscillator model was adopted by Seo and Slotine [233], and the Van Der Pol oscillator was adopted by Yu [234]. Many of these models can generate waves with desired properties, like frequency, amplitude and phase difference. However, the execution time on MCU (Micro Control Unit) is rarely investigated, a critical factor that influences the performance of the CPG model in real-life implementations. To examine the advantage of the proposed CPG model, experiments are conducted on an Arduino Nano board(Atmel Atmega 328) to inspect the online execution time of those CPG models.

The fourth-order Runge-Kutta is adopted to solve the differential equations of a three-CPG network. To ensure the solution accuracy of coupled differential equations, we set the CPG output pe-



(a) $t = 14s$ **(b)** $t = 14.5s$

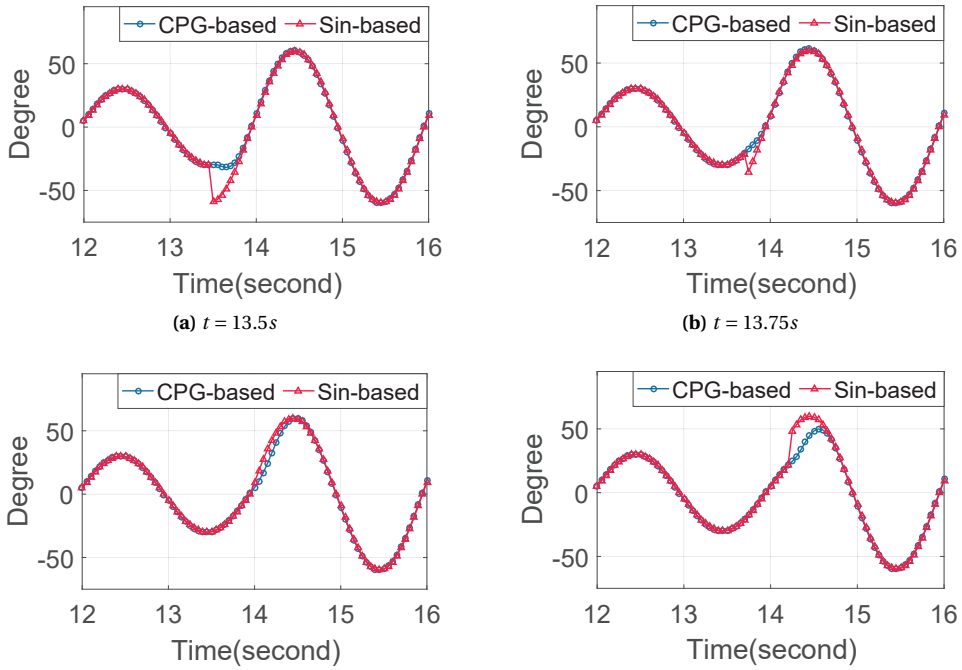
(c) $t = 15s$ **(d)** $t = 15.5s$

Figure 4.5: Changing in frequency from 1Hz to 2Hz, at four different times in a single period.

riod as $50ms$ and the step-size as $5ms$. The online executing experiment results are shown in Figure 4.4. We can observe that the online execution time for ten steps of the proposed CPG model is about $10ms$, which can easily satisfy the required signal output period. Since the total online execution involves on-bus data transmit, other CPG models may consume more that 50 milliseconds, which means a potential failure of output period. Therefore, the conclusion can be made that the proposed CPG model is lighter in weight compared to other CPG models.

4.3 Smooth Transition of Slithering Gait

This section will discuss the effectiveness of the control method based on our proposed CPG model for smoothing the control signals during gait transition contrasted with the sinusoid-based method. The slithering gait is then modelled and achieved, based on our CPG method for further evaluation in simulations and experiments.



(a) $t = 13.5s$ **(b)** $t = 13.75s$

(c) $t = 14s$ **(d)** $t = 14.25s$

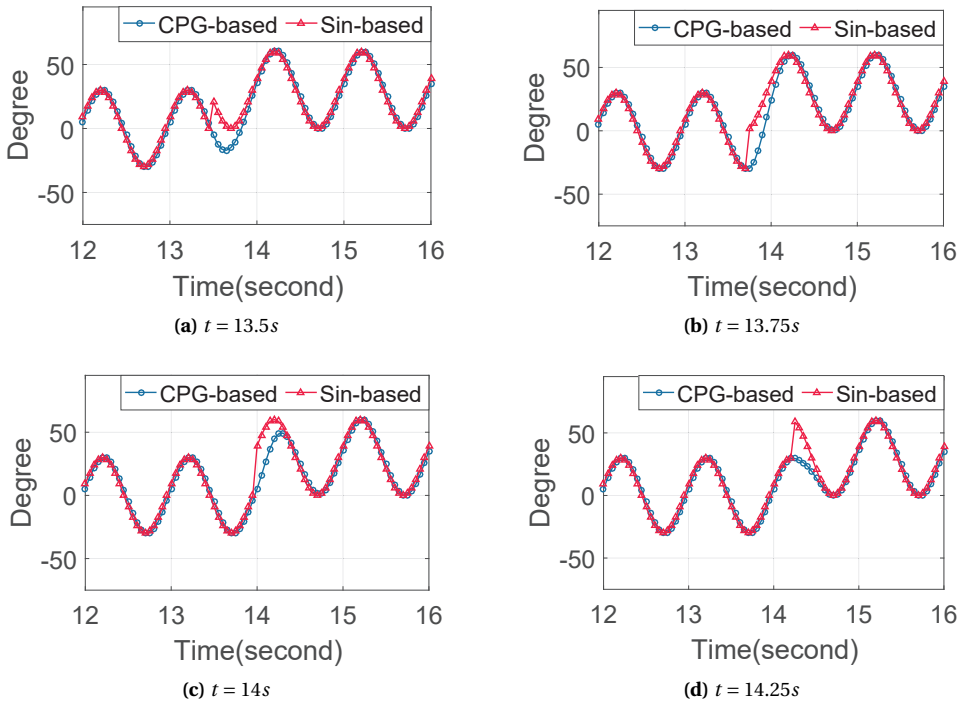
Figure 4.6: Changing in amplitude from 30° to 60°, at four different times in a single period.

4.3.1 Gait transition analysis

As presented in the literature, a smooth transition is dependent on maintaining the gait properties during the transition, not maintaining gait properties will cause undesired movement. Normally, the transition process requires a change of parameters of control signals, including frequency, amplitude, amplitude bias, and phase difference. However, the sinusoid-based control method inherently depends on time that can not be changed continuously. During the gait transition process, each joint position will change to another sinusoid wave, which probably has different values and direction.

In contrast, the CPG-based control method can generate continuous output signals due to self-adjusting ability. As a result, when the transition occurs, the CPG can smoothly switch to the new state of parameters. The numerical analysis results are shown in Figure 4.5, Figure 4.6, and Figure 4.7.

In Figure 4.5(a)-(d), the frequency of the signal is changed from π to 2π, at $t = 14s$, $t = 14.5s$, $t = 15s$, and $t = 15.5s$, respectively. Since the sinusoid-based method inherently relies on time, it generates different discontinuous waves at different times. An obvious set-point jump over 30° and a change of wave traveling direction can be observed in Figure 4.5(b) and Figure 4.5(c). In contrast, our CPG-based method exhibits a continuous transition process, no matter when the transition occurs.



(a) $t = 13.5s$ **(b)** $t = 13.75s$

(c) $t = 14s$ **(d)** $t = 14.25s$

Figure 4.7: Changing in amplitude bias from 0° to 30°, at four different times in a single period.

In Figure 4.6(a)-(d), the amplitude of the signal is changed from 30° to 60°, at $t = 13.5s$, $t = 13.75s$, $t = 14s$, and $t = 14.25s$, respectively. As shown in Figure 4.7(a)-(d), the discontinuity can also be observed when changing the amplitude bias from $0_\circ$ to $30_\circ$, at $t = 13.5s$, $t = 13.75s$, $t = 14s$, and $t = 14.25s$, respectively. In general, although the discontinuity results are not as obvious as Figure 4.5, the CPG-based method still exhibits more smooth waves compared to the sinusoid-based method. According to the results, the time-varying discontinuous waves will cause undesired movement when all the errors are integrated to the joints of the robot. Therefore, it is meaningful to adopt the CPG-based method for scenarios with frequent gait parameter transitions.

4.3.2 Slithering gait transition

As a promising type of gait of a snake-like robot for autonomous locomotion, slithering is adopted by snakes to move forward with a S-like shape. The challenge to utilize this gait, is stability due to the relatively small base, compared to other gaits such as sidewinding or rolling. Especially in an object tracking scenario, the snake-like robot is required to change the locomotion speed and direction according to the target. The slithering gait is modeled in Chapter 3.

In order to track and follow moving objects, the snake-like robot is required to change locomo-

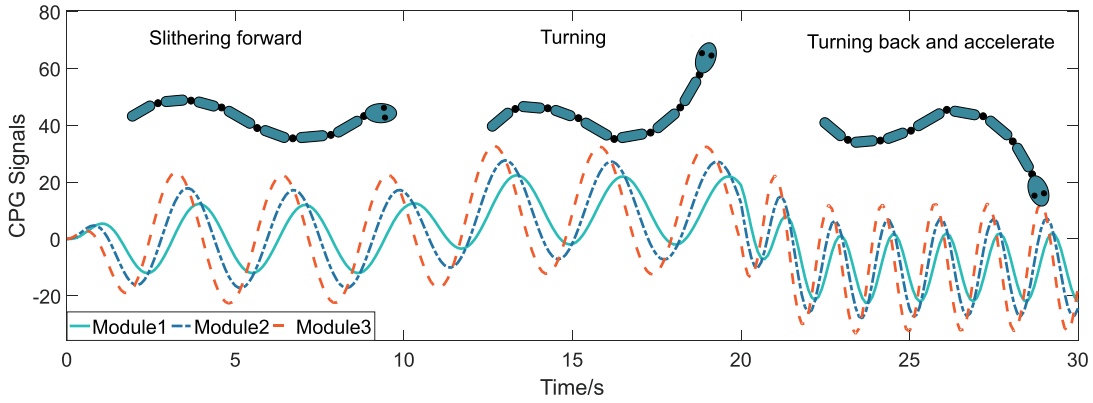
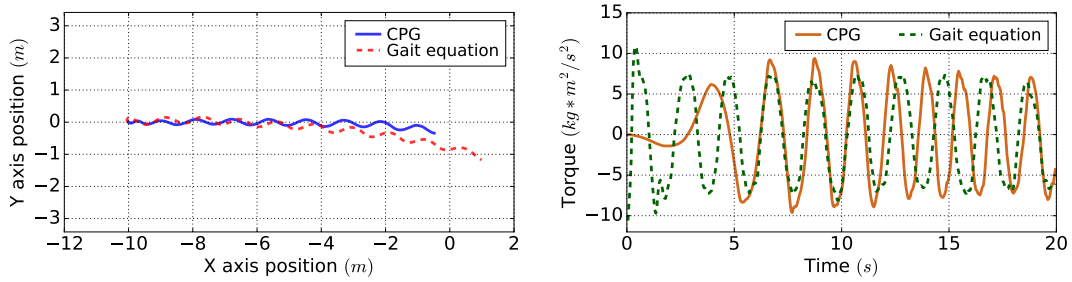


Figure 4.8: Illustration of the first three CPG outputs when the snake-like robot performs slithering gait. The robot slithers forward until $t = 10s$, then turns to left until $t = 20$, finally turns back and accelerates.

tion speed and direction. These are directly related to the frequency and amplitude bias in (7.1). The slithering gait transition process is depicted in Fig. 4.8 and the corresponding CPG signals are listed below. The CPG output waves for the first three modules are represented by the solid line, dotted line, and dash line, respectively. These CPG commands can make the snake-like robot slither forward before $t = 10s$. Then, the snake changes direction to the left by increasing amplitude bias C_{odd} in (7.1). At $t = 20s$, the robot accelerates and turns back by doubling the frequency ω_{even}, ω_{odd}, and decreasing amplitude bias C_{odd}. No obvious sudden change of output waves can be observed during the process.

4.4 Simulations

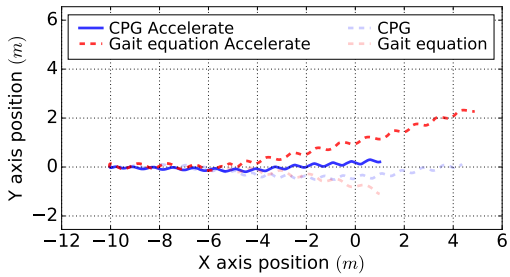


(a) Trajectory of slithering forward based on CPG method.

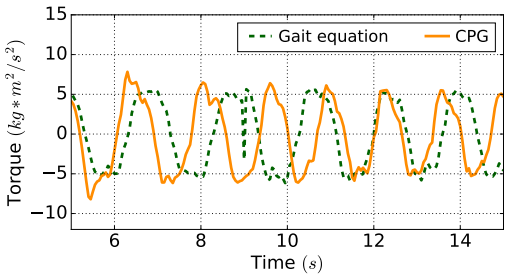
(b) Torque contrast of CPG and sinusoid based methods.

Figure 4.9: Trajectory and torque comparison for the starting of slithering.

Two aspects will be demonstrated in simulations via a scenario of tracking a moving target com-

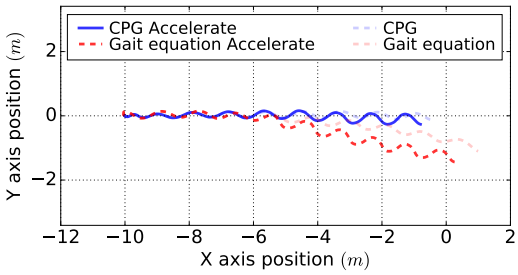


(a) Trajectory of slithering acceleration based on CPG method.

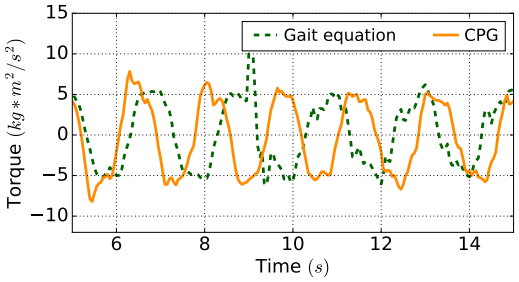


(b) The accelerating slithering torque contrast of CPG and sinusoid based methods, at $t = 9\ s$, from $\omega = \pi$ to $\omega = 1.2\pi$.

Figure 4.10: Trajectory and torque comparison for slithering acceleration.



(a) Trajectory for changing body shape based on CPG method.

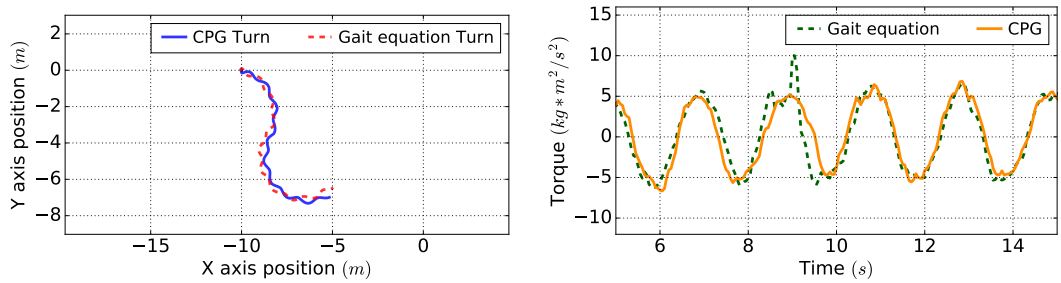


(b) Torque contrast of CPG and sinusoid based methods.

Figure 4.11: Slithering body shape change, at $t = 9\ s$, from $a = 40°$ to $a = 60°$.

paring the CPG-based method with the sinusoid-based method. The CPG-based control method has two advantages. First, the CPG-based control method can ensure smooth gait trajectory, when the body shape and locomotion speed are changed. Second, the CPG-based control method can effectively decrease the abnormal torque during transition.

The trajectories and torque comparison results are shown in Fig. 4.9, 4.10, 4.11, and 4.12. The slithering process are shown in Fig. 4.9(a) and Fig. 4.9(b). The trajectories of head and tail modules of the robot are represented by a solid and dash line, respectively. At the initial position, the snake robot lies along the x axis and the head coordinates are $(0,0)$. By comparing the two figures, one can see that the sinusoid-based method generates an inclination angle deviated from the body direction. However, the CPG-based method trajectory is almost aligned with the body direction indicating an accurate locomotion direction control effect. The torque curves are shown in Fig. 4.9(c). The dash line exhibits jitter phenomenon when the snake starts to move, the torque curve of the proposed



(a) Trajectory for turning direction based on CPG method.

(b) Torque contrast of CPG and sinusoid based methods.

Figure 4.12: Trajectory and torque comparison for the steering of slithering.

CPG-based method fluctuates slightly at initiation of movement.

During the tracking process, the snake-like robot is required to accelerate or decelerate. Fig 4.10 shows the acceleration process. The CPG-based method maintains the direction of movement when doubling the forward speed (Fig. 4.10(a)). On the contrary, the sinusoid-based method deflects to another direction when accelerating (Fig. 4.10(b)). Meanwhile, a sudden change of the torque curve can be observed in Fig. 4.10(c).

To increase the throughput capacity for narrow terrain, a change in body shape is necessary. By adjusting the amplitude, the snake robot can contract the body, as shown in Fig. 4.11. After the transition, the trajectories of the robot shrink together and continue moving forward. In contrast to the CPG-based method (Fig. 4.11(a)), the sinusoid-based trajectories (Fig. 4.11(b)) deviate slightly from the previous direction. The torque generates a jerky jump of set points, as shown in Fig. 4.11(c).

Another requirement for tracking locomotion ability is turning direction (Fig. 4.12). The snake-like robot turns to the left when $t = 30$. Although it is not so obvious, the discontinuity of the body trajectories are still apparent, as compared to Fig. 4.12(a) and Fig. 4.12(b). The sharp edge of the trajectories in the enlarge figure is not acceptable, especially for unstable direction turning locomotion. As expected, the torque curves (Fig. 4.12(c)) demonstrate an abnormally large value.

Chapter 5

High-Level Control: Spiking Neural Networks

As the core component and the foundation of the nervous system, high-level controller is responsible for perceiving information from environment and making action decision accordingly. Due to the superiorities of SNNs as introduced in Chapter 2, we will adopt SNNs as the high-level controller to further implement autonomous locomotion tasks. Therefore, in this chapter, we will first introduce the modeling of our SNN, the encoding and decoding strategy, and its learning rule.

5.1 Neuron Model

As the functional and structural unit of an SNN, neurons are responsible for information processing by receiving, processing, generating, and propagating spikes within its network. There are many neuron models mimicking the function characteristics of biological neurons, such as integrate-and-fire (IF) neuron, Hodgkin-Huxley neuron, and leaky integrate-and-fire neuron. These models are based on the assumption that the timing of spikes rather than the specific shape carries neural information. In the scope of this thesis, we adopt the leaky integrate-and-fire (LIF) neuron model with alpha-function shaped synaptic currents to construct our SNNs.

Generally, an SNN consists of many synapses that connect presynpatic neurons to a postsynaptic neurons. One example is given in Figure 5.1, where pre-neuron i is connected to post-neuron j and its synaptic strength is w_{ij}. Then this connection can be modeled as the sequence: the spike train generated by the pre-neuron will trigger a synaptic electric current into the post-neuron.

First, a sequence of firing times are called spike trains and can be described as

$$S(t) = \sum_{t} \delta(t - t^f), \tag{5.1}$$

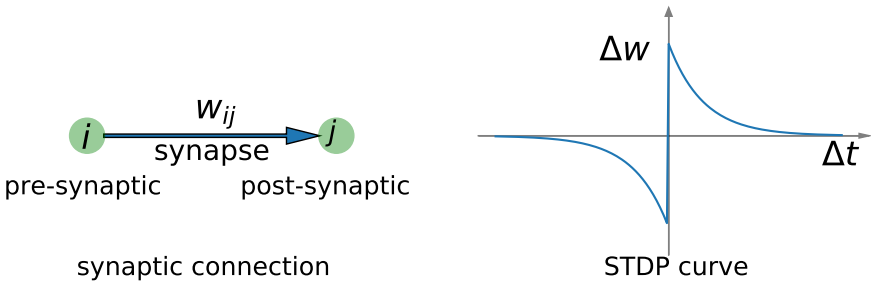


Figure 5.1: An example of a synaptic connection and the STDP curve. The learning curve of the STDP mechanism with $A_+ = 1.0$ and $A_- = 1.0$. τ_+ and τ_- setting as $20ms$ in (5.5). If $\delta_t > 0$, STDP exhibits a potentiation effect on the synapse connection. Otherwise, it shows a depression effect.

where $f = 1, 2, ...$ is the label of a spike and $\delta(\cdot)$ is a Dirac function.

Passing a simplified synapse model, the incoming spike train will trigger a synaptic electric current into the postsynaptic neuron. This alpha-function shaped synaptic current induced by a spike train $S_j(t)$ can be described as

$$i(t) = \int_0^\infty S_j(s-t)\exp(-s/\tau_s)ds. \tag{5.2}$$

Here, τ_s denotes the synaptic time constant. In the literature, a function with the form $x \cdot exp(-x)$ is often called an α-function.

The postsynaptic current then charges the LIF neuron model. Mathematically we write

$$C\frac{du}{dt} = u_{res} - \frac{1}{R}u(t) + (i_0(t) + \sum w_j i_j(t)) \tag{5.3}$$

to describe the effects on the membrane potential u over time. This can be explained from the perspective of electronics, which corresponds to charging a capacity C with current I through the resistance R. $i_0(t)$ denotes an external current driving the neural state, $i_j(t)$ is the input current from the j^{th} synaptic input and w_j represents the strength of the j^{th} synapse. Meanwhile, this circuit will slowly leaks the voltage away over time. Once u crosses a certain threshold u_{th}, the neuron will fire a spike δ and its membrane potential u is set back to u_{res}. Inspired by the refractory period of biological neuron, the LIF neuron model will turn to an inactive state and can not be charged again.

Figure 5.2 shows the neural activity of a LIF neuron driven by external injected current and synaptic current. The time-based external injected current $i_0(t)$ is shown by blue solid line. As we can observe, the membrane potential $u(t)$ (green line) of the LIF neuron increases under the effect of $i_0(t)$ and leaks away all the time. Once $u(t)$ crosses the threshold V_{th}, it will emit a spike (black bar) and maintain at the reset value u_{res} during the refractory time τ_{ref}. On the other hand, under the effect

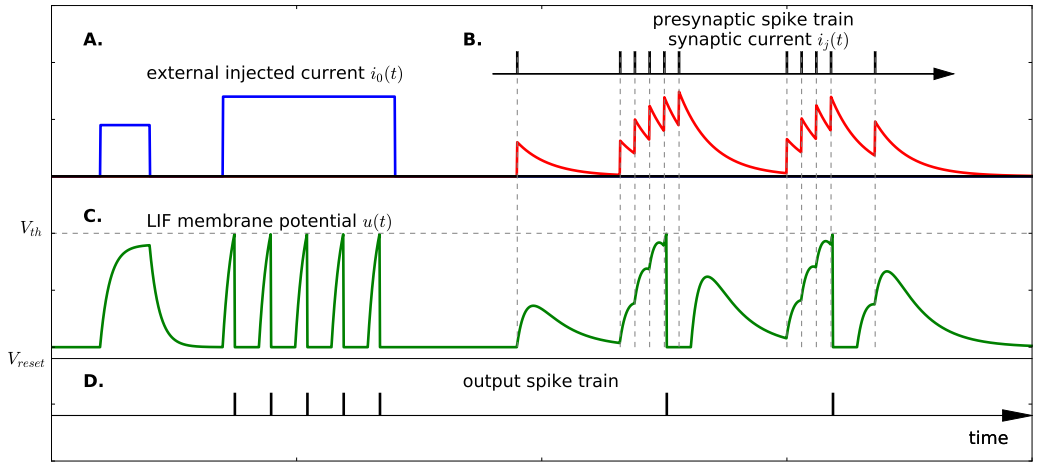


Figure 5.2: The neural activity of a LIF neuron driven by the external injected current $i_0(t)$ or the synaptic current from the j^{th} neuron. All these signals are time-based. **A.** The external injected current $i_0(t)$. **B.** The incoming spike train from j^{th} neuron and its synaptic current $i_j(t)$. **C.** The membrane potential $u(t)$. **D.** The output spike train generated by the LIF neuron.

of the spike train (black bar) from the presynaptic neuron, the synaptic current also has an impact on $u(t)$. As we can see, the synaptic current $i_j(t)$ (red line) rises as long as there is a spike and decays under the effect of an alpha function. Since we build and simulate our SNN in NEST [235], the neuron model iaf_psc_alpha is used to implement a leaky integrate-and-fire model with alpha-function shaped synaptic currents.

5.2 Synaptic Plasticity

Once the neuron and synaptic models are decided on, the synapse plasticity model should be carefully chosen to dynamically change those neuron connections inside and among the layers of SNNs. Synaptic plasticity is the ability of synapses to strengthen or weaken their activities over time and is used to optimize the performance of an SNN for a given task. The process of parameter adaptation is called *learning* and the procedure for adjusting the weights is referred to as a *learning rule*. Therefore, a biological plausible model of synaptic plasticity will be discussed in this section.

5.2.1 STDP Learning Rule

There are several learning rules proposed based on the accumulated biological findings on synaptic plasticity, from LTP to R-STDP. As the most important theory in neuroscience explaining the adaption of synaptic efficacies in the brain during the learning process, the spike-timing-dependent plasticity (STDP) learning rule [236] has been successfully proven by neuroscience experiments [237, 238] (see

[239] for a review; [240] discusses more recent in-vivo results).

For this study, the weight update rule under STDP as a function of the time difference between pre- and postsynaptic spikes is defined as:

$$\Delta t = t_{post} - t_{pre} \tag{5.4}$$

$$W(\Delta t) = \begin{cases} A_+ e^{-\Delta t/\tau_+}, \text{ if } \Delta t \geq 0 \\ -A_- e^{\Delta t/\tau_-}, \text{ if } \Delta t < 0 \end{cases} \tag{5.5}$$

$$\Delta w = \sum_{t_{pre}} \sum_{t_{post}} W(\Delta t) \tag{5.6}$$

, where w is the synaptic weight, Δw is the change of the synaptic weight, and t_{pre} and t_{post} stand for the timing of the firing spike from pre-neuron and post-neuron. A_+ and A_- represent positive constants scaling the strength of potentiation and depression, respectively. τ_+ and τ_- are positive time constants defining the width of the positive and negative learning window.

5.2.2 R-STDP Learning Rule

Because of the absence of direct goals, correction functions or a knowledgeable supervisor, STDP-based learning is usually used for unsupervised learning tasks and not well suited for filed applications that lack of experience or knowledge in advance. On the other hand, neuroscience studies show that the brain modifies the outcome of STDP synapse using one or more chemicals emitted by a given neuron and known as neuromodulation. While, as one of the key substrates for the brains functions founded in some brain structures, neural modulation signals offers field robots an inspirational way to learn and evolve.

A learning rule that incorporates a global reward signal in combination with STDP was proposed by Izhikevich [132] and Florian [133] and called reward-modulated spiking-timing-dependent plasticity (R-STDP). It is based on the standard rule for STDP, modeling the synaptic weight change in dependence on the time difference between firing times of pre- and post-synaptic neurons (Figure 5.1). Since R-STDP can modulate an SNN with external signal that even sparse or delayed, this method is well suited for mobile robotic tasks. Instead of instantaneously applying these weight changes to synapses as done in STDP, these changes are collected in an eligibility trace to serve those delayed reward signals. The eligibility function is inspired by the residence time distribution, since the neuromodulatory is chemical signal. Actually, there are evidences that humans utilize the eligibility trace to solve the credit-assignment problem when learning from sparse and delayed reward [241]. And R-STDP can reinforce firing patterns occurring on a millisecond timescale even when they are followed by reward that is delayed by seconds.

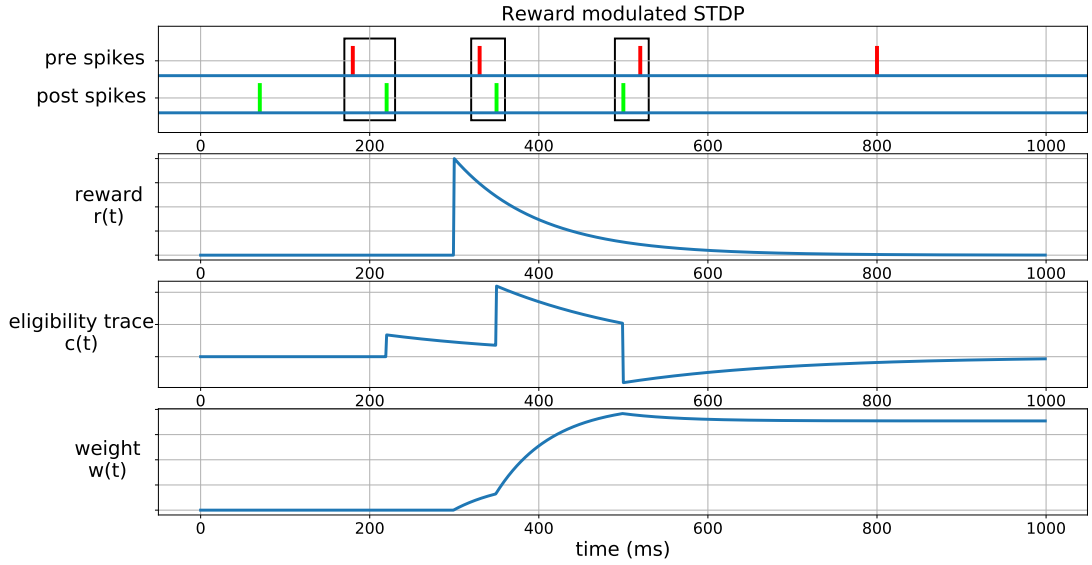


Figure 5.3: Learning curve of the R-STDP mechanism extended from [242]. The spike trains from the pre- and postsynaptic neuron are shown with red and green vertical bars. The time-based reward signal is shown in the second subplot. Under the effect of R-STDP, the eligibility trace is plotted according to the first two subplots. Consequently, the weight of the synapse is presented in the last subplot.

In the R-STDP, the synaptic weight w changes with the reward signal R. The eligibility trace c of a synapse can be defined as,

$$\dot{c}(t) = -\frac{c}{\tau_c} + W(\Delta t)\delta(t - s_{pre/post})C_1 \tag{5.7}$$

where the contribution of all spike pairings with the second spike time at $t - s_{pre/post}$ is modeled in Figure 5.3. $s_{pre/post}$ means the time of a pre- or post-synaptic spikes. $c(t)$ decays exponentially with the time constant τ_c. The decay rate controls the sensitivity of plasticity to delayed reward. C_1 is a constant coefficient to modulate the decay rate. Δ is the Dirac delta function. W is the synaptic change under STDP mechanism defined in (5.5).

One of the key problems in reinforcement learning is that the receipt of a reward is often delayed after the action or neural pattern (See Figure 5.3) [243]. By combining the eligibility trace with the global reward signal, a simple learning rule can be defined changing the weight of a synapse in proportion to the product of eligibility trace and reward signal, defined as

$$\dot{w}_{ji}(t) = r(t) \times c_{ji}(t), \tag{5.8}$$

where w_{ji} is the weight of a synapse from neuron i to neuron j, $c_{ji}(t)$ is an eligibility trace of this synapse which collects weight changes proposed by STDP, and $d(t)$ results from a neuromodulatory signal.

Figure 5.3 shows the learning curve of the R-STDP mechanism. In the first subplot, the pre- and post spike trains are displayed with red and green bars. Three groups of spikes in the correlation time period are outlined with black boxes. Under the STDP effect of pre- and post spike pairs, the eligibility trace $c(t)$ is shown with blue line in the third subplot. The time-based reward signal $r(t)$ is shown in the second subplot. The weight $w(t)$ is plotted with blue line according to (5.8). Specifically, the first spike pair exhibit a potentiation effect, therefore, the eligibility trace $c(t)$ increases under the STDP mechanism. Meanwhile, since the reward signal $r(t) = 0$, the weight $w(t)$ does not change. For the second spike pair, the synapse weight increases under the combined action of $r(t)$ and $c(t)$ according to (5.8). Because the third spike pair results in a depression effect, the eligibility trace $c(t)$ decreases accordingly. Therefore, the weight $w(t)$ is weekended as well.

5.3 Information Encoding

The term neural encoding refers to representing information from the physical world (such as direction of a moving stimulus) in the activity of a neuron. Therefore, it is essential to investigate how to encode different types of sensory information into spike trains. A number of neural information encoding strategies have been proposed, such as binary encoding [71], population encoding [72], temporal encoding [244], and the most commonly used rate encoding [73]. In this section, we will take target tracking task as an example to introduce some common sensor information encoding methods.

5.3.1 Vision Sensor Based Binary Encoding

Just as vital as the visual system to living creatures, the vision sensor is one of the most effective and widely used sensors for field robots to perceive information of the environment. In an autonomous obstacle climbing experiment, Arena *et al.* [167] used a vision sensor to acquire the height information of an obstacle and then controlled an insect robot to step over it. Similarly, Floreano [168] equipped a Khepera robot with a linear camera to sense the wall with black and white vertical stripes of random size at irregular intervals.

In this work, we also use the vision sensor to perceive some of the environmental information for performing autonomous tasks (See Chapter 6). Take RGB image as an example for information encoding, one solution is to transfer the image pixel value into spike trains with different firing rate. For reasons of simplicity, we first blur it into lower resolution according to the input neuron number of the SNN. Then, the image is filtered by extracting one specific color, e.g. the red channel, as shown in Figure 5.4. Every pixel has an intensity value in the range of [0, 255] according to its RGB value. This

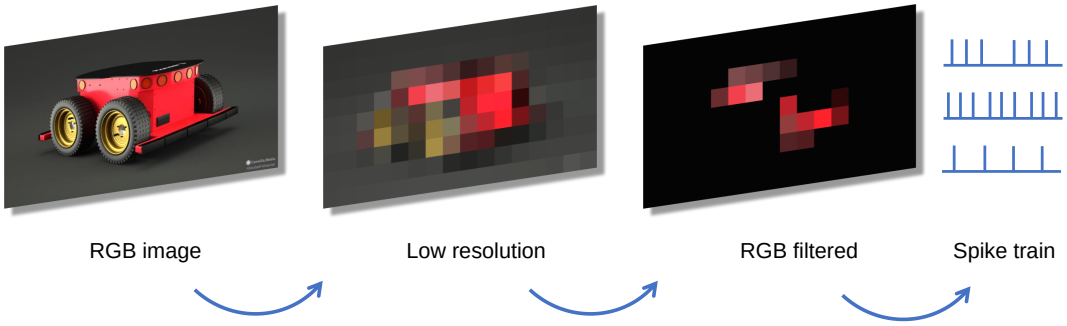


Figure 5.4: Steps for encoding an image into spikes. A. The blurred image with a resolution. B. The filtered image, with reds as the target. C. The mask of the filtered image. D. The spikes for the input layer of the SNN.

value is then normalized and multiplied by the maximum firing rate. Finally, the value is transferred into Poisson-distributed spike trains by setting as the mean firing rate.

5.3.2 Proximity Sensor Based Continuous Encoding

Another crucial sensing method for living creatures is to measure or sense the world by auditory sense, for example, the auditory system of the barn owl is able to locate sources of sound in the horizontal plane with a precision of 1 to 2 degrees, which equates to a temporal difference of only a few microseconds ($5\mu s$) between the arrival of soundwaves at the left and right ears [55]. Inspired by this, distance measuring equipments, such as sonar or radar, are developed for performing navigation and ranging tasks. As for field robots, proximity sensor is mainly used to achieve similar functions and returns the distance to the target in the effective range.

Take proximity sensor as an example for rate encoding (See Figure 6.12), the strategy is to stimulate an IF neuron with the sensor readings as

$$\frac{du_i}{dt} = \frac{a \times x_i + b}{ms}. \tag{5.9}$$

The parameter b is used to enable the input neuron firing a spike every $\frac{1}{b}ms$, even when there is no stimulus. This is essential since the proximity sensor may detect nothing within its range. For the network itself, this serves the purpose of helping to generate spikes for low input values in the time window and thus enabling learning via STDP for inputs that would otherwise not have fired the input neurons. With the factor a, the build-up of the membrane potential can be scaled, limiting the amount of spikes generated in T for a maximum input to $(a+b) \times T$. One example is given in Figure 6.12, where a is set to 0.2 and $b = 0.025$. This results in generation of one spike per time window for minimal/no input and 11 spikes for maximum input.

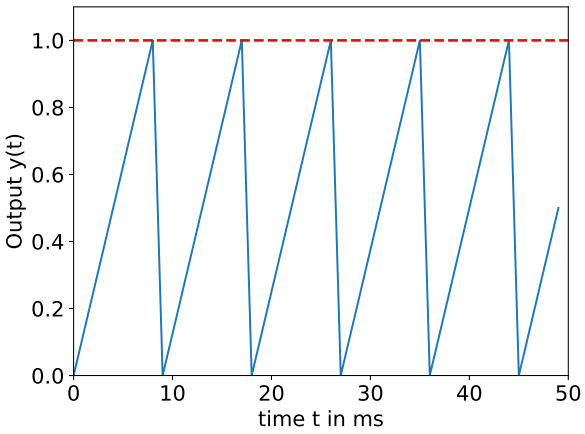


Figure 5.5: An example of encoding continuous value from proximity sensor into spikes. The red dash line is the firing threshold and the blue line describes the membrane potential. The parameters are $a = 0.2$, $b = 0.025$ and $x_i = 0.5$ in a time window $T = 50ms$.

5.3.3 Neuromorphic Vision Sensor

As one of the neuromorphic hardware, the dynamic vision sensor (DVS) is a biologically inspired vision sensor that can mimic the retina and generate spikes in response to the pixel-level changes in illumination caused by movement. Compared to a conventional frame-based camera, a DVS offers great advantages in terms of data rate, speed, and dynamic range [245], especially for mobile scenes. Moreover, spikes generated by a DVS can be directly fed into Spiking Neural Networks (SNNs) for vision processing and motion control. The DVS is suitable for autonomous locomotion control of the robots that work in critical scenarios with either bad light conditions or rough terrain.

Using DVS camera in simulation is not as good as a real one, since the events are acquired by comparing the intensity of illumination of the pixels in consecutive image frames. Unfortunately, this means that pixel events are not independent anymore and are only emitted in batches at every time-step with the same time stamp. Therefore, the simulation time-step length determines the dvs time resolution as well. Usually, this will set the DVS time resolution to a much lower frequency than a real device, which is a drawback of this kind of DVS simulation. In order to get more simulated DVS data, we accumulate ten consecutive DVS frames and count every event regardless of the polarity. An example is shown in Figure 5.6, a lane track is sensed by a simulated DVS.

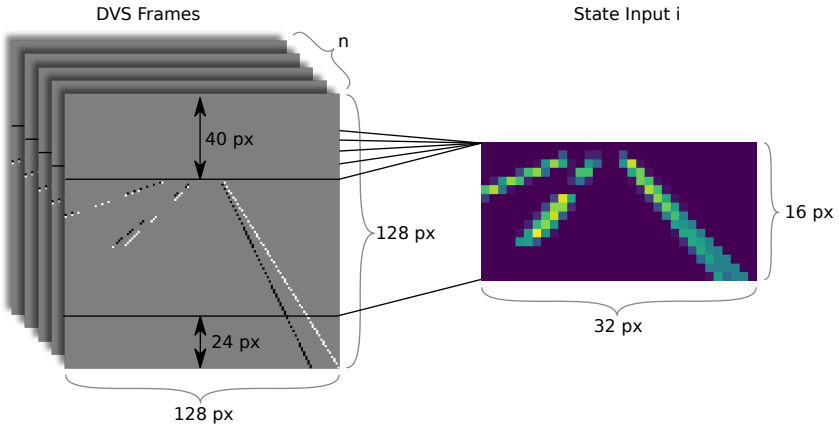


Figure 5.6: Conversion of consecutive DVS frames into state input for reinforcement learning. This is done by dividing the original 128 × 128 DVS frames into small 4 × 4 regions and counting every event over consecutive frames regardless of increase or decrease illumination. Furthermore, the image is cropped at the top and bottom resulting in a 32 × 16 image.

5.4 Motor Based Information Decoding

In the previous section we have studied how a neuron can encode information from different kinds of sensors. On the other hand, information decoding is the reverse process to interpret from neuron activity to electrical signal for actuators (such as muscle or motor). For living creatures, the brain generates the decoded control signals to actuate their muscles for desired actions. For robots, a motor usually serves as the actuator to generate locomotion. Therefore, we will also investigate how can a neuron decode the spikes to motor control signals.

For human beings, a body joint is usually controlled by the contraction and flexion of the agonist and antagonist muscle for the purpose of accuracy and rapidity. Since we steer our snake-like robot by regulating the amplitude bias as (3.8), it can also be regarded as steering a two-wheel car with differential turning radius. In this thesis, a steering wheel model based on an agonist-antagonist muscle system is inspired by [246]. Instead of turning angles, the amplitude bias is computed and added or subtracted for steering left or right and the frequency is adjusted for changing travel velocity as well .

First, the output spike count is scaled by the possible maximum output n_{max}:

$$m_t^{left/right} = \frac{n_t^{left/right}}{n_{max}} \in [0;\ 1] \tag{5.10}$$

$$n_{max} = \frac{T_{sim}}{T_{refrac}}, \tag{5.11}$$

where T_{sim} denotes the simulation time step length and T_{refrac} describes the refractory period length of the LIF neuron. Based on the difference of the normalized activities m_t^{left} and m_t^{right} and a turn constant k, the amplitude bias C is calculated as

$$C = k \cdot a_t \text{ , where } a_t = m_t^{left} - m_t^{right} \in [-1;\ 1]. \tag{5.12}$$

When the snake cannot see the target, the network will receive no input and both outputs will be zero. In this case the network should still move to the previous turning angle as it remembers, since the target get lost continuously in one side of the vision. To address this issue, the bias for the current step C_t is further calculated as

$$C_t = \tau \cdot C + (1 - \tau) \cdot C_{t-1}, \tag{5.13}$$

$$\text{with } \tau = \sqrt{\frac{(m_t^{left})^2 + (m_t^{right})^2}{2}}. \tag{5.14}$$

Thus far, we have a decoding method that translate the discrete network's spiking pattern into continuous motor control signals to steer a snake-like robot.

5.5 The Neurorobotic Platform

With the aforementioned knowledge, we can now build up our spiking neural network for performing autonomous locomotion tasks. In this section, we will introduce the implementation framework which is the neurorobotics platform (NRP) including constructing the SNN, running physical calculation, and communicating between the brain and the actuator of the robot. A screenshot of the snake-like robot in the NRP is presented in Figure 5.7.

Compared to the classic robot control paradigm, neurorobotics are controlled by simulated nervous systems which mimic the biological brain at different levels of detail. While existing approaches are able to simulate either biological realistic brain model or bio-inspired robots and their interacting world, there is so far no work to facilitate the easily establishment of an interactive research platform of robotics and neuroscience. We therefore developed the neurorobotics platform (NRP) in the scope of the European Union–funded Human Brain Project. The NRP facilitated a direct link between robotics and neuroscience, enabling a seamless and efficient exchange of knowledge between these two fields. In the following, we briefly describe the architecture and the key components of the NRP. For a detailed introduction of the NRP, we strongly refer to our report paper [188] and our official website[1].

[1] http://neurorobotics.net/

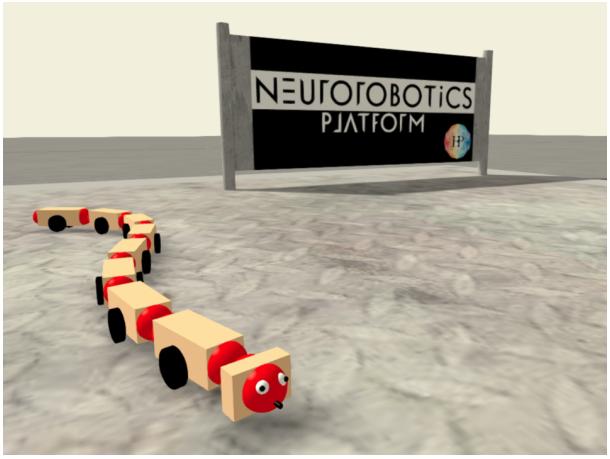


Figure 5.7: A screenshot of the snake-like robot inside NRP.

5.5.1 Functional Architecture of the NRP

The Neurorobotics Platform is composed of six key software components (see Figure 5.8) which are needed to construct neurorobotics experiments from scratch. It can be seen that the NRP provides a complete framework for coupled simulation of robots and brain models. The Brain Simulator simulates the brain by bio-inspired learning algorithms such as spiking neural network to control the robot in a silico neurorobotics experiment.The World Simulator simulates the robots and their interacting environment. The Brain Interface and Body Integrator (BIBI) builds a communication channel between brain models and robot models. The Closed Loop Engine (CLE) is responsible for the control logic of experiments as well as for the data communication between different components. The Backend receives requests from the frontend for the neurorobotics experiment and distributes them to the corresponding component, mainly via ROS. The Frontend is web-based user interface to neurorobotics experiments. Users are able to design a new experiment or editing and visualizing existing template experiments.

5.5.2 Key Components in the NRP

5.5.2.1 Brain Simulator

In the NRP, the Brain Simulator is implemented with spiking neural network (SNN) [29] to simulate a brain circuit. The current supported brain simulator is NEST [235], which is a simulator for spiking neural network models that focus on the dynamics, size and structure of neural systems rather than on the exact morphology of individual neurons. NEST is one of the most used neural simulations in the neuroscience community. It is supported through the use of PyNN [247] abstraction layer

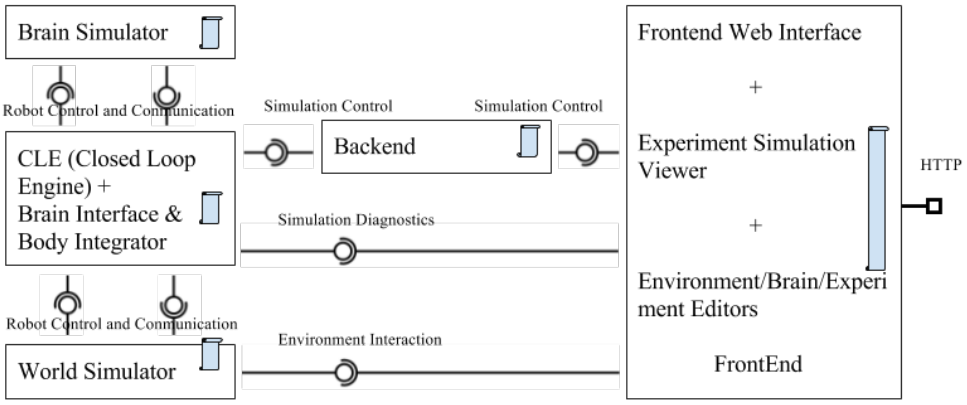


Figure 5.8: Functional architecture of the NRP consisting of six key software components: Brain Simulator, World Simulator, Brain Interface and Body Integrator, Closed Loop Engine, Backbend, and Frontend.

that makes it enjoy the same interface with different software simulators as well as neuromorphic processing hardware, e.g., dedicated computer architecture for the SNN models such as SpiNNaker.

5.5.2.2 World Simulator

In order to couple with the accurate brain simulation to perform a realistic neurorobotic experiment, a realistic simulation of the physic world for both the robot and the environment (in which the robot interacts) is required. The Gazebo is chosen and extended as the world simulator inside the NRP. It offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs. This dynamic simulation can be computed with different supported software libraries like ODE [177] and Bullet [248]. For communicating with the simulated robot, we use the Robot Operating System (ROS) [249] as a middle-ware. The platform uses the asynchronous event-based communication through ROS topics.

5.5.2.3 Brain Interface and Body Integrator

The Brain Interface and Body Integrator (BIBI) is the crucial component in the NRP because it establishes the connection between the robot and brain simulation. The BIBI is composed of a bunch of Transfer Functions which can be defined by the users. We defined two main types of TFs inside the NRP: the Robot to Neuron TFs (R2NTFs) and the Neuron to Robot TFs (N2RTFs). The R2NTFs translate the robot output signals such as images and joint states into neuron signals such as spikes, electric currents or firing rates (which satisfy the specific input formats of the neuron simulation). The N2FTFs convert the neural signals from neurons into the motor commands for robot motors. There-

fore, the perception-action loop of the interaction of the brain, body and environments is closed by these two types of transfer functions which fill the gaps between a neural controller and the robot sensory output.

5.5.2.4 Closed Loop Engine

The Closed Loop Engine orchestrates the two simulations and the data transfer. It is responsible for the synchronization and data exchanges among the above three components. The CLE guarantees that both simulations run at the same time-step, and the TFs run at the end of the simulation steps. A typical execution of a time-step is: after the physics and neural simulations have completed their execution in parallel, the TFs receive and process data from the simulations and produce an output, which is the input for the execution of the next step. The idea behind the proposed synchronization mechanism is to let both simulations run for a fixed time step, receiving and processing the output of the previous steps and yielding data that will be processed in the future steps by the concurrent simulation [188].

5.5.2.5 Backend

The backend is acting as the bridge to connect the user interface and other core components of the NRP. On the user interface end point, it exposes a web server implementing RESTful APIs. It is the first handler for user request, and forwards processed user requests vis ROS on the other end point. The backend in turn provides lists of actions to the user interface which contains simulation listing, simulation handling, simulation creation, experiment listing and manipulation, and backend diagnostic information.

5.5.2.6 Frontend

The Frontend serves as the user interface to the NRP. For the purpose of being accessed and used easily by a broader user base, it is implemented as a web based application using standard web technologies and supporting cross-platform. The Frontend is composed of an Experiment Simulation Viewer and Editors for designing and editing an experiment. The main features of the ESV is that it embeds a high-fidelity 3D view that allows the user to navigate through the virtual environment with the robot model. In order to facilitate users to easily design the experiment from the user interface, the Frontend provide the user with a complete list of editors, *Environment Editor, Brain Editor, Transfer Function Editor and Experiment Workflow Editor*, to configure all the aspects of an experiment.

Chapter 6

Autonomous Locomotion Implementations

In this chapter, the overall control architecture is first presented. In order to demonstrate and examine the effectiveness of our proposed controller, three autonomous locomotion tasks will be performed, namely, target tracking, target tracking with obstacle avoidance, and maze traveling. For each task, details will be given including the reward definition, learning process, and the validation performance.

6.1 Control Architecture Overview

After introducing knowledge about the SNN, CPG, and the slither gait of a snake-like robot. On the basis of them, we now present an overview of our biological-inspired hierarchical controller that integrates with the knowledge in this chapter. This serves as a full view of how those sub-controllers are structured and implemented for performing autonomous locomotion tasks.

6.1.1 Background

With more and more in-depth research findings in neurosicence, the locomotion control system or motor system in a large variety of animals, have been convinced of being organized hierarchically [1, 250, 251, 252], in which behaviors or functions are divided and assigned to different groups and further subdivided into smaller functional units.

As their counterparts in robotics, the behaviors of bionic robots are also controlled with the similar framework. Nagata *et al.* [253] presented a structured hierarchical neural network to control mobile robots with expected behaviors such as capture and escape. Their controller combines a reason network and an instinct network. The robot behaves according to the reason network when it

is given sensory information on which its action should be based, and behaves according to the instinct network when it is not receiving such sensory information, and when it must take a series of actions for a certain period of time. Espinal *et al.* [254] proposed a method to design Spiking Central Pattern Generators (SCPGs) to regulate the walking frequency of a hexapod robot. This work offered an immature to regulate the activities of the CPG controller for changing locomotion, however, failed to integrate the environment by sensing any information. Similar work can also be found in [255, 256, 257], where the CPG is evolved or modulated by sensory information.

6.1.2 Hierarchical Control Architecture

Inspired by the aforementioned studies, we adopt a hierarchical paradigm to control the autonomous locomotion of a snake-like robot, which consist of two parts: the nervous system for making decision and the skeletal system for generating movement (See Figure 6.1). Specifically speaking, the nervous system has two components, namely, the high-level SNN-based controller and the low-level CPG-based controller. The skeletal system adopts a slithering gait to perform autonomous locomotion tasks in the scope of this thesis.

As the high-level controller in the nervous system, spiking neural network is implemented to processes the information from different kinds of sensors and compute the action decisions correspondingly. Different from other ANNs, SNN controller take time-based spikes as input, therefore, the sensor information is first transmitted into a sequence spikes and then fed into the SNN. Meanwhile, it can also take central feedback from CPG controller which affects the local control. The functions inside the high-level controller is structured as a dendrogram, in which the nervous system is treated as the vertice connecting the sensing system and the motor system. Therefore, each sub-SNN controller is responsible for one specific action, such as regulating the body shape, speed, or direction. With this framework, the high-level controller can cope with multiple tasks parallelly as long as there is some mechanism to schedule those functions. The output of the SNN controller is a combination of action commands matching with the lower CPG-based controller, such as speed, turning direction, and other gait parameters. The learning rule for out SNN controller is mainly built on the R-STDP mechanism. In Chapter 5, great details of our proposed SNN controller are given in terms of information processing, modeling, and its learning rule.

Inside the nervous system, CPG is regarded as the most vital component, since it maintains some of the fundamental rhythmic activities controlled by instinct. Similarly in this thesis, the CPG is also adopted as the lower level controller that is responsible for receiving the descending action command and meanwhile sending the joint angles to our snake-like robot. Specifically, the CPG con-

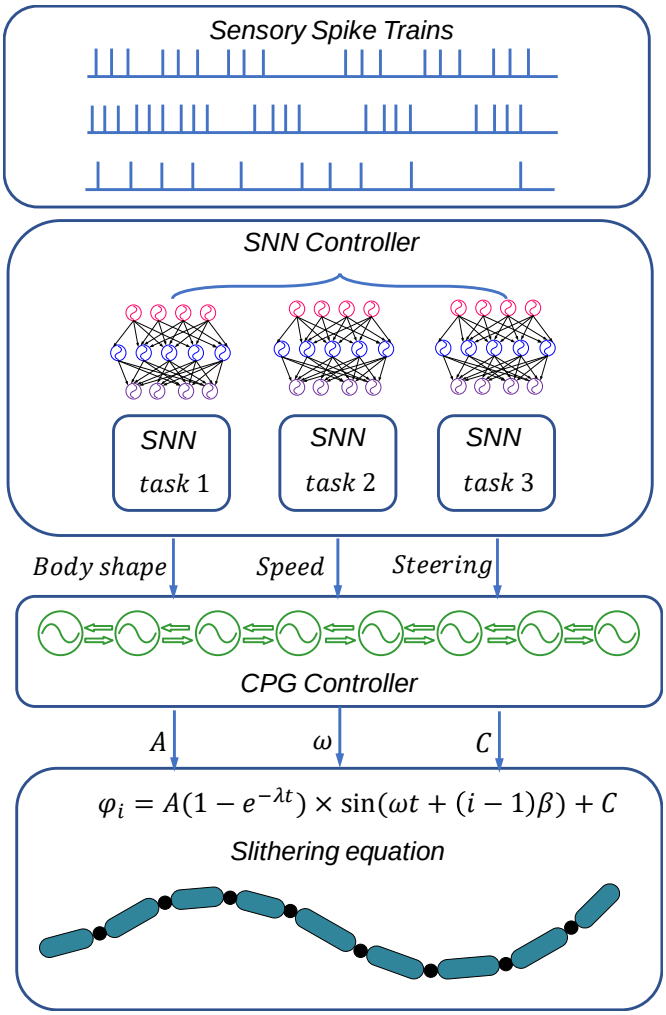


Figure 6.1: The biological-inspired hierarchical control architecture of a snake-like robot for performing autonomous locomotion tasks.

troller will modulate the joint commands as long as there is a gait transition process regarding the the travel speed, direction, or the body curve. Meanwhile, the low-level CPG controller also take charges in reflex actions when there is no descending control signal from higher level controller. This is not only more stable for locomotion control, but also be energy efficient if there is no need to keep executing the heavyweight energy-intensive controllers all the time. The implementation details of our proposed CPG controller are given in Chapter 4.

The slithering gait is regarding as the promising motion morphology for a snake-like robot due to

the fact that its moving direction is almost align with the visual direction. Therefore, it is adopted to perform several kinds of autonomous tasks. Inspired by the *gait equation*, we simplify the gait control by defining parameters that are directly related to the motion state, namely, the speed, turning radius, and body shape. These motion state parameters are controlled by the CPG controller. Under its effect, a smooth gait transition process can be guaranteed when conducting different motions. This gait is further modeled and analyzed in Chapter 3.

Generally speaking, the working flow for our controller is to: 1. Sensing the environment. 2. Making action decision . 3. Generating joint command. 4. Perform action and interacting with the environment. With the proposed hierarchical controller, we will implement three different autonomous locomotion tasks and compare the results with other learning-based ANNs in Chapter 6.

6.2 Snake-like Robot Details

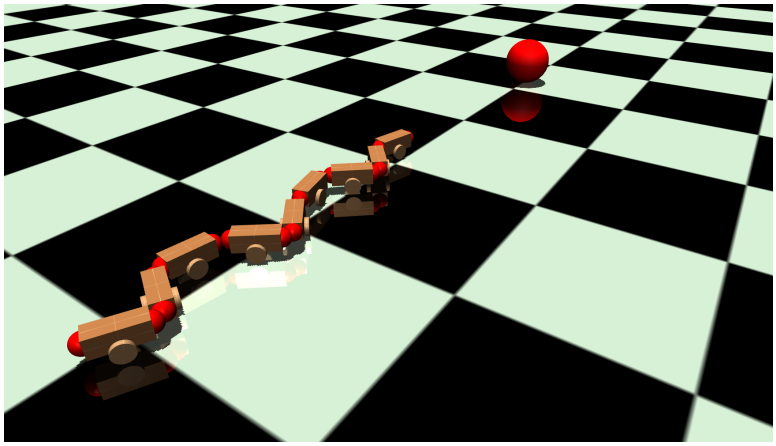


Figure 6.2: An illustration of snake-like robot for target tracking task.

This section mainly describes the components of the snake-like robot model and its equipped sensors. Figure 6.2 first depicts an overview of the snake-like robot performing the target tracking task. The wheeled snake-like robot model consists of eight vertical joints and nine identical modules, including one head module for mounting sensors. This structure was inspired from the ACM series snake-like robot, which also uses the same number of modules. Figure 6.3 shows the technical drawings of the first two modules from the snake-like robot model. According to the requirements of different tasks, selected sensor is deployed at the head module. Take target tracking task as an example, a vision sensor is deployed at the front center of the head module.

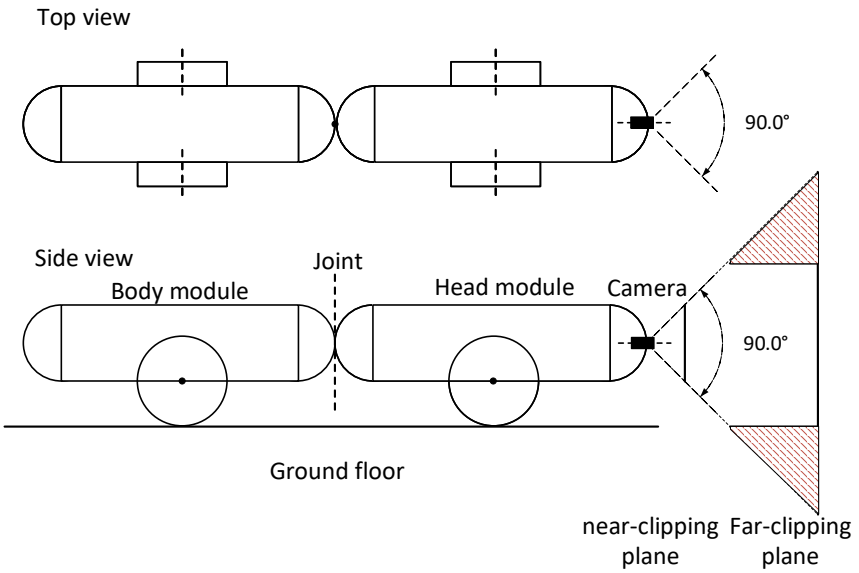


Figure 6.3: The technical details of the wheeled snake-like robot from the top and side view. The perspective angle of the camera is 90° with its range limited by the near-clipping plane and the far-clipping plane. The rasterized area is cropped for the reason of simplicity.

6.3 Target Tracking Task

Since visual motion detection is essential for the survival of many species, we will first implement an autonomous target tracking experiment for our planar snake-like robot. The target is recognized by its red color pixel and its position information is encoded using the binary encoding strategy introduced in section 5.3.

6.3.1 Target Movement

In real life, the target usually moves randomly to the robot agent. Therefore, we have pre-designed the trajectories for training and testing. The training trajectory is calculated as a sinusoid curve as

$$\begin{cases} x_{target} = t + x_0 \\ y_{target} = s \cdot A_{target} \sin(\pi \times t / \tau_{target}) + y_0 \end{cases} \tag{6.1}$$

, where x_{target}, y_{target} are the coordinate position of the target and x_0, y_0 are its initial positions. A_{target} and τ_{target} are constant parameters to modulate the trajectory curve. s is a factor of $+/-1$ that alternates the target's direction every time the simulation resets so that the robot is confronted equally with the target going left or right with respect to its vision sensor. In addition, we also design three more challenging tracks for evaluating the SNN controller as unknown scenes, which are a double-frequency sinusoid curve as the training scene, a zigzag curve, and a ladder curve (See

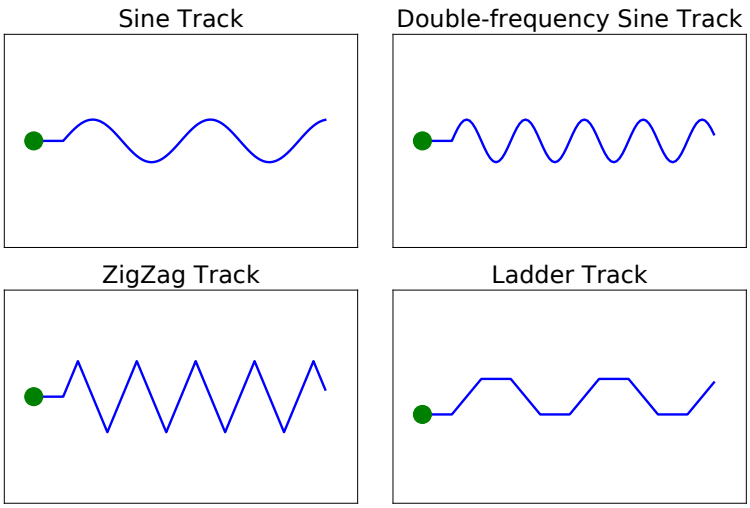


Figure 6.4: The pre-designed target trajectories. The first sine track is used as training task. The other three tracks are used as testing tasks.

Fig. 6.4). In all these tracks, there is a green dot that indicates the starting position and a short straight line for the beginning process.

6.3.2 Information Encoding

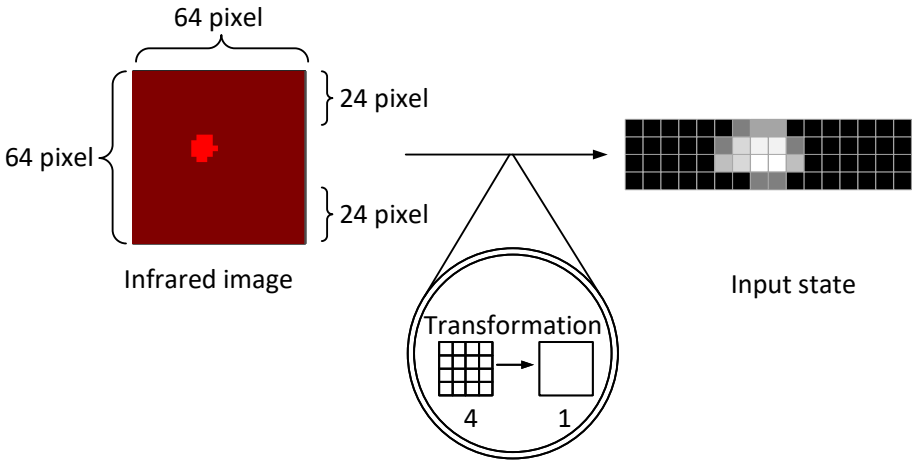


Figure 6.5: State visual input representation for target tracking task.

As shown in Figure 6.3, the RGB vision camera is deployed at the front center of the head module of the snake-like robot. Its perspective projection angle is 90° with a resolution 64 × 64 pixels. For reasons of simplicity, we just cut off the rasterized part of the image since these visual information is

meaningless for representing the target.

The mapping process is visualized in Figure 6.5. The incoming image from the camera is translated into the network's state, which is then given to the input neurons as follow: First, the image resolution of 64 × 64 pixels is reduced to 16 × 64 pixels by cropping the top and bottom by 24 px each. Every pixel has an intensity value in the range of $[0,255]$ according to their infrared radiation. This value per pixel is normalized, so that the sum of the intensities of 16 pixels varies between 0 and 16. Again, the resulting value is normalized and then multiplied by the maximum firing rate of $300Hz$. This value is then assigned to the corresponding spike generator for which it is the mean firing rate. For each outgoing connection the spike generators output unique spike trains adhering to Poisson statistics. Therefore, each spike generator is connected one-to-one to a parrot neuron: one such neuron will replicate the incoming spike train and forward it unchanged to it's entire outbound sub-network.

6.3.3 Reward Definition

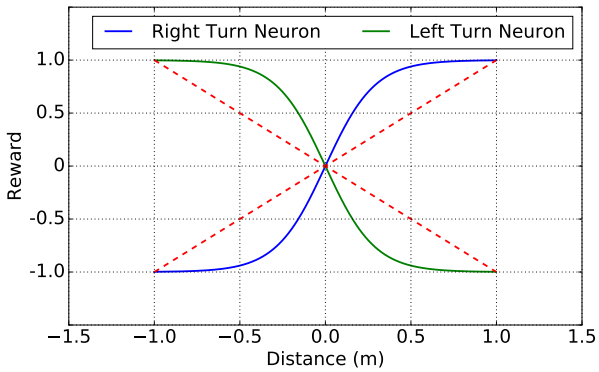


Figure 6.6: The reward definition for target tracking task.

As the synaptic plasticity is modeled as R-STDP, a value for this reward-like variable needs to be calculated every simulation step. This value should somehow reflect the good or bad actions taken by the robot. First, the image from the vision sensor is processed by only extracting its red color. Second, the image moments is calculated to compute the centroid of the target d in the visual field, which ranges from $[-1,1]$ and 0 indicates the middle position in the visual field. Thus, the reward r is defined according to the position d as,

$$d = \frac{2 \cdot M_{10}}{M_{00} \cdot 64} - 1,\ d \in [-1,1]. \tag{6.2}$$

$$r = \frac{2}{1 + e^{-d \cdot \gamma}} - 1, \text{ with } \gamma = 8. \tag{6.3}$$

M_{10} and M_{00} are the first-order and zero-order moment of the image. γ is used to modulate the gradient of the reward curve. From Fig. 6.6, we can see that the reward r is zero when it is located at the center of the visual field. Thus, the reward modulation stops working and the two steering neuron activities are balanced to each other to drive the snake forward. The reward gets steep when it deviates to the boundaries and turns to its maximum or minimum value when it exceeds ±0.6. This can help the robot do its best from loosing the target outside of the visual field.

6.3.4 SNN Architecture

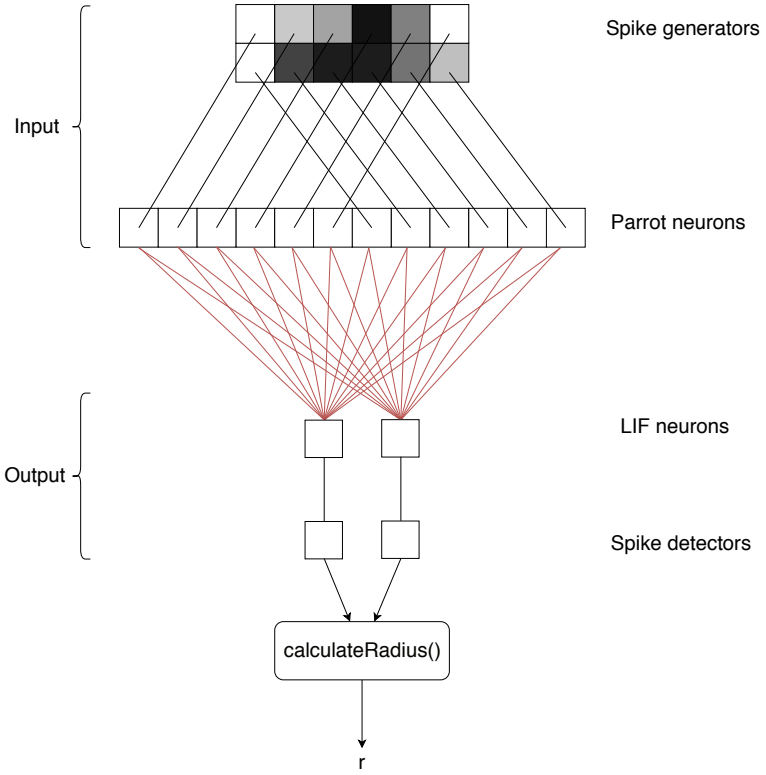


Figure 6.7: A graphical abstraction of the structure of the network without a hidden layer.

Figure 6.7 shows the architecture of the target tracking SNN controller. The vision sensor data is first scaled to stimulate Poisson neurons, yields a single network with $8 \times 2 = 16$ input neurons. The input layer is then processed by the same amount of parrot neurons for insuring the same spike train for the same image input. Afterwards, the input layer is connected to two LIF output neurons in an "all to all" fashion using R-STDP synapses. The output neurons are an implementation of the Leaky-Integrate-and-Fire (LIF) neurons. To register the spikes of these neurons they are connected one-to-one to spike detectors. The detected spikes of these two neurons are then transformed into

a turning radius as explained in (3.16). In order to reduce the complexity involved in the control task, the reward signal in this work was directly set at each simulation time step instead of doing it indirectly by exciting a pool of dopaminergic neurons first.

6.3.5 Training Details

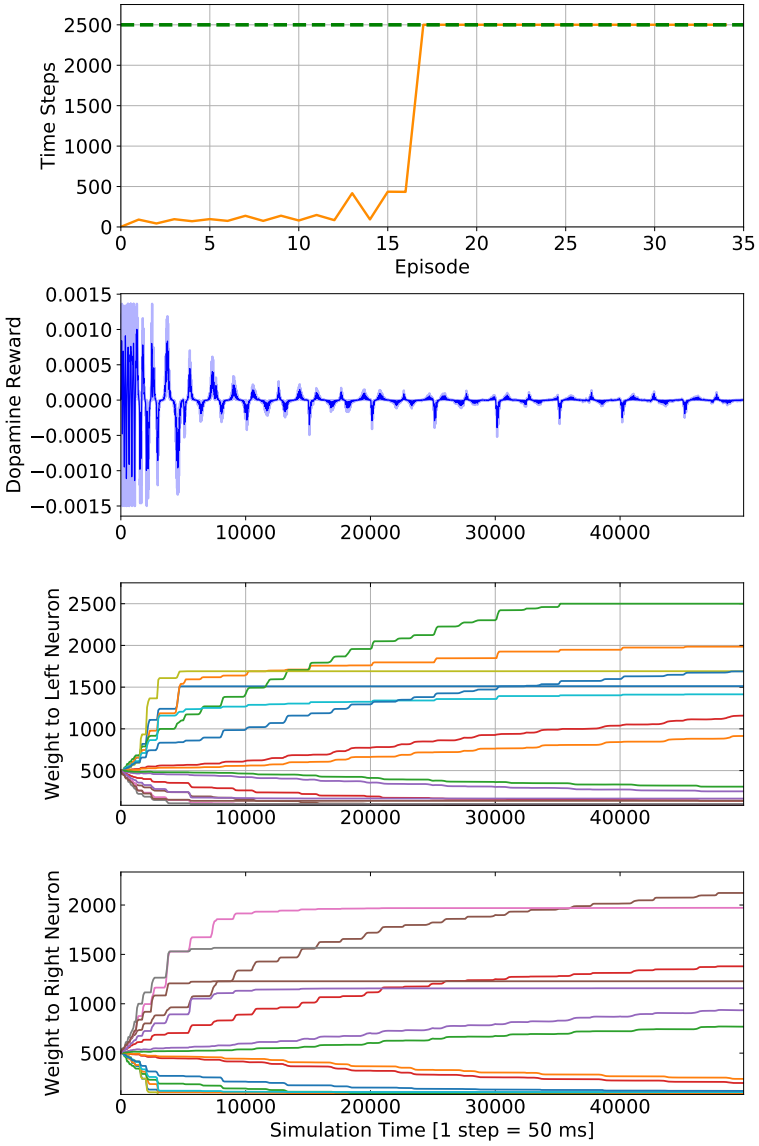


Figure 6.8: Training details of the target tracking controller. The time steps lasting in each episode is shown at the top. The green dash line indicates the maximum time steps 1,500. The reward and the synaptic weights are shown over the number of simulation steps.

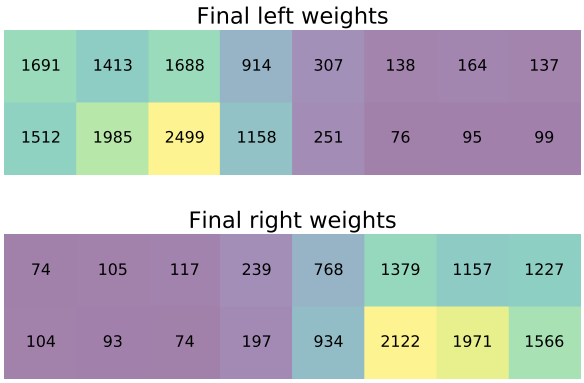


Figure 6.9: Learning progress of the R-STDP controller over every 8,000 steps ($1step = 50ms$). Learned connection weights to the left and right motor neuron of the R-STDP controller are shown in last row after 4,000 simulation steps.

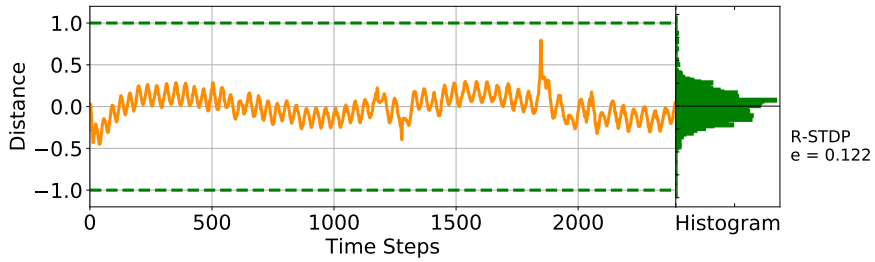


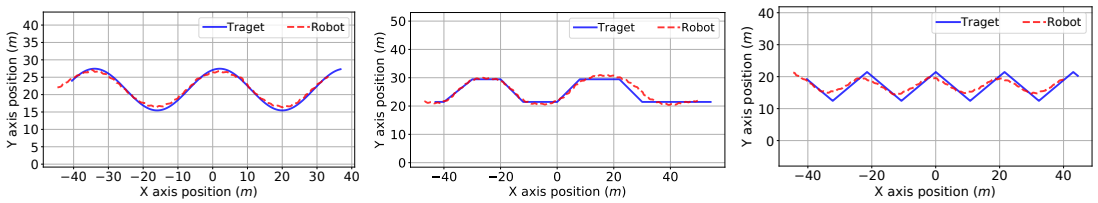
Figure 6.10: Learning progress of the R-STDP controller over every 8,000 steps ($1step = 50ms$). Learned connection weights to the left and right motor neuron of the R-STDP controller are shown in last row after 4,000 simulation steps.

In Figure 6.8, the training progress of the R-STDP controller for autonomous target tracking is shown as well as its reward. Specifically, it shows the time steps in each episode until that the robot loses the target for 20 consecutive time steps or reaches the maximum steps 2500. A simulation step is equivalent to 50 ms both for the simulation of the SNN and the robot simulator itself. Correspondingly, the dopamine reward varies irregularly at the beginning and levels off to zero over time steps. In the beginning of the training procedure, the snake-like robot will just slither forward, since all connection weights for both steering neurons have been set to the same initial value. Therefore, during the first 30 episode, trials are mostly terminated when the target makes its first turn and drifts out of the vision field of the robot. Each time the robot loses its target, it will periodically induce high reward values in the beginning to change the synaptic weights. Consequently, the high reward over a longer period of time causes a significant change in the connection values. Shortly before step 20,000, the snake-like robot has learned to follow the target at its first turn, but it still deviates

from its optimize course. At this point, we can clearly see the time step in each episode is growing steadily. Afterwards, the controller has successfully learned to follow the target regardless how the target turns. Episodes are only terminated once the robot has reached the maximum steps, therefore, the reward changes after step 23,000 are considerably smaller than before and tends to be zero, which is the most ideal situation.

The learned weights after 25,000 simulation steps are shown in Figure 6.9. We can find out that these synapses resemble the spatial pattern of the tracking tasks, in which the SNN tends to drive the target to the visual center once there is a deviation and this function gets stronger when it is closer and closer to the boundaries. This phenomenon is well resembled by the synaptic weights and shown with a color map in the figure.

Finally, we load the well trained synaptic weights to the SNN and execute the controller on the training scenario. The deviation of the target in the visual field is shown in Figure 6.10 and with its error histogram. From the fluctuation of the distance curve, we can see that the head module of the robot still wiggle side to side due to its inherent motion pattern. On the other hand, the curve also resemble the sine-like trajectory of the target. The tracking accuracy is presented by the distance error in the visual field, which is around 0.188 in this case. From that, we can clearly see that the snake-like robot tracks the target well.



(a) Double-frequency sine track. **(b)** Ladder shape track. **(c)** Zigzag track.

Figure 6.11: Three testing tracks. In each task, the target track is presented with blue solid line and the robot trajectory is marked with red dash line.

6.3.6 Performance

To examine the adaptability and practicality of the proposed controller, another three tracking trajectories are used to run the SNN controller (See Fig. 6.4). Meanwhile, these results are shown by plotting the trajectories of the target and robot at the same time. For the first sine-wave trajectory, we have double the frequency as the training scene to enhance the tracking difficulty. From Fig. 6.11(a) we can see that the robot follows the path of the target steadily, even it doubles its frequency as the

training sine track. For the second ladder trajectory in Fig. 6.11(b), it is even more difficult compared with the first one due to those sharp corners. Then we can observe that the robot follows it closely while sometimes cuts across short path to keep going with the target. For the third triangular trajectory in Fig. 6.11(c), the robot has difficulty in following the target precisely due to its ordinary maneuverability. However, it keeps cutting short path when the target makes a sharp turn, yielding a sine curve matching the target triangular path.

6.4 Tracking with Obstacle Avoidance

As shown in Figure 6.1, the high-level SNN control can consist of multiple sub task controllers for dealing with complex tasks. For example, in a target tracking scenario, we inevitably have to get avoid of some obstacles in the target tracking path. Therefore, in this section, we will explore our proposed controller for performing tasks with multiple functions based on the target tracking task by adding obstacle avoidance capability. Besides, the sub controller for obstacle avoidance will take proximity sensors as input and transfer them into event-based signals (See Figure 6.12).

As

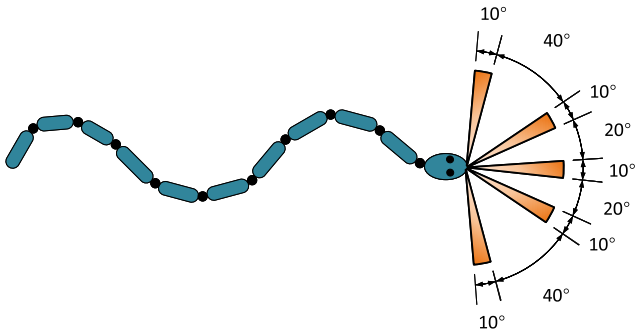
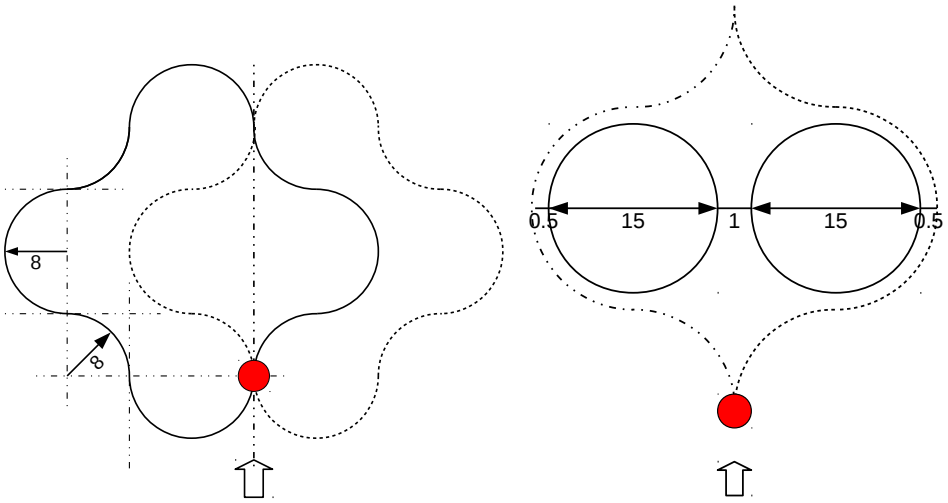


Figure 6.12: The distribution of proximity sensors.

6.4.1 Task Description

When a robot is marching forward, it will always have opportunities to run across some obstacles and hinder its task. Once it encounters such situations, the robot has to avoid those obstacles due to higher priority. Therefore, a controller selection mechanism that decides the proper action has to be investigated.

To be more specific, we first present the task scenario for training and testing as shown in Figure 6.13 In order to train the SNN, two scenarios are used to train the target tracking and obstacle avoidance, respectively. Figure 6.13(a) shows the environment for the target tracking controller. In this scene, there is no obstacle but the target ball and the snake-like robot. As introduced before, the target ball will moving along the predefined path marked in the figure. For this trajectory, the robot will make two left turns and one right turn in a sequence. If trained like this problems arise, the robot makes two left turns for every right turn. This means one side of the SNN will get more positive rewards and the other side more negative rewards. This bias in the training results in a SNN that is biased towards left turns. To prevent this each path is present mirrored a second time. At first



(a) Training environments for target tracking.

(b) Training environments for obstacle avoidance.

Figure 6.13: The training environments for obstacle avoidance tasks.

the robot changed after each episode from one to the other. This again leads to biased training. The robot will learn one path better and episodes in that direction will be much longer. This results in getting even better in this direction while still being bad in the other. If the training rate is to high the robot can even forget how to take one turn. To prevent this the robot keeps track how long it trained on each path and tries to train an equal amount on both.

The obstacle avoidance controller is trained in an environment with two obstacles and paths (See Figure 6.13)(b). One is the mirrored version of the other for the same reason as with the target following controller. The obstacle avoidance SNN needs to learn to avoid collisions with the obstacle while letting the target following SNN successfully track the target. This controller is evaluated in the training environment since different learning rates do not affect the outcome as long as they are reasonable.

6.4.2 Information Encoding

In this task, we have two types of sensors that are being used for acquiring information form the environment: the infrared vision sensor for the moving target and the proximity sensors for obstacle detection. For the vision sensor, it has been introduced in previous section (See Figure 6.5). For the proximity sensors, we deploy four of them equally distributed along the two sides of the head module, as seen in Figure 6.12. The four sideway sensors are using for detecting a possible obstacle

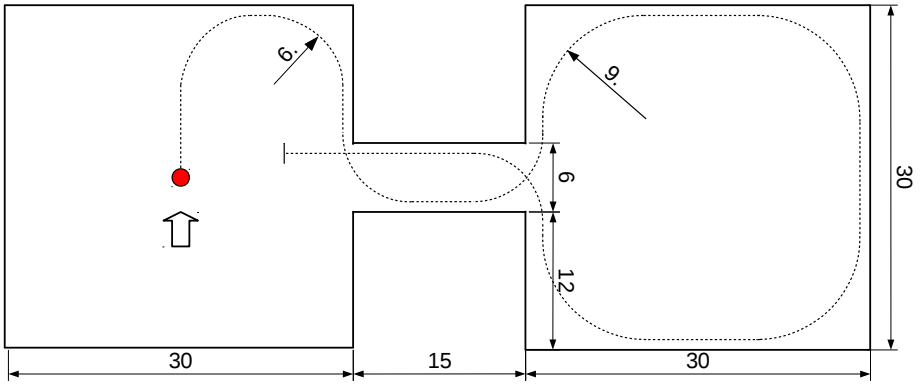


Figure 6.14: The evaluation environment consists of two rooms connected by a corridor. The robot follows the target from one room to the other and back.

or wall, while the middle one is used for detecting the distance between the target and the robot itself in order to track it with speed control.

Algorithm 1 Proximity data preprocessing

```
1: function PREPROCESSING(distances, d_max)
2:     for i ∈ [distances.size] do
3:         if distances[i] < 0 or distances[i] > d_max then
4:             distances[i] = d_max
5:         end if
6:         distances[i] = 1 − distance[i] / d_max
7:     end for
8:     return distances
9: end function
```

After obtaining the sensor information, we have to transfer them into spikes.

6.4.3 Reward Definition

Two rewards will be given to the SNN controller since we need to achieve target tracking and obstacle avoidance functions at the same time.

6.4.3.1 Reward for Obstacle Avoidance

The rewards for the obstacle avoidance SNN are given event based. This means both output neurons will receive no reward, if the episode is not terminated or there is no obstacle detected So rewards are only given at the end of each episode and only to the neuron that received a input, which happens if there is a obstacle detected on the opposing side. Then, a positive reward of 1 is given for a collision. Since each neuron is only connected to the proximity sensors on the opposing side of the snake,

this leads to obstacle avoidance. For example, if the snake collides with a obstacle on the left side, the left proximity sensors will have a big input the positive reward strengthens the connection. And thus the right neuron will fire more frequently next time. This leads to effective obstacle avoidance. In fact it will lead to a obstacle avoidance that will learn to turn away as much as possible from any obstacle. This would cause the snake from loosing sight of the target and must therefore be prevented by the second case. When no collision happens but the robot looses sight while avoiding an obstacle the neuron that was evading the obstacle receives a reward of -1. This negative reward causes the network to fire less next time. The network will learn to not just turn away from obstacles, but to turn away form obstacles as little as possible to avoid collisions while tracking the target.

Algorithm 2 Reward for obstacle avoidance SNN

```
1: function GET_REWARD(terminate, collision, obstacle_detected)
2:     if terminate and obstacle_detected then
3:         if collision then
4:             return 1
5:         else
6:             return −1
7:         end if
8:     end if
9:     return 0
10: end function
```

6.4.4 SNN Architecture

For this multi-function task, the SNN controller consist of two sub SNN, namely one for the target tracking and one for the obstacle avoidance. These two SNNs are organized based on our proposed controller architecture in Figure 6.1. Each of these two SNNs are discussed as follows.

6.4.4.1 Target Following Controller

The target following controller receives as input a 16×4 pixel infrared image and returns the spike trains of the two output neurons. It has a simple feed forward architecture consisting of a input layer that is directly connected to the output layer in an all to all fashion. The input layer consist of 64 Poisson generators which transform the preprocessed input values to spike trains. The output layer consists of the left and right neuron. The connections are separated into two groups depending on the output neuron at which they end.

6.4.4.2 Obstacle avoidance controller

The goal of the obstacle avoidance controller is to keep a certain distance to obstacles. This behaviour is achieved by connecting the left neuron to the proximity sensors that detect the right side for obstacles. When they detect an obstacle the left neuron can fire and move the robot away from the obstacle. The sensors on the left side are connected to the right neuron for the same reason. This connection pattern corresponds to the coward behaviour of Braitenberg vehicles[258]. Yet the goal is not to train this controller to fear obstacles, but to keep a distance that prevents collisions, whenever possible without loosing track of the target.

Algorithm 3 Controller selection

```
1: function CHOOSE_ANGLE(α_tf, β_oa)
2:     if obstacles are detected then
3:         if |α_tf| > |β_oa| and sgn(α_tf) = sgn(β_oa) then
4:             return α_tf
5:         else
6:             return β_oa
7:         end if
8:     else
9:         return α_tf
10:    end if
11: end function
```

6.4.4.3 Controller Selection

Neither of the two controller is able to solve the proposed problem alone, but by working together they can archive target tracking while avoiding obstacles on the sides. The pseudocode 3.2 details which output angle is given to the snake car robot. The main goal is to track the target, so in general the target following controller has the command and his output angle is used to follow the target. Once the proximity sensors sense a obstacle that is close to the snake the obstacle avoidance controller becomes active and moves the snake away form the obstacle. There is one important edge case. If the obstacle avoidance controller becomes active and suggests a right turn to prevent a collision with a obstacle to the left, the target could be already be moving farther to the right. Then the target following controller also wants to move to the right to follow the target. This already prevents a collision, so following is the better choice. The choose angle function selects between the output of the target following angle α_{tf} and the obstacle avoidance angle α_{oa} . First a decision is made based on the presence or absence of obstacles. Then if some are present the edge case is checked. This happens by comparing the absolute value and the signs of the angle. If the signs are equal the one

with the bigger absolute value is chosen If they have different signs then obstacle avoidance angle is returned.

6.4.5 Training Details

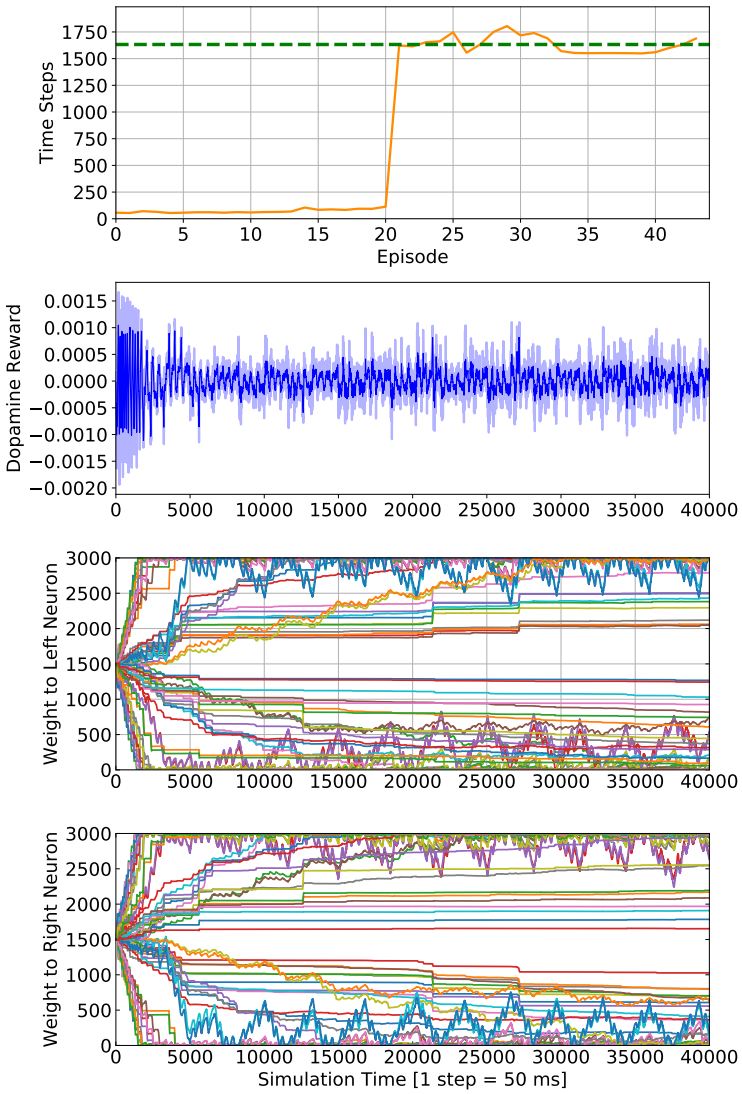


Figure 6.15: The training details of the target following controller.

Figure 6.15 shows the training progress of the target following SNN. The network is trained for 40000 steps. The first graph shows the duration of the 43 episodes completed during training. In the first 21 episodes vision of the target is lost in under 250 steps. The robot fails to track the target in the first 12s of an episode. This part of the training is only 5% or 2000 steps of the whole training process,

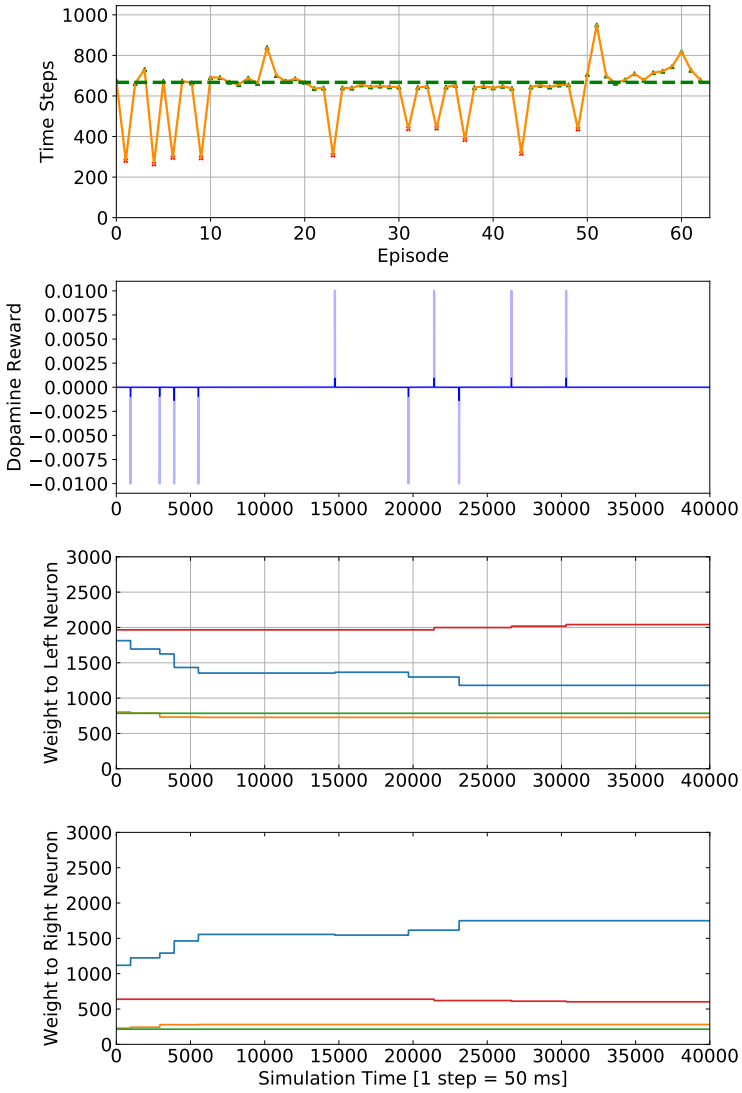


Figure 6.16: The training details of the obstacle avoidance controller.

but the weights change the most in this phase. Then weights hit the maximum and minimum weight values and then the snake manages to complete a episode successfully. This transition can be best seen in the second graph. Here the averaged dopamine reward at each time step is shown. The graph starts with fast changing high rewards until the robot manages to complete the path for the first time. After this the rewards fall into a regular oscillation, that is composed from the shape of the track and the movement of the robot head module. The robot completes all other 22 episodes without loosing track of the target. This part are the other 95% of the training time, around 3800 steps. Each episode lasts for 1700 steps, or $1.5min$, the time needed for the target to complete the training path.

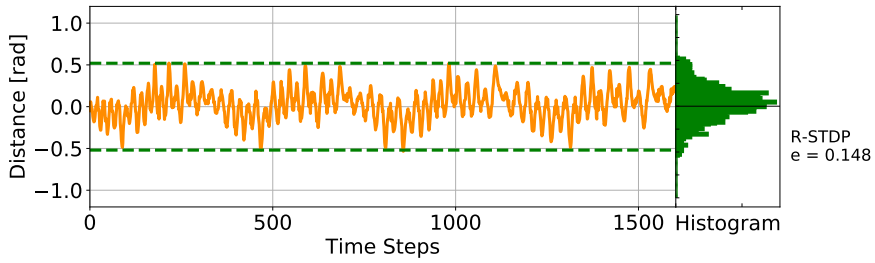


Figure 6.17: The performance in one episode of the target following controller in an unknown environment.

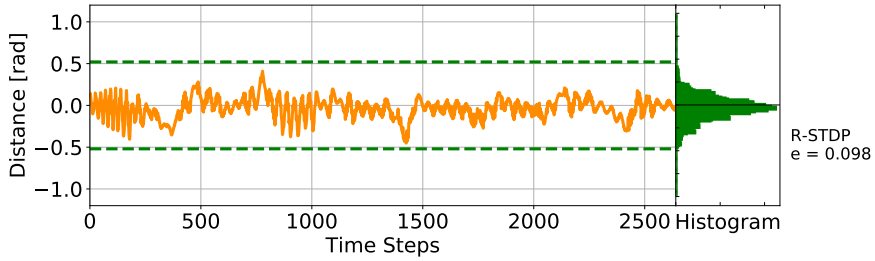


Figure 6.18: Performance of the controllers in the evaluation environment.

Figure 6.16 shows the training progress of the obstacle avoidance SNN. Again the first graph shows the steps in each episode. At step 200 the target arrives at the obstacle and after step 500 the target passed the obstacle. Figure 6.13 shows the training path with obstacle. The initial weights are assigned at random. The second graph shows the rewards at each time step. Rewards are given only at the end of each episode.

6.4.6 Performances

Like examining the target tracking task using unknown scenarios, we also build up testing tasks for the obstacle avoidance controller. Figure 6.18 shows the performance of the target following controller in one episode. The controller is evaluated in an unknown environment. The graph shows the angle between the direction the robot looks and the direction from the robot to the target, at each time step for one episode. On the right a histogram of the errors is given. The average error $e = 0.148$ is equal to an average angle error of $e = 8,4798°$. To evaluate the collaboration of the controllers the unknown environment from Figure 6.14 is used. One episode is shown in Figure 6.18.

6.5 Wall Following Task

Wall following, as one of the robot self navigation abilities, is essential for some of rescue or military tasks. In this section, we will control the snake-like robot to conduct a wall-following task, which uses a DVS to sense the environment. Furthermore, this task also serves as an example for taking neuromorphic sensors as input for our controller.

6.5.1 Task Description

The wall following scenario is an eight-shaped maze consisting of an inner wall and a outer wall. The distance between them is $5m$ (See Figure 6.19). Each straight segment is $10m$ and the angle between two segments is set as $45°$. It is designed with the idea in mind that the snake encounters as many right as left turns if it starts alternately in both directions. Then it will experience all the situations equally. For a whole round in the positive direction, the snake first needs to turn left, then six times right, two times left, six times right and finally left again. If the snake stays perfectly in the middle between the two walls for a whole round, the length of the covered track would amount to $160m$. The other way round is exactly the opposite.

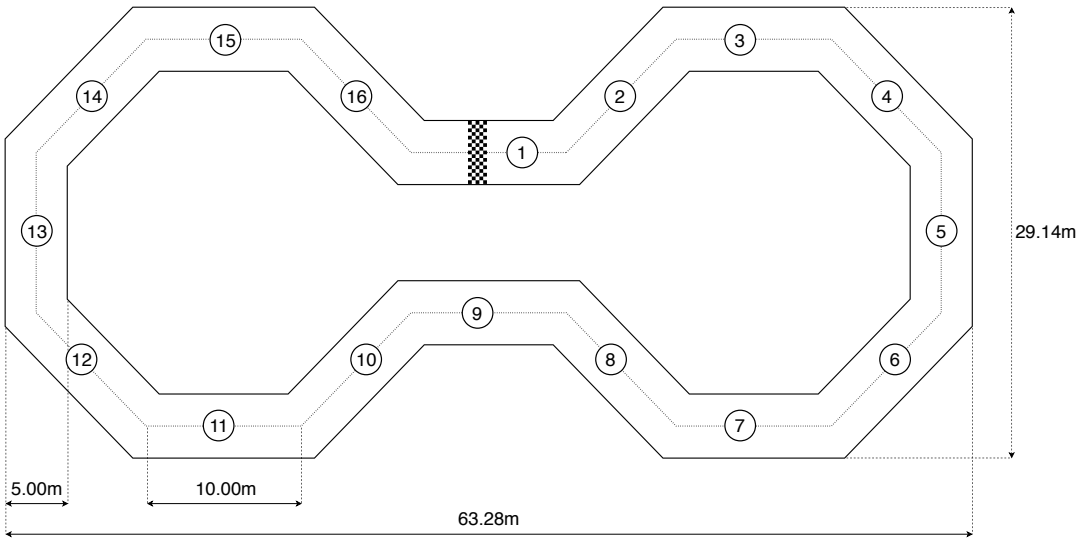


Figure 6.19: Dimensions of the eight-shaped maze. The sections are numbered increasingly for the positive travel direction. The snake finishes the maze successfully if it crosses the finish line.

6.5.2 Information Encoding

Similar to the setup in Figure 6.3, a DVS replaces the infrared camera and will be located at the same position. Different from the conventional vision sensors, the DVS generates sparse, event-based output that represents the positive and negative relative luminance change of a scene (See Figure 6.21). By letting the pixels operate asynchronously sending out events nearly instantaneously on an address-event bus, no complete image frames exist in the system. The output consists of a stream of precisely-timed, individual events representing local changes in temporal contrast.

First, since we mainly focus on the events generated by the walls, the bottom of the DVS image is cropped and results the size changed from the original size of 128×128 to 128×80. Then, to increase the events in one time step and reduce the computing burden, the resolution is decreased to 8×5. Although the DVS image frames will be cropped and reduced to a lower resolution as well as the infrared camera, there is still one more step to process them. Due to the event-based nature of the DVS data, events coming from each simulation step do not always contain sufficient information for the network to make meaningful decisions. Therefore, the state input was computed by condensing information of ten consecutive DVS frames into a single image.

6.5.3 Reward Definition

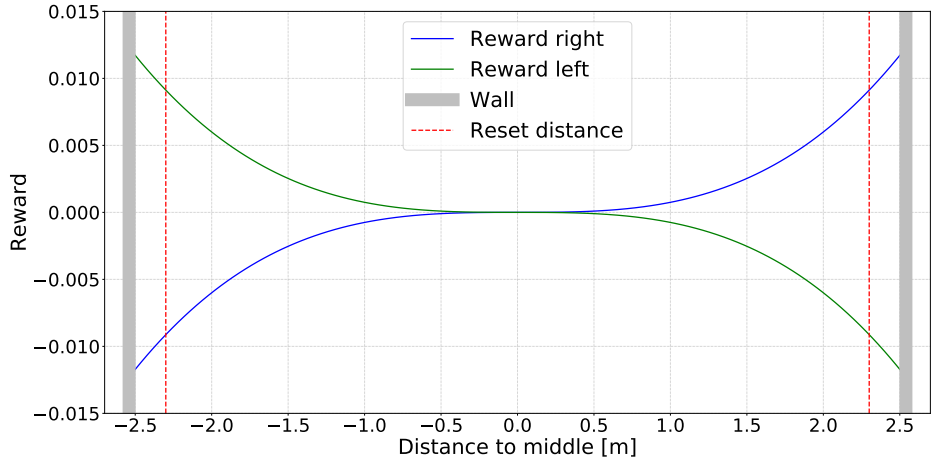


Figure 6.20: Reward given by the R-STDP controller with $c_r = 0.00025$, which is the value used in the simulation.

Similarly to the target tracking task, the reward modulates the neurotransmitter concentration of the R-STDP synapses and is defined with the following considerations: If the snake is close to the left part of the wall, it needs to turn right to stay in the middle position. Therefore, the left output

neuron needs to fire more spikes than the right neuron. Thus, the neurotransmitter concentration of the right neuron should be high and the concentration for the left low. Additionally, to compensate for the sideways motion of the robot there should be a corridor around the middle that counts as the optimal position of the snake. These considerations lead to the reward signal being defined as cubic function dependent on the distance to the center position between the two walls:

$$r_{left/right} = -/+ \left(d^3 \cdot c_r\right), \tag{6.4}$$

where d is the distance to the center position and c_r is a scaling constant. The reward is depicted in Figure 6.20.

6.5.4 SNN Architecture

Figure 6.21 shows the architecture of the wall following SNN controller. In order to feed them effectively into an SNN they have to be "spread" again in time across a simulation time step. Therefore, instead of generating spikes directly from DVS events, the pixel values of the 4×4 images are used to excite Poisson neurons to fire with the respective rate during a simulation time step. Similar to the SNN architecture for target tracking, the input layer is then connected to two steering motor neurons with R-STDP synapse. Under the effect of the dopamine modulation defined in (6.4), all the synapse will be updated according to the performance of the robot. According to different DVS input pattern, the SNN will compute a desired turning radius as explained in (3.16).

6.5.5 Training Details

First, the training progress of the SNN controller for wall following is shown in Figure 6.22. To be specific, the robot will be terminated and reseted once it hits the wall in either side, which is depicted by the deviation distance of 0.25 m to the middle line. Then the robot will take the other direction. At beginning, the robot will go straight forward, because all connection weights for both motor neurons have been set to the same initial value 200. Therefore, during the first 1,000 steps, trials are mostly terminated at the first turn in both directions when the robot misses the turn and the lane-center distance exceeds 0.25 m. Each time the robot fails a turn, it will periodically induce high reward values in the beginning of changing the synaptic weights. As a consequence, the reward over a period of time causes a significant change in the synapse values. After step 500, the robot has learned to take the first turn, but it still deviates from the optimal lane-center position. Finally, the controller has learned to manage the full lap both left and right without causing a reset. Following both lanes close to the optimal lane-center position means low reward values as well. Therefore, the weight changes after step 10000 are considerably smaller than before.

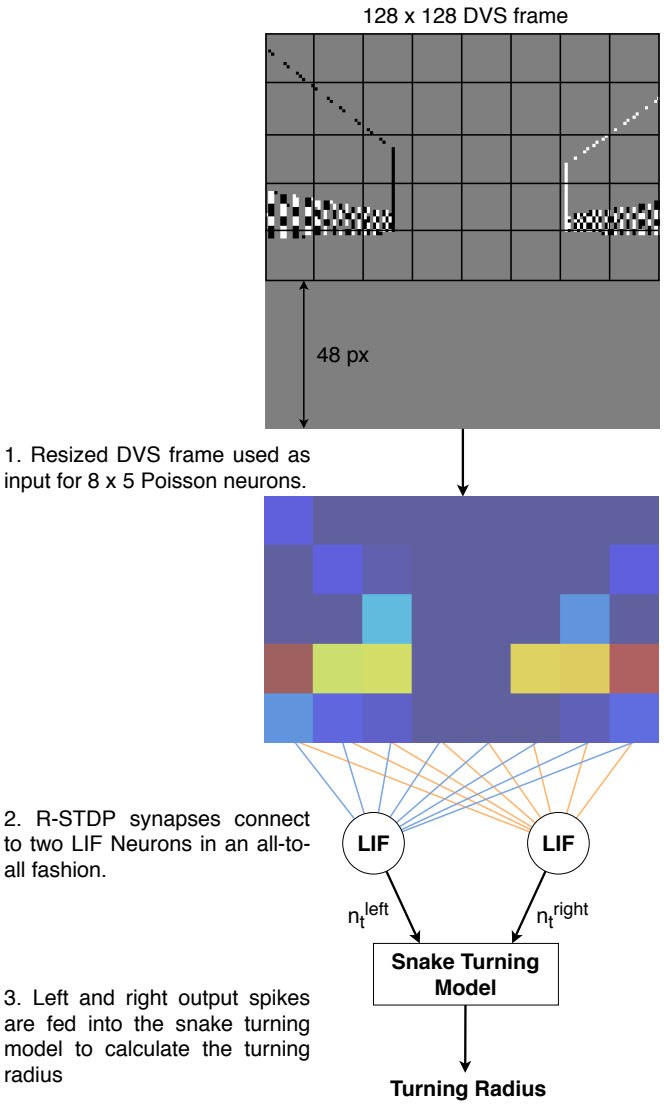


Figure 6.21: A graphical abstraction of the structure of the network for the wall following task.

The final synaptic weights are depicted in Figure 6.22. One can clearly see that the middle left and middle right corner have the greatest influence on the left and right output motor neurons, respectively . For example, if the snake is too close to the right wall, it needs to turn left to return to the middle position. Turning left is achieved if the right motor neurons fires more spikes than the left motor neuron, explaining why the right half of the synaptic weights connecting to the left motor neuron are high excitatory and their counterparts connecting to the left motor neuron are low inhibitory. Another remarkable property of the synaptic weights distribution is that the right

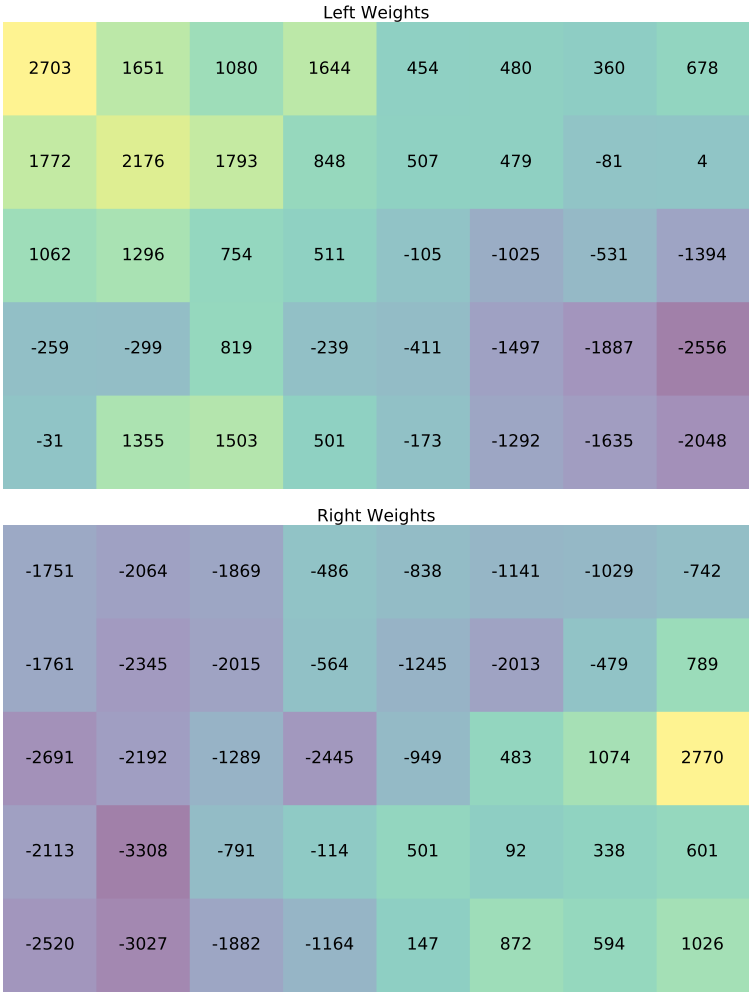


Figure 6.22: The final weights of the wall following controller.

weights have a greater influence over the controller than the left ones. A reason for this might be that the snake traveled more steps in the negative direction than in the positive resulting in more encountered right than left turns. It is also visible that the left and right weights complement each other. A synapse with a high value from the left weights results in a synapse with a high value from the right weights and the other way round. The strongest changes in synaptic weights happen in the beginning of the training, which can be seen in Figure 6.23. First, the right motor neurons are changed excitatorely because the snake encounters a left turn in the first episode resulting in a lot of DVS events on the right half. After the first episode reset, the snake travels in the opposite direction and encounters a right turn, explaining the rise in weights of the left motor neuron and the drop of the weights of the right motor neurons. After these initial changes, the weights only change slightly

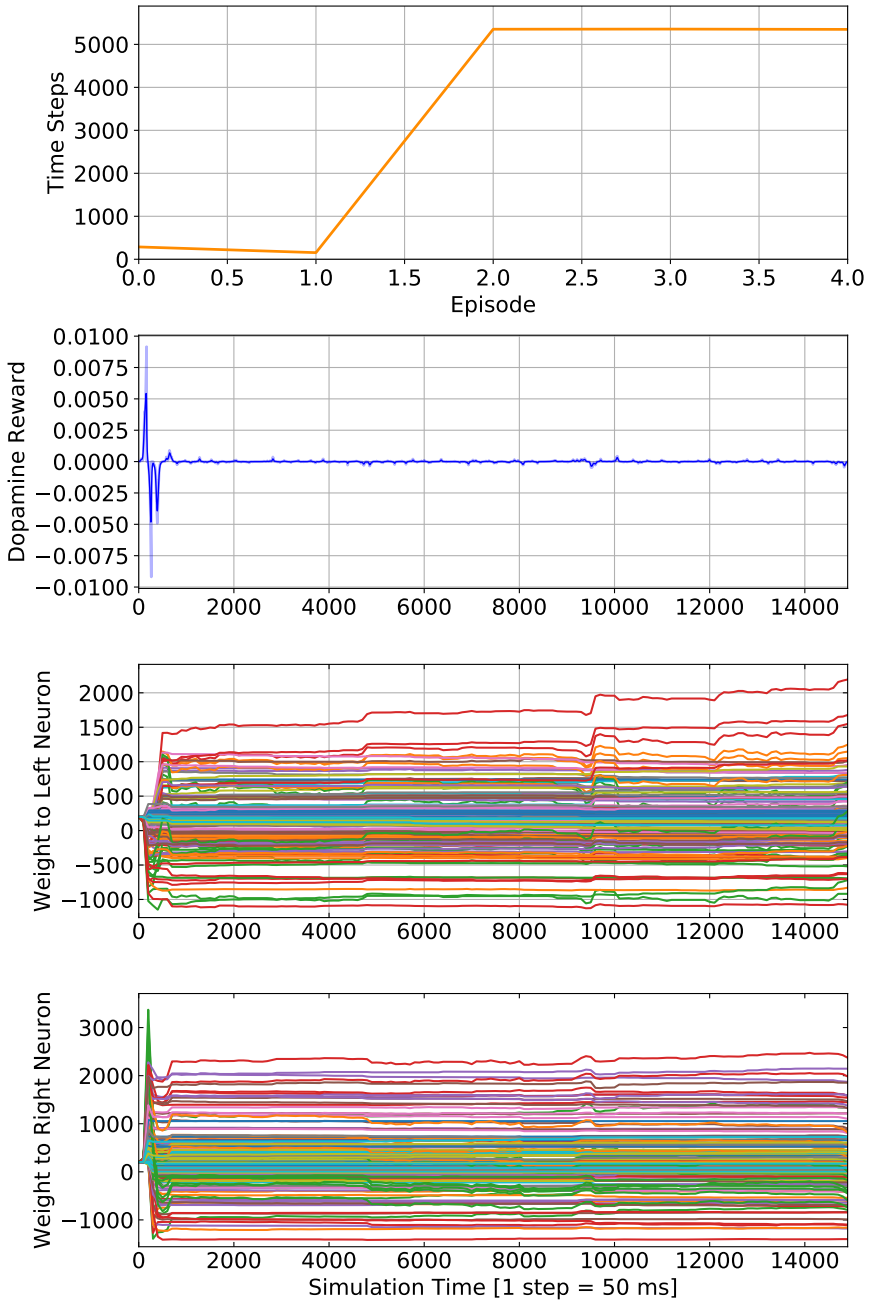


Figure 6.23: Weights over steps plotted for the controller. The vertical lines mark new episodes.

as the snake gets less and less reward by staying close to middle and finishing each episode reaching the starting position again.

6.5.6 Performances

In order to further examine the performance of the wall following controller. It is evaluated on six scenarios with heights varying from $0.5m$ to $3m$ as these wall heights could be encountered in a real world deployment of the robot. The terminology is as follows: The six scenarios are referred to as *scenario_0_5*, *scenario_1*, etc. whereby the number in the end depicts the wall height.

The controller's performance is evaluated based on the following values that are only calculated for scenarios that the controller is able to cope with. A controller can cope with a scenario if it manages to finish one round of the maze.

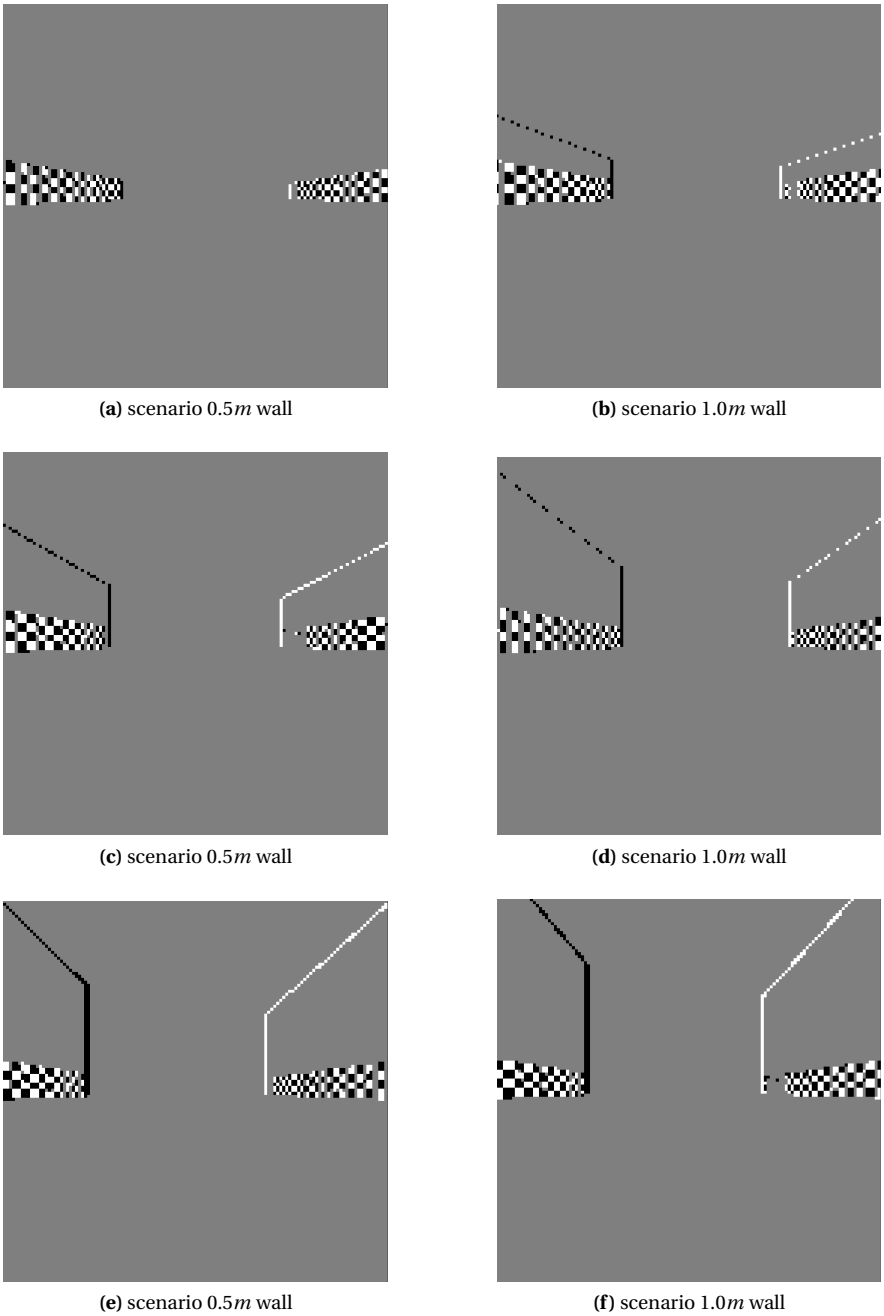
First, the mean of the distance to the middle position between the two walls over one episode is determined. Second, an error value is used to asses how well a controller performs on a scenario given that it is able to finish one round. The error is defined as the mean of the absolute values of the distance to the middle position. And third, the length of the covered track is used as an additional assessment of how well a controller performs that can finish one round.

6.5.6.1 Different Wall Heights

As a first evaluation of the controller's ability to cope with new situations, the snake is tested on six eight-shaped maze scenarios with wall heights that vary from $0.5m$ to $3m$. These heights are chosen for evaluation as they could be encountered in a real-world deployment of the robot, for example, in a collapsed factory building or something similar. The six scenarios are referred to as *scenario_0_5*, *scenario_1*, etc. whereby the number in the end depicts the wall height. Figure 6.24 shows DVS frames from the six different scenarios.

As Figure 6.25 shows, the controller is able to finish a full round of the eight-shaped maze in each scenario. One can see the course over the maze the snake takes in the distance over steps plot of *scenario_0_5*, for example. Whereas during left turns, which take place as the first, eighth, ninth, and last turn the snake moves closer to the outer wall resulting in a negative distance value, during right turns the snake moves closer to the inner wall resulting in a positive distance. The snakes position's in the maze are also depicted in Figure 6.26 where the paths of the snake from the different scenarios are laid on top of each other.

For a better comparison, the mean value, the error value, and length of the traveled distance are displayed in Table 6.1. The test on *scenario_2* can be seen as the comparative basis as the snake was trained on this scenario and thus reaches the second best value for the error with $0.263m$ and the best value for the mean with $0.005m$. The values for *scenario_1_5* have the same scale with a mean



(a) scenario 0.5m wall

(b) scenario 1.0m wall

(c) scenario 0.5m wall

(d) scenario 1.0m wall

(e) scenario 0.5m wall

(f) scenario 1.0m wall

Figure 6.24: Raw DVS frames from the scenarios with different wall height.

value of $-0.008m$ and a slightly better error value of $0.253m$. The length of the traveled distance is about one meter longer for *scenario_1_5* with $170.56m$ compared to $169.49m$.

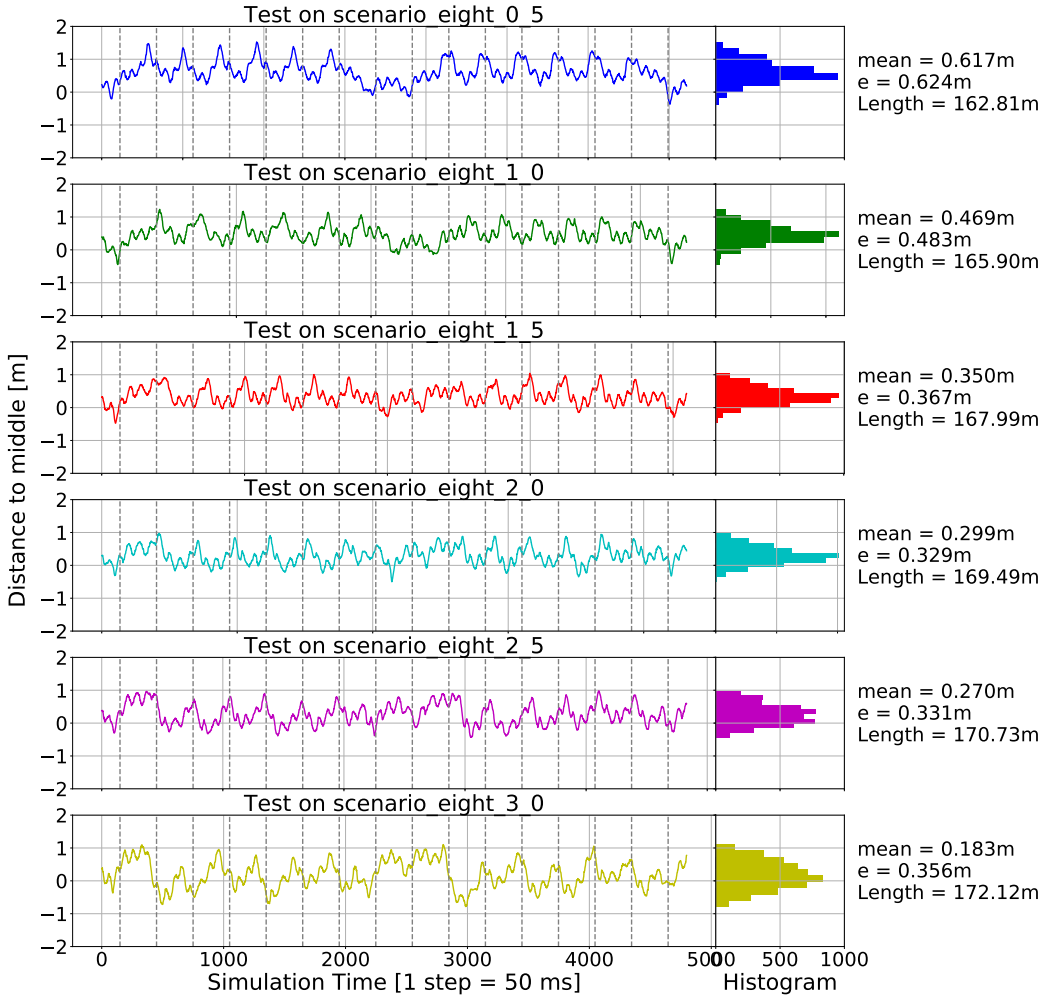


Figure 6.25: Plot of the distance to the middle position over the traveled distance for the scenarios with different wall heights. The dashed vertical lines mark the 16 turns in the maze whereas the first one is a left turn, followed by six right, two left, six right, and a final left turn until the snake is at the starting position again.

For the other scenarios, the following trends are recognizable: The more wall height gets removed, the higher the mean value becomes. In numbers this means $0.217m$ for *scenario_1*, and $0.258m$ for *scenario_0_5*. These numbers show that for these two scenarios, the snake moves closer to the inner wall leading to smaller traveled distances of $163.42m$ and $163.75m$, respectively. The reverse is observable for the higher wall heights with a mean value of $-0.019m$ for *scenario_2_5* and $-0.080m$ for *scenario_3* resulting in higher values for traveled distances of $170.55m$ and $172.21m$, respectively. This behaviour can also be seen in Figure 6.26 where the blue path results from the snake in *scenario_0_5* moving close to the inner wall and thus needing less distance to finish the maze, and

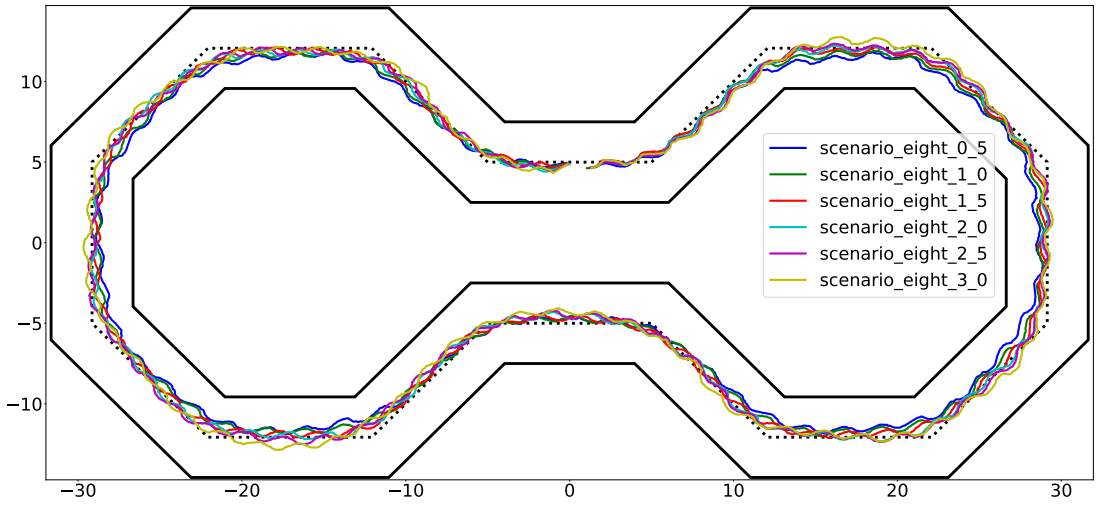


Figure 6.26: The paths of the snake in the different wall heights scenarios laid on top of each other.

the yellow path resulting from the snake in *scenario_3* resulting in more distance.

For the error value, the following trend is recognizable: The more wall height gets removed (in case of *scenario_1* and *scenario_0_5*) or added *scenario_2_5* and *scenario_3*), the greater the error becomes having its highest value for *scenario_0_5* with $0.414m$ and its second highest for *scenario_3* with $0.359m$. As Figure 6.24 shows, these two scenarios differ the most from *scenario_2* leading to DVS events in the top of the frame in case of *scenario_3* or missing events in the middle area of the frame in case of *scenario_0_5*.

Table 6.1: Comparison of the mean value, error value and length of the traveled distance for the eight-shaped maze scenarios with wall heights ranging from $0.5m$ to $3m$.

	Wall Height [m]					
	0.5	1.0	1.5	2.0	2.5	3.0
Mean [m]	0.258	0.217	−0.008	0.005	−0.019	−0.080
Error [m]	0.414	0.280	0.253	0.263	0.291	0.359
Length [m]	163.75	163.42	170.56	169.48	170.55	172.21

Chapter 7

Prototype Validation for CPG

Apart from the wheeled snake like robot in Chapter 6, a modular snake-like robot exhibits better locomotion ability in three-dimensional terrain. In this chapter, we will introduce our snake-like robot designed for field locomotion with diverse indoor and outdoor gaits. Besides, the function that CPG can smooth the gait transition process are demonstrated with our modular snake-like robot.

7.1 Mechanical Design

Our snake-like robot has a modular design, consisting 13 actuated modules and a head module, as seen in Figure 7.1. Those modules communicate with each other via I^2C bus. All the output shafts are alternately aligned with the robot's lateral and dorsal planes to generate 3D locomotion. Each module is connected to the adjacent modules and allows a full 180° rotation.

Figure 7.1: The snake-like robot is slithering forward. This snake-like robot is modular designed. Each joint axis is orthogonal to its neighbors with a rotation range of ±90° .

Each module contains a servo and a set of gears to actuate the joint. The DC servo (DS1509MG)

has a maximum torque of $12.8Kg \cdot cm$ and drives a gearbox with a reduction factor of 3.71. For the electronic hardware part, each module has an Arduino Nano board, a printed circuit board and an angle sensor. The Arduino Nano board runs three tasks: control- ling the servo, reading the joint angle, and communicating with the other modules. Table 7.1 summarizes the technical specifications of our snake robot.

Table 7.1: Overview of Snake-like robot specifications

Items	Discriptions
Dimensions	Diameter 60mm Length 70cm
Mass	Module 0.3kg Full 2kg
Actuation	Max Torque 12.8kg.cm Max Speed 0.07sec/60°
Power	7.4V DC
Communication	I^2C Bus
Sensing	Angular Sensor MLX90316KDC

7.2 Gaits

For our modular snake-like robots, there are several typical indoor and outdoor gaits, such as rolling, sidewinding, and slithering. By setting different parameters, these gaits can be obtained from the extended *gait equation*, which is described as,

$$\begin{cases} \phi(n,t)_{odd} = C_{odd} + P \cdot A_{odd} \cdot \sin(\Omega_{odd} \cdot n + \omega_{odd} \cdot t) \\ \phi(n,t)_{even} = C_{even} + P \cdot A_{even} \cdot \sin(\Omega_{even} \cdot n + \omega_{even} \cdot t + \delta) \\ \Omega_{odd/even} = x_{odd/even} \times \frac{2\pi}{N} \\ P = (\frac{n}{N} \cdot z + y) \in [0,1], \quad \forall n \in [0,N] \end{cases} \tag{7.1}$$

ϕ_{odd} and ϕ_{even} present the angle value for the odd or even joint at time t, where n is the joint index.

For both the rolling and sidewinding gait, the amplitude bias $C_{odd/even}$ is set as 0, the linear reduction P is also cut off by setting $z = 0$ and $y = 1$, and the spatial frequency $\Omega_{odd/even}$ is 0 as well.

Two aspects are demonstrated in simulations, compared with sinusoid-based control method. First, the CPG-based control method can ensure smooth gait trajectory, when the body shape and locomotion speed are changed. Second, the CPG-based control method can effectively decrease the abnormal torque during the transition.

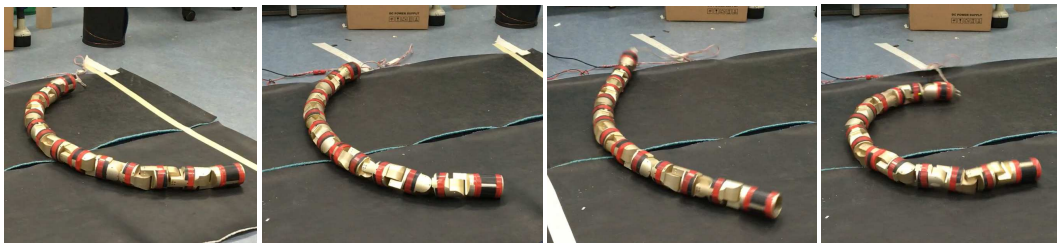


Figure 7.2: Screenshots of the amplitude changes from 20° to 40° of rolling gait. From left to right, the figures show the 20° rolling, the state before transition, the jerky movement during transition, and the 40° rolling after transition, respectively. Due to space limit, other frequency changing and sidewinding gait experiments are shown in the attached video.

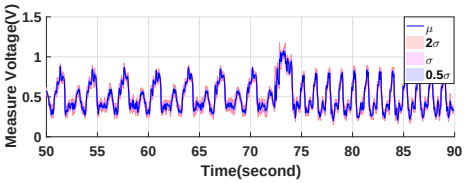
7.3 CPG Validation

In Chapter 4, the characters of the proposed CPG model has been provided via numerical simulation and online execution experiments. Meanwhile, simulations are performed to prove that CPG-based method can achieve smooth transition for a snake-like robot, when the body shape or the locomotion speed is changed. Now, we report the results of a set of experiments conducted on our modular snake-like robot (See Figure 7.1) to demonstrate that our CPG-based method can ensure smooth transition of the body shape and locomotion speed.

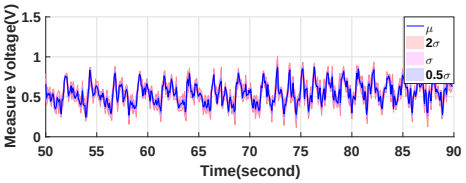
The CPG network is implemented on a Master Arduino Nano board, which is located in the tail module and spreads the commands to each module via I^2C bus. Meanwhile, it will receive the feed-

Table 7.2: Descriptions of the extended gait equations

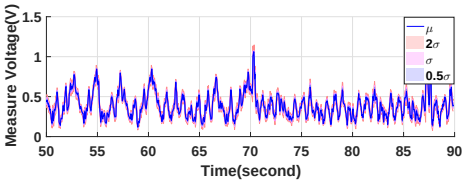
Items	Descriptions
ϕ_{odd}, ϕ_{even}	Joints commands in lateral and dorsal planes
C_{odd}, C_{even}	Body shape offset in lateral and dorsal planes
A_{odd}, A_{even}	Amplitude in lateral and dorsal planes
$\Omega_{odd}, \Omega_{even}$	Spatial frequency in lateral and dorsal planes
$\omega_{odd}, \omega_{even}$	Time frequency in lateral and dorsal planes
N	Module numbers
P	Linear dependency
x	Cycle numbers
n	Module subscript
δ	Phase difference
y, z	Linear coefficient



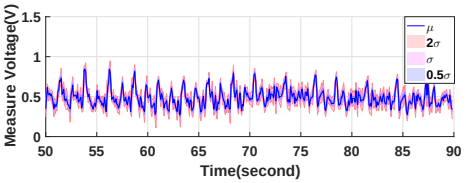
(a) Sin-based method for frequency changes from 0.75Hz to 1.5Hz of rolling gait.



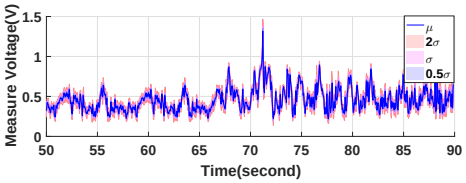
(b) CPG-based method for Frequency changes from 0.75Hz to 1.5Hz of rolling gait.



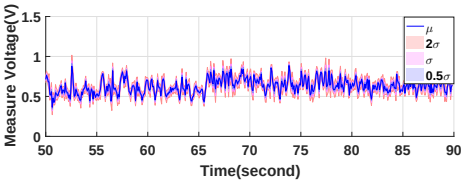
(c) Sin-based method for amplitude changes from 20° to 40° of rolling gait.



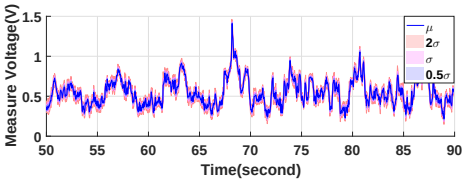
(d) CPG-based method for amplitude changes from 20° to 40° of rolling gait.



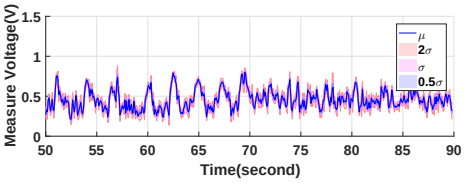
(e) Sin-based method for frequency changes from 0.75Hz to 1.5Hz of sidewinding gait.



(f) CPG-based method for frequency changes from 0.75Hz to 1.5Hz of sidewinding gait.



(g) Sin-based method for amplitude changes from 20° to 40° of sidewinding gait.



(h) CPG-based method for amplitude changes from 20° to 40° of sidewinding gait.

Figure 7.3: Voltage measurements of the high-load resistance. The left column figures are sinusoid-based method measurements. The right column figures are CPG-based method measurements.

back angle position of each module. Every slave module runs a PD controller to control the servo such that the joint can reach the desired position.

As an illustration of undesired movement, the body shape transition in rolling gait is shown in Figure. 11. The amplitude $A_{odd/even}$ is changed from 20° to 40°. As shown in Figure 7.2, the four figures from left to right, present the body shape of 20° rolling, the body shape before transition, the jerky movement during transition (the curve of the arc should increase, not decrease), and the body shape of 40° rolling. The undesired movement can not be observed in the contrast experiments, as shown in attached video.

The output torque of the snake-like robot is directly related to the current. Therefore, by measuring the current of the robot, we can obtain the changing trend of the output torque. A $0.1\Omega/50W$ high-load resistance is stringed into the circuit and the voltage of the resistance is measured by oscilloscope. Thus, the changing trend of the torque can be present by the voltage measurements.

As seen in Figure 7.3, the results of sinusoid-based method and CPG-based method are in the left column and the right column, respectively. The average value is presented with solid lines and the standard deviation of each figure is represented with σ . Figure 7.3(a, b) describe the frequency transition of rolling gait, from $0.375Hz$ to $0.75Hz$. The transition of the sinusoid-based method happens at $t \approx 73s$, when an abnormal torque is generated. However, the transition of the CPG-based method does not show obvious abnormal torque. The amplitude transition of rolling gait from 20° to 40° is shown in Figure 7.3(c, d), the similar abnormal torque happens at $t \approx 70s$ in the left. On the contrary, by adopting the CPG-based method, the voltage measurements exhibit relative smooth processes. In order to ensure the accuracy of the experiments, sidewinding gait is also conducted, changing the frequency and the amplitude as well. In Figure 7.3(e, f), frequency is adjusted from $0.375Hz$ to $0.75Hz$ at $t \approx 72s$. In Figure 7.3(g, h), amplitude is adjusted from 20° to 40° at $t \approx 68s$. Both the figures in the left show the abnormal torque when start the transition process. On the opposite side, the CPG-based method, the voltage measurements exhibit no abnormal torque during the transition.

Chapter 8

Conclusion and Future Work

This conclusion chapter will bring together and summary the main contributions of our biological-inspired hierarchical controller in this thesis, together with its pros and cons. Meanwhile, it makes suggestion for the further improvement of this work and speculates on future research direction for bio-inspired locomotion control of field robots.

8.1 Primary Contributions of the Thesis

The main contributions of this work lie in the development of the biological-inspired hierarchical controller consisting of the low-level CPG controller and the high-level SNN controller. Meanwhile, three autonomous locomotion tasks are performed on a snake-like robot based on the proposed controller. To be specific, the contributions are summarized as follow:

1. ***The spiking neural network*:**
    - **A survey of robotics control based on learning-inspired SNNs.** In chapter 2, the primary impetuses of SNN-based robotics tasks is first highlighted in terms of speed, energy efficiency, and computation capabilities. Those SNN-based robotic applications according to different learning rules are classified and those learning rules are explicated with their corresponding robotic applications.
    - **General SNN framework for mobile robot tasks.** In chapter 5, the general learning rule of an SNN for field operation tasks is given, together with some sensor encoding and motor decoding strategies and an implementation paradigm. Under this paradigm, we have implemented three different autonomous locomotion tasks.
    - **Dopamine modulation assignment within SNNs with hidden layer.** In chapter 6, a dopamine modulation assignment method is given for implementing SNNs with hidden layers. With

this method, we successfully trained SNNs with separated hidden layer and agnostic hidden layer for performing tracking task.

2. ***The central pattern generator*:**

   - **Modeling of the central pattern generator.** Based on the gradient theory, a lightweight CPG model with fast computing time is designed to control the locomotion of a snake-like robot. The CPG model is at least 3 times faster than the other widely adopted CPG models in the literature and demonstrated with experiments.
   - **Gait transition smoothness.** Simulations and prototype measurements are conducted on the robot trajectory and output torque for the change of both body shape and locomotion speed. The results prove that the CPG-based method can effectively avoid undesirable movement and abnormal torque.

3. ***The slithering gait of a wheeled snake-like robot*:**

   - **Modeling of the slithering gait.** In order to perform autonomous locomotion tasks, a slithering gait of a snake-like robot is further modeled on the basis of the serpentine curve in terms of speed control, steering control, and body shape regulation.
   - **Steering and head control.** A precise steering control of the snake-like robot is introduced and demonstrated by simulations. A head control method is proposed to overcome the head module swinging from side to side by strengthening the environment perceiving capabilities.

4. ***Implementations*:**

   - **Autonomous locomotion tasks.** In order to demonstrate and examine the effectiveness of the proposed control architecture, the target tracking, wall following, and obstacle avoidance tasks are implemented. The training details are given and analyzed, together with their performances in different testing scenarios.
   - **SNN training discussions.** According to different tasks, different SNN architectures are analyzed and the impact of different training factors are also discussed.

8.2 Shortcomings of the Control Architecture

Even we have achieved the aforementioned contributions, a number of limitations should to be noted regarding the present study.

- **Gait transition**: In chapter 4 section 4.3, we have introduced out gait transition process in terms of changing speed, body shape, and steering direction, controlled by CPG-based controller. However, in some circumstances, we have to change the gait type instead of only changing its properties, which is mainly dominated by the phase β in (3.8). Different from other parameters like the amplitude or frequency, the phase modulation process will significantly disturb the overall locomotion behavior of the snake-like robot.

- **Spatial encoding**: In chapter 6, we have utilized conventional camera as the vision sensor, from which the information is binary encoded based on spiking rate. However, other features of the visual image have been neglected, such as the color information or the depth information.

8.3 Future Work

In this section, the possible future work is given from three aspects, namely, the underlying mechanism of spiking neural network, the locomotion skill of snake-like robots, and the practical training methods for field robots.

8.3.1 Implementations with modular snake-like robot

As we all know, wheeled snake-like robots can move smoothly on the ground under serpentine gait. But their applications are greatly limited in wild field environment, since they lack of satisfactory traversability in diverse terrain. On the other hand, modular snake-like robots have great potential of being implemented in complex surroundings, since they mainly propel themselves by twisting or pushing the bodies.

Even so, there are a lot of unsolved tasks involving both the mechanism and control algorithms. For instance, the head module of the modular snake-like robot waggles around to contribute to the locomotion, which brings great difficulties for sensing the environment.

8.3.2 Practical training methods for field robots

In order to learn an expected behavior or function, the robot has to repeatedly explore and exploit the environment until it learns the task successfully. First, different from the robotics arm that can reset themselves to the initial position easily, it is nearly impossible to set field robots to the exact initial states automatically, especially when the training process can be easily up to hours and even days. Second, during this unpredictable process, the robot may break itself anytime or can not make

sure to repeat each episode without deviations. Hu *et al.* [259] present an automated learning environment for developing control policies directly on the hardware of a modular legged robot, which is relocating the robot to the initial position using a resetting mechanism. Even this brilliant relocating system help this legged robot to achieve reinforcement learning, many field robots still face this critical training problem.

One possible solution is to learn the task with simulations first and them transfer the experience to its prototype counterpart. To make sure the consistence works, we have to increase the similarity between the simulation and prototype experiment, which is usually difficult to model.

8.3.3 Spiking neural network

Although an increasing amount of work has been done to explore the theoretical foundations and practical implementations of SNNs for robotics control, many related topics need to be investigated, especially in the following areas.

8.3.3.1 Biological Mechanism

Despite the extensive exploration of the functions and structure of the brain, the exact mechanisms of learning in biological neurons remain unknown. To be specific for some of those related to robotics applications:

1. How is diverse information coded in many neural activities other than the rates and timing of spikes?

2. How are memories distinguished, stored, and retrieved in such an efficient and precise manner?

3. How do brains simulate the future, since it involves the concept of "previous steps," thus requiring some form of memory?

As long as we can constantly address these unsolved mysteries of the brain, the robots of the future definitely can achieve more advanced intelligence.

8.3.3.2 Designing and Training SNNs

None of the currently suggested algorithms are general-purpose able, at least in principle, to learn an arbitrary task in the way that backpropagation (through time) and its variants (with all their limitations) do for rate neurons [28]. Therefore, there is no general design framework that could offer the functionalities of modeling and training, as well as those substantial tools for the conventional ANNs

do, for instance, Tensorflow [260], Theano [261], and Torch [262]. The nature of this situation is that training these kind of networks is notoriously difficult, especially when it comes to deep-network architectures. Since error backpropagation mechanisms commonly used in ANNs cannot be directly transferred to SNNs due to non-differentiabilities at spike times, there has been a void of practical learning methods.

Moreover, training should strengthen the combination with the burgeoning technologies of reinforcement learning, for instance, extending SNN into deep architecture or generating continuous action space [263]. In the future, combining the R-STDP with a reward-prediction model could lead to an algorithm that is actually capable of solving sequential decision tasks such as MDPs as well.

8.3.3.3 Combination with Neuromorphic Devices

Another important general issue that needs extensive research and is not clearly defined is how to integrate SNN-based controllers into neuromorphic devices, since they have the potential to offer fundamental improvements in computational capabilities such as speed and lower power consumption [174]. These are of vital importance for robot applications, especially in mobile applications where real-time responses are important and energy supply is limited. An overview of how to program SNNs based on neuromorphic chips can be found [264].

SNNs computation can highly benefit from parallel computing, substantially more so than conventional ANNs. Unlike a traditional neuron in rate coding, a spiking neuron does not need to receive weight values from each presynaptic neuron at each compution step. Since at each time step only a few neurons are active in an SNN, the classic bottleneck of message passing is removed. Moreover, computing the updated state of membrane potential is more complex than computing a weighted sum. Therefore communication time can computation cost are much more well-balanced in SNN parallel implementation as compared to conventional ANNs.

8.3.3.4 Interdisciplinary Research of Neuroscience and Robotics

Another barrier that needs to be removed comes from a dilemma for the researchers of neuroscience and robotics: Roboticists often use a simplified brain model in a virtual robot to make a real-time simulation, or neuro-scientists develop detailed brain models that are not possible to be embedded into the real world due to their high complexity. An ongoing solution is the Neurorobotics Platform, which offers adequate tools to model virtual robots, high-fidelity environments, and complex neural network models for both neuroscientists and roboticists.

Appendix A

Appendix

Simulation parameters for the snake-like robot is shown in Table A.1. Common simulation parameters for SNNs are shown in Table A.2 The target tracking, obstacle avoidance, and wall following controllers are listed in Tables A.3, A.4, A.5, respectively.

Table A.1: Simulation Parameters for the snake-like robot.

	Parameter	**Value**	**Unit**
Slithering Gait	Amplitude	$A = 40$	degree
	Initial Temporal Frequency	$\omega = 0.5$	
	Initial Bias	$C = 0$	degree
	Spatial frequency	$\beta = 0$	
	converge factor	$\lambda = 0$	
Steering Model	Turn constant	$c_{turn} = 0.5$	
	Maximum radius	25	m
	Minimum Radius	$r = 1$	m

Table A.2: Simulation Parameters for the SNN simulation.

	Parameter	Value	Unit
LIF Neuron	NEST Model	iaf_psc_alpha	px
	Resting membrane potential	$E_L = -70.0$	mV
	Capacity of the membrane	$C_m = 250.0$	pF
	Membrane time constant	$\tau_m = 10.0$	ms
	Time constant of postsynaptic excitatory currents	$\tau_{syn,ex} = 2.0$	ms
	Time constant of postsynaptic inhibitory currents	$\tau_{syn,in} = 2.0$	ms
	Duration of refractory period	$t_{ref} = 2.0$	ms
	Reset membrane potential	$V_{reset} = -70.0$	mV
	Spike threshold	$V_{th} = -55.0$	mv
	Constant input current	$I_e = 0.0$	pA
Network Simulation	Simulation Time per Step	50.0	ms
	Time Resolution	0.1	ms

Table A.3: Simulation Parameters for target tracking task.

	Parameter	Value	Unit
SNN Input	Infrared Vision Resolution	64×64	px
	Crop Top and Bottom	24	px
	SNN Input Resolution	16×4	px
R-STDP	Maximum Synapse Value	2500.0	
	Minimum Synapse Value	-2500.0	
	Maximum Initial Random Synapse Value	501.0	
	Minimum Initial Random Synapse Value	500.0	
	Time Constant of Reward Signal	$\tau_n = 200.0$	ms
	Time Constant of Eligibility Trace	$\tau_c = 1000.0$	ms
	Reward Factor scaling the Reward Signal	$\lambda = 0.0015$	
	Constant scaling Strength of Potentiation	$A_+ = 1.0$	
	Constant scaling Strength of Depression	$A_- = 1.0$	

Table A.4: Simulation parameters for obstacle avoidance.

	Parameter	Value	Unit
SNN Input 1	Infrared Vision Resolution	64×64	px
	Crop Top and Bottom	24	px
	SNN Input Resolution	16×4	px
SNN Input 2	Proximity Sensor	$0.1 - 1$	m
R-STDP	Maximum Synapse Value	3000.0	
	Minimum Synapse Value	0	
	Maximum Initial Random Synapse Value	501.0	
	Minimum Initial Random Synapse Value	500.0	
	Time Constant of Reward Signal	$\tau_n = 200.0$	ms
	Time Constant of Eligibility Trace	$\tau_c = 1000.0$	ms
	Reward Factor scaling the Reward Signal	$\lambda = 0.0015$	
	Constant scaling Strength of Potentiation	$A_+ = 1.0$	
	Constant scaling Strength of Depression	$A_- = 1.0$	

Table A.5: Simulation Parameters for wall following task.

	Parameter	Value	Unit
SNN Input	DVS Resolution	128×128	px
	Crop Bottom	48	px
	SNN Input Resolution	8×5	px
R-STDP	Maximum Synapse Value	10000.0	
	Minimum Synapse Value	-1000.0	
	Maximum Initial Random Synapse Value	201.0	
	Minimum Initial Random Synapse Value	200.0	
	Time Constant of Reward Signal	$\tau_n = 200.0$	ms
	Time Constant of Eligibility Trace	$\tau_c = 1000.0$	ms
	Reward Factor scaling the Reward Signal	$\lambda = 0.00025$	
	Constant scaling Strength of Potentiation	$A_+ = 1.0$	
	Constant scaling Strength of Depression	$A_- = 1.0$	

References

[1] Oxford University Press, 1999. 1, 87

[2] FRASCA MATTIA, ARENA PAOLO, AND FORTUNA LUIGI. *Bio-inspired emergent control of locomotion systems*, **48**. World scientific, 2004. 2, 4

[3] E. CAMBRIA AND B. WHITE. **Jumping NLP Curves: A Review of Natural Language Processing Research [Review Article]**. *IEEE Computational Intelligence Magazine*, **9**(2):48–57, May 2014. 2

[4] JOHN C RUSS. *The image processing handbook*. CRC press, 2016. 2

[5] DAVID SILVER, JULIAN SCHRITTWIESER, KAREN SIMONYAN, IOANNIS ANTONOGLOU, AJA HUANG, ARTHUR GUEZ, THOMAS HUBERT, LUCAS BAKER, MATTHEW LAI, ADRIAN BOLTON, ET AL. **Mastering the game of Go without human knowledge**. *Nature*, **550**(7676):354, 2017. 2

[6] ALFREDO CANZIANI, ADAM PASZKE, AND EUGENIO CULURCIELLO. **An analysis of deep neural network models for practical applications**. *arXiv preprint arXiv:1605.07678*, 2016. 2

[7] DANIEL DRUBACH. *The brain explained*. Prentice Hall Health Upper Saddle River, NJ, 2000. 2, 15

[8] WOLFGANG MAASS. **Networks of spiking neurons: The third generation of neural network models**. *Neural Networks*, **10**(9):1659 – 1671, 1997. 3

[9] CHRIS ELIASMITH, TERRENCE C. STEWART, XUAN CHOO, TREVOR BEKOLAY, TRAVIS DEWOLF, YICHUAN TANG, AND DANIEL RASMUSSEN. **A Large-Scale Model of the Functioning Brain**. *Science*, **338**(6111):1202–1205, 2012. 3

[10] S. B. FURBER, F. GALLUPPI, S. TEMPLE, AND L. A. PLANA. **The SpiNNaker Project**. *Proceedings of the IEEE*, **102**(5):652–665, May 2014. 3

[11] P. LICHTSTEINER, C. POSCH, AND T. DELBRUCK. **A 128×128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor**. *IEEE Journal of Solid-State Circuits*, **43**(2):566–576, Feb 2008. 3

[12] J. L. ROSSELLÓ, V. CANALS, A. OLIVER, M. ALOMAR, AND A. MORRO. **Spiking neural networks signal processing**. In *Design of Circuits and Integrated Systems*, pages 1–6, Nov 2014. 3

[13] S. LOISELLE, J. ROUAT, D. PRESSNITZER, AND S. THORPE. **Exploration of rank order coding with spiking neural networks for speech recognition**. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, **4**, pages 2076–2080 vol. 4, July 2005. 3

[14] ERIC NICHOLS, L. J. MCDAID, AND N. H. SIDDIQUE. **CASE STUDY ON A SELF-ORGANIZING SPIKING NEURAL NETWORK FOR ROBOT NAVIGATION**. *International Journal of Neural Systems*, **20**(06):501–508, 2010. PMID: 21117272. 3

[15] AUKE JAN IJSPEERT. **Central pattern generators for locomotion control in animals and robots: A review**. *Neural Networks*, **21**(4):642 – 653, 2008. Robotics and Neuroscience. 4

[16] J. YU, M. TAN, J. CHEN, AND J. ZHANG. **A Survey on CPG-Inspired Control Models and System Implementation**. *IEEE Transactions on Neural Networks and Learning Systems*, **25**(3):441–456, March 2014. 4

[17] PÅL LILJEBÄCK, KRISTIN YTTERSTAD PETTERSEN, ØYVIND STAVDAHL, AND JAN TOMMY GRAVDAHL. *Snake robots: modelling, mechatronics, and control*. Springer Science & Business Media, 2012. 4, 6, 42, 43

[18] ACKERMAN EVAN. **What CMUs Snake Robot Team Learned While Searching for Mexican Earthquake Survivors**, 2017. 4

[19] H. ANDO, Y. AMBE, A. ISHII, M. KONYO, K. TADAKUMA, S. MARUYAMA, AND S. TADOKORO. **Aerial Hose Type Robot by Water Jet for Fire Fighting**. *IEEE Robotics and Automation Letters*, **3**(2):1128–1135, April 2018. 4

[20] E. KELASIDI, P. LILJEBACK, K. Y. PETTERSEN, AND J. T. GRAVDAHL. **Innovation in Underwater Robots: Biologically Inspired Swimming Snake Robots**. *IEEE Robotics Automation Magazine*, **23**(1):44–62, March 2016. 4

[21] ZHENSHAN BING, CLAUS MESCHEDE, GUANG CHEN, FLORIAN RÖHRBEIN, MAHMOUD AKL, KAI HUANG, AND ALOIS KNOLL. **End to End Learning of Spiking Neural Network based on R-**

STDP for a Lane Keeping Vehicle. In *2012 IEEE International Conference on Robotics and Automation*, May 2018. 6, 22

[22] Z. BING, L. CHENG, K. HUANG, M. ZHOU, AND A. KNOLL. **CPG-based control of smooth transition for body shape and locomotion speed of a snake-like robot**. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4146–4153, May 2017. 6, 42, 45

[23] ZHENSHAN BING, LONG CHENG, GUANG CHEN, FLORIAN RÖHRBEIN, KAI HUANG, AND ALOIS KNOLL. **Towards autonomous locomotion: CPG-based control of smooth 3D slithering gait transition of a snake-like robot**. *Bioinspiration & Biomimetics*, **12**(3):035001, 2017. 6, 45, 61

[24] Z. BING, L. CHENG, K. HUANG, Z. JIANG, G. CHEN, FLORIAN RÖHRBEIN, AND A. KNOLL. **Towards autonomous locomotion: Slithering gait design of a snake-like robot for target observation and tracking**. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2698–2703, Sept 2017. 6, 46

[25] H. OHNO AND S. HIROSE. **Design of slim slime robot and its gait of locomotion**. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, **2**, pages 707–715 vol.2, 2001. 7, 44

[26] J. KAISER, J. C. V. TIECK, C. HUBSCHNEIDER, P. WOLF, M. WEBER, M. HOFF, A. FRIEDRICH, K. WOJTASIK, A. ROENNAU, R. KOHLHAAS, R. DILLMANN, AND J. M. ZÖLLNER. **Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks**. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 127–134, Dec 2016. 8, 20, 24, 41

[27] FILIP PONULAK AND ANDRZEJ KASINSKI. **Introduction to spiking neural networks: Information processing, learning and applications**. *Acta neurobiologiae experimentalis*, **71**(4):409—433, 2011. 11, 18

[28] ANDRÉ GRÜNING AND SANDER M BOHTE. **Spiking Neural Networks: Principles and Challenges.** In *ESANN*, 2014. 11, 126

[29] SAMANWOY GHOSH-DASTIDAR AND HOJJAT ADELI. **Spiking neural networks**. *International journal of neural systems*, **19**(04):295–308, 2009. 11, 83

[30] JILLES VREEKEN. **Spiking neural networks, an introduction**, 2003. 11

[31] **Neuron - Wikipedia**. http://en.wikipedia.org/wiki/Neuron. Accessed: 2017-01-30. 12

[32] **Synapse - Wikipedia**. http://en.wikipedia.org/wiki/Synapse. Accessed: 2017-01-30. 12

[33] SUZANA HERCULANO-HOUZEL. **The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost**. *Proceedings of the National Academy of Sciences*, **109**(Supplement 1):10661–10668, 2012. 11

[34] WOLFGANG MAASS. **Computing with spikes**. *Special Issue on Foundations of Information Processing of TELEMATIK*, **8**(1):32–36, 2002. 11

[35] WARREN S. MCCULLOCH AND WALTER PITTS. **A logical calculus of the ideas immanent in nervous activity**. *The bulletin of mathematical biophysics*, **5**(4):115–133, Dec 1943. 13

[36] J J HOPFIELD. **Neural networks and physical systems with emergent collective computational abilities**. *Proceedings of the National Academy of Sciences*, **79**(8):2554–2558, 1982. 13

[37] JUN HAN AND CLAUDIO MORAGA. **The influence of the sigmoid function parameters on the speed of backpropagation learning**. In JOSÉ MIRA AND FRANCISCO SANDOVAL, editors, *From Natural to Artificial Neural Computation*, pages 195–201, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. 13

[38] KURT HORNIK, MAXWELL STINCHCOMBE, AND HALBERT WHITE. **Multilayer feedforward networks are universal approximators**. *Neural Networks*, **2**(5):359 – 366, 1989. 13

[39] ROBERT HECHT-NIELSEN. **Theory of the backpropagation neural network**. In *Neural networks for perception*, pages 65–93. Elsevier, 1992. 13

[40] WOLFGANG MAASS. **Networks of spiking neurons: The third generation of neural network models**. *Neural Networks*, **10**(9):1659 – 1671, 1997. 14, 15

[41] WOLFGANG MAASS. **On the relevance of time in neural computation and learning**. In MING LI AND AKIRA MARUOKA, editors, *Algorithmic Learning Theory*, pages 364–384, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. 14

[42] MAURIZIO FIASCHÉ AND MARCO TAISCH. 14

[43] BHASKAR DASGUPTA AND GEORG SCHNITGER. **The power of approximating: a comparison of activation functions**. In *Advances in neural information processing systems*, pages 615–622, 1993. 14

[44] MARVIN M CHUN AND MARY C POTTER. **A two-stage model for multiple target detection in rapid serial visual presentation.** *Journal of Experimental psychology: Human perception and performance*, **21**(1):109, 1995. 14

[45] JUN HAENG LEE, TOBI DELBRUCK, AND MICHAEL PFEIFFER. **Training Deep Spiking Neural Networks Using Backpropagation**. *Frontiers in Neuroscience*, **10**:508, 2016. 14

[46] DAVID FERSTER AND NELSON SPRUSTON. **Cracking the Neuronal Code**. *Science*, **270**(5237):756–757, 1995. 14

[47] FLORIAN WALTER, FLORIAN RÖHRBEIN, AND ALOIS KNOLL. **Computation by Time**. *Neural Processing Letters*, **44**(1):103–124, 2016. 15

[48] SIMON THORPE, ARNAUD DELORME, AND RUFIN VAN RULLEN. **Spike-based strategies for rapid processing**. *Neural Networks*, **14**(6):715 – 725, 2001. 15

[49] B. SENGUPTA AND M. B. STEMMLER. **Power Consumption During Neuronal Computation**. *Proceedings of the IEEE*, **102**(5):738–750, May 2014. 15

[50] RUFIN VANRULLEN, RUDY GUYONNEAU, AND SIMON J. THORPE. **Spike times make sense**. *Trends in Neurosciences*, **28**(1):1 – 4, 2005. 16

[51] ARTHUR R HOUWELING AND MICHAEL BRECHT. **Behavioural report of single neuron stimulation in somatosensory cortex**. *Nature*, **451**(7174):65, 2008. 16

[52] DANIEL HUBER, LEOPOLDO PETREANU, NIMA GHITANI, SACHIN RANADE, TOMÁŠ HROMÁDKA, ZACH MAINEN, AND KAREL SVOBODA. **Sparse optical microstimulation in barrel cortex drives learned behaviour in freely moving mice**. *Nature*, **451**(7174):61, 2008. 16

[53] JASON WOLFE, ARTHUR R HOUWELING, AND MICHAEL BRECHT. **Sparse and powerful cortical spikes**. *Current opinion in neurobiology*, **20**(3):306–312, 2010. 16

[54] SUZANA HERCULANO-HOUZEL, KAMILLA AVELINO-DE SOUZA, KLEBER NEVES, JAIRO PORFÍRIO, DÉBORA MESSEDER, LARISSA MATTOS FEIJÓ, JOSÉ MALDONADO, AND PAUL R. MANGER. **The elephant brain in numbers**. *Frontiers in Neuroanatomy*, **8**:46, 2014. 16

[55] W GERSTNER, R KEMPTER, JL VAN HEMMEN, AND H WAGNER. **Hebbian Learning of Pulse Timing in the Barn Owl Auditory System, ser. Pulsed Neural Networks, W. Maass and CM Bishop, Eds**, 1999. 16, 79

[56] W. METZNER, C. KOCH, R. WESSEL, AND F. GABBIANI. **Feature Extraction by Burst-Like Spike Patterns in Multiple Sensory Maps**. *Journal of Neuroscience*, **18**(6):2283–2300, 1998. 16

[57] FABRIZIO GABBIANI, WALTER METZNER, RALF WESSEL, AND CHRISTOF KOCH. **From stimulus encoding to feature extraction in weakly electric fish**. *Nature*, **384**(6609):564, 1996. 16

[58] S. M. BOHTE, H. LA POUTRE, AND J. N. KOK. **Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks**. *IEEE Transactions on Neural Networks*, **13**(2):426–435, March 2002. 16

[59] JOHN J HOPFIELD. **Pattern recognition computation using action potential timing for stimulus representation**. *Nature*, **376**(6535):33, 1995. 16

[60] SIMON THORPE, ARNAUD DELORME, AND RUFIN VAN RULLEN. **Spike-based strategies for rapid processing**. *Neural Networks*, **14**(6):715 – 725, 2001. 16

[61] RUDY GUYONNEAU, RUFIN VANRULLEN, AND SIMON J. THORPE. **Temporal codes and sparse representations: A key to understanding rapid processing in the visual system**. *Journal of Physiology-Paris*, **98**(4):487 – 497, 2004. Decoding and interfacing the brain: from neuronal assemblies to cyborgs. 16

[62] J. SHIN, D. SMITH, W. SWIERCZ, K. STALEY, J. T. RICKARD, J. MONTERO, L. A. KURGAN, AND K. J. CIOS. **Recognition of Partially Occluded and Rotated Images With a Network of Spiking Neurons**. *IEEE Transactions on Neural Networks*, **21**(11):1697–1709, Nov 2010. 16

[63] S. GSPANDL, S. PODESSER, M. REIP, G. STEINBAUER, AND M. WOLFRAM. **A dependable perception-decision-execution cycle for autonomous robots**. In *2012 IEEE International Conference on Robotics and Automation*, pages 2992–2998, May 2012. 17

[64] HODGKIN A. L. AND HUXLEY A. F. **A quantitative description of membrane current and its application to conduction and excitation in nerve**. *The Journal of Physiology*, **117**(4):500–544. 18

[65] A. N. BURKITT. **A Review of the Integrate-and-fire Neuron Model: I. Homogeneous Synaptic Input**. *Biological Cybernetics*, **95**(1):1–19, Jul 2006. 18

[66] EUGENE M IZHIKEVICH. **Which model to use for cortical spiking neurons?** *IEEE transactions on neural networks*, **15**(5):1063–1070, 2004. 18, 19

[67] RICHARD B. STEIN. **A Theoretical Analysis of Neuronal Variability**. *Biophysical Journal*, **5**(2):173 – 194, 1965. 18

[68] ALEX M ANDREW. **Spiking neuron models: Single neurons, populations, plasticity**. *Kybernetes*, **32**(7/8), 2003. 18, 20

[69] VALENTINO BRAITENBERG. **Cell assemblies in the cerebral cortex**. In *Theoretical approaches to complex systems*, pages 171–188. Springer, 1978. 19

[70] WULFRAM GERSTNER AND WERNER M KISTLER. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002. 19

[71] ROBERT GÜTIG AND HAIM SOMPOLINSKY. **The tempotron: a neuron that learns spike timing–based decisions**. *Nature neuroscience*, **9**(3):420, 2006. 19, 78

[72] 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 ALESSANDRO E. P. VILLA, WŁODZISŁAW DUCH, PÉTER ÉRDI, FRANCESCO MASULLI, AND GÜNTHER PALM, editors, *Artificial Neural Networks and Machine Learning – ICANN 2012*, pages 209–216, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 19, 22, 37, 38, 78

[73] ROBERT URBANCZIK AND WALTER SENN. **Reinforcement learning in populations of spiking neurons**. *Nature neuroscience*, **12**(3):250, 2009. 19, 78

[74] JOHN J WADE, LIAM J MCDAID, JOSE A SANTOS, AND HEATHER M SAYERS. **SWAT: a spiking neural network training algorithm for classification problems**. *IEEE Transactions on Neural Networks*, **21**(11):1817–1830, 2010. 19

[75] WULFRAM GERSTNER, RICHARD KEMPTER, J LEO VAN HEMMEN, AND HERMANN WAGNER. **A neuronal learning rule for sub-millisecond temporal coding**. *Nature*, **383**(6595):76, 1996. 20

[76] ERIC R KANDEL, JAMES H SCHWARTZ, THOMAS M JESSELL, DEPARTMENT OF BIOCHEMISTRY, MOLECULAR BIOPHYSICS THOMAS JESSELL, STEVEN SIEGELBAUM, AND AJ HUDSPETH. *Principles of neural science*, **4**. McGraw-hill New York, 2000. 20

[77] EDGAR D ADRIAN. **The impulses produced by sensory nerve endings**. *The Journal of physiology*, **61**(1):49–72, 1926. 20

[78] CHRIS BISHOP, CHRISTOPHER M BISHOP, ET AL. *Neural networks for pattern recognition*. Oxford university press, 1995. 20

[79] MIROSLAV KUBAT. **Neural networks: a comprehensive foundation by Simon Haykin, Macmillan, 1994, ISBN 0-02-352781-7.-**. *The Knowledge Engineering Review*, **13**(4):409–412, 1999. 20

[80] RICHARD KEMPTER, WULFRAM GERSTNER, AND J LEO VAN HEMMEN. **Hebbian learning and spiking neurons**. *Physical Review E*, **59**(4):4498, 1999. 21

[81] WULFRAM GERSTNER, RAPHAEL RITZ, AND J. LEO VAN HEMMEN. **Why spikes? Hebbian learning and retrieval of time-resolved excitation patterns**. *Biological Cybernetics*, **69**(5):503–515, Oct 1993. 21

[82] PATRICK D ROBERTS. **Computational consequences of temporally asymmetric learning rules**. *Journal of Computational Neuroscience*, **7**(3):235–246, 1999. 21

[83] BERTHOLD RUF AND MICHAEL SCHMITT. **Unsupervised learning in networks of spiking neurons using temporal coding**. In WULFRAM GERSTNER, ALAIN GERMOND, MARTIN HASLER, AND JEAN-DANIEL NICOUD, editors, *Artificial Neural Networks — ICANN'97*, pages 361–366, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. 21

[84] WALTER SENN, MISHA TSODYKS, AND HENRY MARKRAM. **An algorithm for synaptic modification based on exact timing of pre-and post-synaptic action potentials**. In *International Conference on Artificial Neural Networks*, pages 121–126. Springer, 1997. 21

[85] HENRY MARKRAM, JOACHIM LÜBKE, MICHAEL FROTSCHER, AND BERT SAKMANN. **Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs**. *Science*, **275**(5297):213–215, 1997. 21

[86] GUO-QIANG BI AND MU-MING POO. **Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type**. *Journal of neuroscience*, **18**(24):10464–10472, 1998. 21

[87] WULFRAM GERSTNER AND WERNER M. KISTLER. **Mathematical formulations of Hebbian learning**. *Biological Cybernetics*, **87**(5):404–415, Dec 2002. 21

[88] HIDEYUKI CÂTEAU AND TOMOKI FUKAI. **A stochastic method to predict the consequence of arbitrary forms of spike-timing-dependent plasticity**. *Neural Computation*, **15**(3):597–620, 2003. 21

[89] JONATHAN RUBIN, DANIEL D LEE, AND H SOMPOLINSKY. **Equilibrium properties of temporally asymmetric Hebbian plasticity**. *Physical review letters*, **86**(2):364, 2001. 21

[90] SEN SONG, KENNETH D MILLER, AND LARRY F ABBOTT. **Competitive Hebbian learning through spike-timing-dependent synaptic plasticity**. *Nature neuroscience*, **3**(9):919, 2000. 21

[91] PETER DIEHL AND MATTHEW COOK. **Unsupervised learning of digit recognition using spike-timing-dependent plasticity**. *Frontiers in Computational Neuroscience*, **9**:99, 2015. 21

[92] LAURENT PERRINET, MANUEL SAMUELIDES, AND SIMON THORPE. **Sparse spike coding in an asynchronous feed-forward multi-layer neural network using matching pursuit**. *Neurocomputing*, **57**:125 – 134, 2004. New Aspects in Neurocomputing: 10th European Symposium on Artificial Neural Networks 2002. 21

[93] OLIVIER ROCHEL, DOMINIQUE MARTINEZ, ETIENNE HUGUES, AND FRÉDÉRIC SARRY. **Stereo-olfaction with a sniffing neuromorphic robot using spiking neurons**. In *16th European Conference on Solid-State Transducers-EUROSENSORS*, pages 4–p, 2002. 22

[94] A. CASSIDY AND V. EKANAYAKE. **A biologically inspired tactile sensor array utilizing phase-based computation**. In *2006 IEEE Biomedical Circuits and Systems Conference*, pages 45–48, Nov 2006. 22

[95] **Recursive neural network - Wikipedia**. `https://en.wikipedia.org/wiki/Recursive_neural_network`. Accessed: 2017-01-30. 22

[96] N. KUBOTA AND K. NISHIDA. **The Role of Spiking Neurons for Visual Perception of a Partner Robot**. In *2006 IEEE International Conference on Fuzzy Systems*, pages 122–129, 2006. 22

[97] ELMAR RUECKERT, DAVID KAPPEL, DANIEL TANNEBERG, DEJAN PECEVSKI, AND JAN PETERS. **Recurrent spiking networks solve planning tasks**. *Scientific reports*, **6**:21142, 2016. 22, 32

[98] HÉDI SOULA, ARAVIND ALWAN, AND GUILLAUME BESLON. **Learning at the edge of chaos: Temporal coupling of spiking neurons controller for autonomous robotic**. In *Proceedings of the AAAI spring symposia on developmental robotics*. AAAI Palo Alto, CA, 2005. 22

[99] ELENI VASILAKI, NICOLAS FRÉMAUX, ROBERT URBANCZIK, WALTER SENN, AND WULFRAM GERSTNER. **Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail**. *PLoS computational biology*, **5**(12):e1000586, 2009. 22

[100] M. A. LEWIS, R. ETIENNE-CUMMINGS, A. H. COHEN, AND M. HARTMANN. **Toward biomorphic control using custom aVLSI CPG chips**. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, **1**, pages 494–500 vol.1, 2000. 24

[101] ALESSANDRO AMBROSANO, LORENZO VANNUCCI, UGO ALBANESE, MURAT KIRTAY, EGIDIO FALOTICO, GEORG HINKEL, JACQUES KAISER, STEFAN ULBRICH, PAUL LEVI, CHRISTIAN MORILLAS, ET AL. **Retina color-opponency based pursuit implemented through spiking neural networks in the neurorobotics platform**. In *Conference on Biomimetic and Biohybrid Systems*, pages 16–27. Springer, 2016. 24, 41

[102] X. WANG, Z. G. HOU, M. TAN, Y. WANG, AND L. HU. **The Wall-Following Controller for the Mobile Robot Using Spiking Neurons**. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, **1**, pages 194–199, Nov 2009. 24

[103] HEBB DO. *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949. 24

[104] GEOFFREY E HINTON, TERRENCE JOSEPH SEJNOWSKI, AND TOMASO A POGGIO. *Unsupervised learning: foundations of neural computation*. MIT press, 1999. 25

[105] XIUQING WANG, ZENG-GUANG HOU, ANMIN ZOU, MIN TAN, AND LONG CHENG. **A behavior controller based on spiking neural networks for mobile robots**. *Neurocomputing*, **71**(4):655 – 666, 2008. Neural Networks: Algorithms and Applications 50 Years of Artificial Intelligence: a Neuronal Approach. 25, 27, 34

[106] XIUQING WANG, ZENG-GUANG HOU, FENG LV, MIN TAN, AND YONGJI WANG. **Mobile robot modular navigation controller using spiking neural networks**. *Neurocomputing*, **134**:230 – 238, 2014. Special issue on the 2011 Sino-foreign-interchange Workshop on Intelligence Science and Intelligent Data Engineering (IScIDE 2011) Learning Algorithms and Applications. 25, 27

[107] P. ARENA, S. DE FIORE, L. PATANÉ, M. POLLINO, AND C. VENTURA. **Insect inspired unsupervised learning for tactic and phobic behavior enhancement in a hybrid robot**. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2010. 25, 27, 34

[108] ERIC I KNUDSEN. **Supervised learning in the brain**. *Journal of Neuroscience*, **14**(7):3985–3997, 1994. 25

[109] WT THACH. **On the specific role of the cerebellum in motor learning and cognition: Clues from PET activation and lesion studies in man**. *Behavioral and Brain Sciences*, **19**(3):411–433, 1996. 25

[110] JOHN MONTGOMERY, GUY CARTON, AND DAVID BODZNICK. **Error-driven motor learning in fish**. *The Biological Bulletin*, **203**(2):238–239, 2002. 25, 26

[111] R CHRISTOPHER MIALL AND DANIEL M WOLPERT. **Forward models for physiological motor control**. *Neural networks*, **9**(8):1265–1279, 1996. 26

[112] MITSUO KAWATO AND HIROAKI GOMI. **A computational model of four regions of the cerebellum based on feedback-error learning**. *Biological cybernetics*, **68**(2):95–103, 1992. 26

[113] MEGAN R CAREY, JAVIER F MEDINA, AND STEPHEN G LISBERGER. **Instructive signals for motor learning from visual cortical area MT**. *Nature neuroscience*, **8**(6):813, 2005. 26

[114] RICHARD R. CARRILLO, EDUARDO ROS, CHRISTIAN BOUCHENY, AND OLIVIER J.-M.D. COENEN. **A real-time spiking cerebellum model for learning robot control**. *Biosystems*, **94**(1):18 – 27, 2008. Seventh International Workshop on Information Processing in Cells and Tissues. 26, 34

[115] A. BOUGANIS AND M. 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)*, pages 1–8, July 2010. 26, 34

[116] **Hond van Pavlov - Wikipedia**. `https://nl.wikipedia.org/wiki/Hond_van_Pavlov`. Accessed: 2017-07-05. 27

[117] **Classical conditioning - Wikipedia**. `https://en.wikipedia.org/wiki/Classical_conditioning`. Accessed: 2017-07-05. 27

[118] IVAN P PAVLOV. **Conditioned reflexes**. *Trans, by GV*, **7**, 1927. 27

[119] P. ARENA, L. FORTUNA, M. FRASCA, AND L. PATANE. **Learning Anticipation via Spiking Networks: Application to Navigation Control**. *IEEE Transactions on Neural Networks*, **20**(2):202–216, Feb 2009. 27, 35

[120] P ARENA, S DE FIORE, L PATANÉ, M POLLINO, AND C VENTURA. **STDP-based behavior learning on the TriBot robot**. In *Bioengineered and Bioinspired Systems IV*, **7365**, page 736506. International Society for Optics and Photonics, 2009. 27, 35

[121] ANDRÉ CYR AND MOUNIR BOUKADOUM. **Classical conditioning in different temporal constraints: an STDP learning rule for robots controlled by spiking neural networks**. *Adaptive Behavior*, **20**(4):257–272, 2012. 27, 35

[122] KENJI IWADATE, IKUO SUZUKI, MICHIKO WATANABE, MASAHITO YAMAMOTO, AND MASASHI FURUKAWA. **An Artificial Neural Network Based on the Architecture of the Cerebellum for Behavior Learning**. In YOUNG IM CHO AND ERIC T. MATSON, editors, *Soft Computing in Artificial Intelligence*, pages 143–151, Cham, 2014. Springer International Publishing. 28, 35

[123] CRISTIAN JIMENEZ-ROMERO, DAVID SOUSA-RODRIGUES, JEFFREY H JOHNSON, AND VITORINO RAMOS. **A model for foraging ants, controlled by spiking neural networks and double pheromones**. *arXiv preprint arXiv:1507.08467*, 2015. 28, 34

[124] CRISTIAN JIMENEZ-ROMERO. *A Heterosynaptic Spiking Neural System for the Development of Autonomous Agents*. PhD thesis, The Open University, 2017. 28, 34

[125] CRISTIAN JIMENEZ-ROMERO, DAVID SOUSA-RODRIGUES, AND JEFFREY JOHNSON. **Designing Behaviour in Bio-inspired Robots Using Associative Topologies of Spiking-Neural-Networks**. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 197–200. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016. 28, 34

[126] ANDRÉ CYR, MOUNIR BOUKADOUM, AND FRÉDÉRIC THÉRIAULT. **Operant conditioning: a minimal components requirement in artificial spiking neurons designed for bio-inspired robot's controller**. *Frontiers in neurorobotics*, **8**:21, 2014. 28, 35

[127] ANDRÉ CYR AND FRÉDÉRIC THÉRIAULT. **Action selection and operant conditioning: a neuro-robotic implementation**. *Journal of Robotics*, **2015**:6, 2015. 28, 35

[128] E. DUMESNIL, P. O. BEAULIEU, AND M. BOUKADOUM. **Robotic implementation of classical and Operant Conditioning as a single STDP learning process**. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 5241–5247, July 2016. 29, 34

[129] E. DUMESNIL, P. O. BEAULIEU, AND M. BOUKADOUM. **Robotic implementation of classical and operant conditioning within a single SNN architecture**. In *2016 IEEE 15th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*, pages 322–330, Aug 2016. 29, 34

[130] MICHAEL E. HASSELMO. **Neuromodulation: acetylcholine and memory consolidation**. *Trends in Cognitive Sciences*, **3**(9):351 – 359, 1999. 29

[131] WOLFRAM SCHULTZ. **Predictive reward signal of dopamine neurons**. *Journal of neurophysiology*, **80**(1):1–27, 1998. 29

[132] EUGENE M. IZHIKEVICH. **Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling**. *Cerebral Cortex*, **17**(10):2443–2452, 2007. 29, 76

[133] RĂZVAN V FLORIAN. **Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity**. *Neural Computation*, **19**(6):1468–1502, 2007. 29, 76

[134] RICHARD EVANS. **Reinforcement learning in a neurally controlled robot using dopamine modulated STDP**. *arXiv preprint arXiv:1502.06096*, 2015. 30, 35

[135] FARAMARZ FAGHIHI, AHMED A. MOUSTAFA, RALF HEINRICH, AND FLORENTIN WÖRGÖTTER. **A computational model of conditioning inspired by Drosophila olfactory system**. *Neural Networks*, **87**:96 – 108, 2017. 30, 34

[136] T. S. CLAWSON, S. FERRARI, S. B. FULLER, AND R. J. WOOD. **Spiking neural network (SNN) control of a flapping insect-scale robot**. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 3381–3388, Dec 2016. 30, 35

[137] G. FODERARO, C. HENRIQUEZ, AND S. FERRARI. **Indirect training of a spiking neural network for flight control via spike-timing-dependent synaptic plasticity**. In *49th IEEE Conference on Decision and Control (CDC)*, pages 911–917, Dec 2010. 30, 34

[138] XU ZHANG, GREG FODERARO, CRAIG HENRIQUEZ, ANTONIUS MJ VANDONGEN, AND SILVIA FERRARI. **A radial basis function spike model for indirect learning via integrate-and-fire sampling and reconstruction techniques**. *Advances in Artificial Neural Systems*, **2012**:10, 2012. 31

[139] X. ZHANG, Z. XU, C. HENRIQUEZ, AND S. FERRARI. **Spike-based indirect training of a spiking neural network-controlled virtual insect**. In *52nd IEEE Conference on Decision and Control*, pages 6798–6805, Dec 2013. 31, 34

[140] DI HU, XU ZHANG, ZIYE XU, S. FERRARI, AND P. MAZUMDER. **Digital implementation of a spiking neural network (SNN) capable of spike-timing-dependent plasticity (STDP) learning**. In *14th IEEE International Conference on Nanotechnology*, pages 873–876, Aug 2014. 31, 34

[141] P. MAZUMDER, D. HU, I. EBONG, X. ZHANG, Z. XU, AND S. FERRARI. **Digital implementation of a virtual insect trained by spike-timing dependent plasticity**. *Integration, the VLSI Journal*, **54**:109 – 117, 2016. 31, 34

[142] GEORGE L. CHADDERDON, SAMUEL A. NEYMOTIN, CLIFF C. KERR, AND WILLIAM W. LYTTON. **Reinforcement Learning of Targeted Movement in a Spiking Neuronal Model of Motor Cortex**. *PLOS ONE*, **7**(10):1–8, 10 2012. 31, 35

[143] SAMUEL A NEYMOTIN, GEORGE L CHADDERDON, CLIFF C KERR, JOSEPH T FRANCIS, AND WILLIAM W LYTTON. **Reinforcement learning of two-joint virtual arm reaching in a computer model of sensorimotor cortex**. *Neural computation*, **25**(12):3263–3293, 2013. 31, 35

[144] MARTIN SPÜLER, SEBASTIAN NAGEL, AND WOLFGANG ROSENSTIEL. **A spiking neuronal model learning a motor control task by reinforcement learning and structural synaptic plasticity**. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015. 31, 35

[145] SALVADOR DURA-BERNAL, XIANLIAN ZHOU, SAMUEL A. NEYMOTIN, ANDRZEJ PRZEKWAS, JOSEPH T. FRANCIS, AND WILLIAM W. LYTTON. **Cortical Spiking Network Interfaced with Virtual Musculoskeletal Arm and Robotic Arm**. *Frontiers in Neurorobotics*, **9**:13, 2015. 31, 35

[146] MEHMET KOCATURK, HALIL OZCAN GULCUR, AND RESIT CANBEYLI. **Toward Building Hybrid Biological/in silico Neural Networks for Motor Neuroprosthetic Control**. *Frontiers in Neurorobotics*, **9**:8, 2015. 31

[147] M. SARIM, T. SCHULTZ, R. JHA, AND M. KUMAR. **Ultra-low energy neuromorphic device based navigation approach for biomimetic robots**. In *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*, pages 241–247, July 2016. 31, 34

[148] MOHAMMAD SARIM, THOMAS SCHULTZ, MANISH KUMAR, AND RASHMI JHA. **An artificial brain mechanism to develop a learning paradigm for robot navigation**. In *ASME 2016 Dynamic Systems and Control Conference*, pages V001T03A004–V001T03A004. American Society of Mechanical Engineers, 2016. 31, 34

[149] TING-SHUO CHOU, LIAM BUCCI, AND JEFFREY KRICHMAR. **Learning touch preferences with a tactile robot using dopamine modulated STDP in a model of insular cortex**. *Frontiers in Neurorobotics*, **9**:6, 2015. 31, 35

[150] WIEBKE POTJANS, ABIGAIL MORRISON, AND MARKUS DIESMANN. **A spiking neural network model of an actor-critic learning agent**. *Neural computation*, **21**(2):301–339, 2009. 32

[151] NICOLAS FRÉMAUX, HENNING SPREKELER, AND WULFRAM GERSTNER. **Reinforcement learning using a continuous time actor-critic framework with spiking neurons**. *PLoS computational biology*, **9**(4):e1003024, 2013. 32

[152] E. NICHOLS, L. J. MCDAID, AND N. SIDDIQUE. **Biologically Inspired SNN for Robot Control**. *IEEE Transactions on Cybernetics*, **43**(1):115–128, Feb 2013. 32

[153] JOHANNES FRIEDRICH AND MÁTÉ LENGYEL. **Goal-Directed Decision Making with Spiking Neurons**. *Journal of Neuroscience*, **36**(5):1529–1546, 2016. 32

[154] XIUQING WANG, ZENG-GUANG HOU, FENG LV, MIN TAN, AND YONGJI WANG. **Mobile robots modular navigation controller using spiking neural networks**. *Neurocomputing*, **134**:230 – 238, 2014. Special issue on the 2011 Sino-foreign-interchange Workshop on Intelligence Science and Intelligent Data Engineering (IScIDE 2011) Learning Algorithms and Applications. 34

[155] LEI WANG, SIMON X YANG, AND MOHAMMAD BIGLARBEGIAN. **Bio-inspired navigation of mobile robots**. In *Autonomous and Intelligent Systems*, pages 59–68. Springer, 2012. 35

[156] STEVEN SKORHEIM, PETER LONJERS, AND MAXIM BAZHENOV. **A Spiking Network Model of Decision Making Employing Rewarded STDP**. *PLOS ONE*, **9**(3):1–15, 03 2014. 35

[157] LOVISA IRPA HELGADOTTIR, JOACHIM HAENICKE, TIM LANDGRAF, RAUL ROJAS, AND MARTIN P NAWROT. **Conditioned behavior in a robot controlled by a spiking neural network**. In *International IEEE/EMBS Conference on Neural Engineering, NER*, pages 891–894, 2013. 35

[158] ZBIGNIEW MICHALEWICZ. **Evolution strategies and other methods**. In *Genetic Algorithms+ Data Structures= Evolution Programs*, pages 159–177. Springer, 1996. 36

[159] DARIO FLOREANO AND CLAUDIO MATTIUSSI. **Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots**. In TAKASHI GOMI, editor, *Evolutionary Robotics. From Intelligent Robotics to Artificial Life*, pages 38–61, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 36, 38

[160] HANI HAGRAS, ANTHONY POUNDS-CORNISH, MARTIN COLLEY, VICTOR CALLAGHAN, AND GRAHAM CLARKE. **Evolving spiking neural network controllers for autonomous robots**. In *IEEE*

International conference on robotics and automation, **5**, pages 4620–4626. IEEE; 1999, 2004. 36, 38

[161] DAVID HOWARD AND ALBERTO ELFES. **Evolving spiking networks for turbulence-tolerant quadrotor control**. Citeseer. 36, 38

[162] R. BATLLORI, C.B. LARAMEE, W. LAND, AND J.D. SCHAFFER. **Evolving spiking neural networks for robot control**. *Procedia Computer Science*, **6**:329 – 334, 2011. Complex adaptive sysytems. 36, 38

[163] URSZULA MARKOWSKA-KACZMAR AND MATEUSZ KOLDOWSKI. **Spiking neural network vs multilayer perceptron: who is the winner in the racing car computer game**. *Soft Computing*, **19**(12):3465–3478, Dec 2015. 36, 38

[164] FADY ALNAJJAR AND K. MURASE. **Self-Organization of Spiking Neural Network Generating Autonomous Behavior in a Miniature Mobile Robot**. In KAZUYUKI MURASE, KOSUKE SEKIYAMA, TOMOHIDE NANIWA, NAOYUKI KUBOTA, AND JOAQUIN SITTE, editors, *Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005)*, pages 255–260, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 36, 38

[165] TADASHI YAMAZAKI AND SHIGERU TANAKA. **The cerebellum as a liquid state machine**. *Neural Networks*, **20**(3):290 – 297, 2007. Echo State Networks and Liquid State Machines. 37

[166] HARALD BURGSTEINER. **Training networks of biological realistic spiking neurons for real-time robot control**. In *Proceedings of the 9th international conference on engineering applications of neural networks, Lille, France*, pages 129–136, 2005. 37, 38

[167] ELEONORA ARENA, PAOLO ARENA, ROLAND STRAUSS, AND LUCA PATANÉ. **Motor-Skill Learning in an Insect Inspired Neuro-Computational Control System**. *Frontiers in Neurorobotics*, **11**:12, 2017. 37, 38, 78

[168] DARIO FLOREANO, YANN EPARS, JEAN-CHRISTOPHE ZUFFEREY, AND CLAUDIO MATTIUSSI. **Evolution of spiking neural circuits in autonomous mobile robots**. *International Journal of Intelligent Systems*, **21**(9):1005–1024, 2006. 38, 78

[169] N. KUBOTA. **A spiking neural network for behavior learning of a mobile robot in a dynamic environment**. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, **6**, pages 5783–5788 vol.6, Oct 2004. 38

[170] S. B. FURBER, D. R. LESTER, L. A. PLANA, J. D. GARSIDE, E. PAINKRAS, S. TEMPLE, AND A. D. BROWN. **Overview of the SpiNNaker System Architecture**. *IEEE Transactions on Computers*, **62**(12):2454–2467, Dec 2013. 39

[171] PAUL A. MEROLLA, JOHN V. ARTHUR, RODRIGO ALVAREZ-ICAZA, ANDREW S. CASSIDY, JUN SAWADA, FILIPP AKOPYAN, BRYAN L. JACKSON, NABIL IMAM, CHEN GUO, YUTAKA NAKAMURA, BERNARD BREZZO, IVAN VO, STEVEN K. ESSER, RATHINAKUMAR APPUSWAMY, BRIAN TABA, ARNON AMIR, MYRON D. FLICKNER, WILLIAM P. RISK, RAJIT MANOHAR, AND DHARMENDRA S. MODHA. **A million spiking-neuron integrated circuit with a scalable communication network and interface**. *Science*, **345**(6197):668–673, 2014. 39

[172] B. V. BENJAMIN, P. GAO, E. MCQUINN, S. CHOUDHARY, A. R. CHANDRASEKARAN, J. M. BUSSAT, R. ALVAREZ-ICAZA, J. V. ARTHUR, P. A. MEROLLA, AND K. BOAHEN. **Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations**. *Proceedings of the IEEE*, **102**(5):699–716, May 2014. 39

[173] KIT CHEUNG, SIMON R. SCHULTZ, AND WAYNE LUK. **NeuroFlow: A General Purpose Spiking Neural Network Simulation Platform using Customizable Processors**. *Frontiers in Neuroscience*, **9**:516, 2016. 39

[174] IK SCHULLER AND R STEVENS. **Neuromorphic computing: from materials to systems architecture**. *report of a roundtable convened to consider neuromorphic computing basic research needs*, 2015. 39, 127

[175] S. IVALDI, J. PETERS, V. PADOIS, AND F. NORI. **Tools for simulating humanoid robot dynamics: A survey based on user feedback**. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 842–849, Nov 2014. 39

[176] N. KOENIG AND A. HOWARD. **Design and use paradigms for Gazebo, an open-source multi-robot simulator**. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, **3**, pages 2149–2154 vol.3, Sept 2004. 39

[177] RUSSELL SMITH ET AL. **Open dynamics engine**. 2005. 39, 84

[178] E. ROHMER, S. P. N. SINGH, AND M. FREESE. **V-REP: A versatile and scalable robot simulation framework**. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, Nov 2013. 39

[179] ROMAIN BRETTE, MICHELLE RUDOLPH, TED CARNEVALE, MICHAEL HINES, DAVID BEEMAN, JAMES M BOWER, MARKUS DIESMANN, ABIGAIL MORRISON, PHILIP H GOODMAN, FREDERICK C HARRIS, ET AL. **Simulation of networks of spiking neurons: a review of tools and strategies**. *Journal of computational neuroscience*, **23**(3):349–398, 2007. 39

[180] TREVOR BEKOLAY, JAMES BERGSTRA, ERIC HUNSBERGER, TRAVIS DEWOLF, TERRENCE STEWART, DANIEL RASMUSSEN, XUAN CHOO, AARON VOELKER, AND CHRIS ELIASMITH. **Nengo: a Python tool for building large-scale functional brain models**. *Frontiers in Neuroinformatics*, **7**:48, 2014. 39

[181] AARON STARANOWICZ AND GIAN LUCA MARIOTTINI. **A Survey and Comparison of Commercial and Open-source Robotic Simulator Software**. In *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '11, pages 56:1–56:8, New York, NY, USA, 2011. ACM. 39

[182] A. HARRIS AND J. M. CONRAD. **Survey of popular robotics simulators, frameworks, and toolkits**. In *2011 Proceedings of IEEE Southeastcon*, pages 243–249, March 2011. 39

[183] D GAMEZ, A K FIDJELAND, AND E LAZDINS. **iSpike: a spiking neural interface for the iCub robot**. *Bioinspiration & Biomimetics*, **7**(2):025008, 2012. 39, 41

[184] THOMAS VOEGTLIN. **CLONES : a closed-loop simulation framework for body, muscles and neurons**. *BMC Neuroscience*, **12**(1):P363, Jul 2011. 39

[185] DAN GOODMAN AND ROMAIN BRETTE. **The Brian simulator**. *Frontiers in Neuroscience*, **3**:26, 2009. 39

[186] JÉRÉMIE ALLARD, STÉPHANE COTIN, FRANÇOIS FAURE, PIERRE-JEAN BENSOUSSAN, FRANÇOIS POYER, CHRISTIAN DURIEZ, HERVÉ DELINGETTE, AND LAURENT GRISONI. **Sofa-an open source framework for medical simulation**. In *MMVR 15-Medicine Meets Virtual Reality*, **125**, pages 13–18. IOP Press, 2007. 39

[187] DAVID COFER, GENNADY CYMBALYUK, JAMES REID, YING ZHU, WILLIAM J. HEITLER, AND DONALD H. EDWARDS. **AnimatLab: A 3D graphics environment for neuromechanical simulations**. *Journal of Neuroscience Methods*, **187**(2):280 – 288, 2010. 39, 41

[188] EGIDIO FALOTICO, LORENZO VANNUCCI, ALESSANDRO AMBROSANO, UGO ALBANESE, STEFAN ULBRICH, JUAN 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 MARTÍNEZ-CAÑADA, EDUARDO ROS, PATRICK MAIER, SANDRO WEBER, MANUEL 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**. *Frontiers in Neurorobotics*, **11**:2, 2017. 39, 41, 82, 85

[189] J OBERTS AND S SANDERS. **Brain-inspired intelligent robotics: The intersection of robotics and neuroscience**. *Science/AAAS*, pages 1–50, 2016. 39

[190] C. RICHTER, S. JENTZSCH, R. HOSTETTLER, J. A. GARRIDO, E. ROS, A. KNOLL, F. ROHRBEIN, P. VAN DER SMAGT, AND J. CONRADT. **Musculoskeletal Robots: Scalability in Neural Control**. *IEEE Robotics Automation Magazine*, **23**(4):128–137, Dec 2016. 41

[191] G. S. CHIRIKJIAN AND J. W. BURDICK. **The kinematics of hyper-redundant robot locomotion**. *IEEE Transactions on Robotics and Automation*, **11**(6):781–793, Dec 1995. 42

[192] PÅL LILJEBÄCK, KRISTIN YTTERSTAD PETTERSEN, ØYVIND STAVDAHL, AND JAN TOMMY GRAVDAHL. **A review on modelling, implementation, and control of snake robots**. *Robotics and Autonomous Systems*, **60**(1):29–40, 2012. 42

[193] AKSEL ANDREAS TRANSETH, KRISTIN YTTERSTAD PETTERSEN, AND PÅL LILJEBÄCK. **A survey on snake robot modeling and locomotion**. *Robotica*, **27**(7):999–1015, 2009. 42

[194] F. SANFILIPPO, J. AZPIAZU, G. MARAFIOTI, A. A. TRANSETH, Ø. STAVDAHL, AND P. LILJEBÄCK. **A review on perception-driven obstacle-aided locomotion for snake robots**. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–7, Nov 2016. 42

[195] L. PAEZ AND K. MELO. **A preliminary review on metrics for modular snake robots locomotion**. In *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent*, pages 539–545, June 2014. 42

[196] S. MA. **Analysis of snake movement forms for realization of snake-like robots**. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, **4**, pages 3007–3013 vol.4, 1999. 42, 44, 45, 49

[197] SHIGEO HIROSE AND HIROYA YAMADA. **Snake-like robots [tutorial]**. *IEEE Robotics & Automation Magazine*, **16**(1):88–98, 2009. 42

[198] JUAN GONZÁLEZ GÓMEZ. **Modular robotics and locomotion: application to limbless robots**. *Pdd. Universidad Autonoma de Madrid. Madrid*, 2008. 42

[199] MATTHEW TESCH, KEVIN LIPKIN, ISAAC BROWN, ROSS HATTON, AARON PECK, JUSTINE REMBISZ, AND HOWIE CHOSET. **Parameterized and scripted gaits for modular snake robots**. *Advanced Robotics*, **23**(9):1131–1158, 2009. 42, 45, 46

[200] K. MELO AND L. PAEZ. **Experimental determination of control parameter intervals for repeatable gaits in modular snake robots**. In *2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014)*, pages 1–7, Oct 2014. 42, 45

[201] X. NIU, J. XU, Q. REN, AND Q. WANG. **Locomotion Learning for an Anguilliform Robotic Fish Using Central Pattern Generator Approach**. *IEEE Transactions on Industrial Electronics*, **61**(9):4780–4787, Sept 2014. 42

[202] JUNZHI YU, RUI DING, QINGHAI YANG, MIN TAN, AND JIANWEI ZHANG. **Amphibious Pattern Design of a Robotic Fish with Wheel-propeller-fin Mechanisms**. *Journal of Field Robotics*, **30**(5):702–716, 2013. 42

[203] KEEHONG SEO, SOON-JO CHUNG, AND JEAN-JACQUES E SLOTINE. **CPG-based control of a turtle-like underwater vehicle**. *Autonomous Robots*, **28**(3):247–269, 2010. 42

[204] AUKE JAN IJSPEERT, ALESSANDRO CRESPI, DIMITRI RYCZKO, AND JEAN-MARIE CABELGUEN. **From swimming to walking with a salamander robot driven by a spinal cord model**. *science*, **315**(5817):1416–1420, 2007. 42, 43

[205] SHIGEO HIROSE. *Biologically inspired robots: snake-like locomotors and manipulators*, **1093**. Oxford university press Oxford, 1993. 42

[206] DAVID L HU, JASMINE NIRODY, TERRI SCOTT, AND MICHAEL J SHELLEY. **The mechanics of slithering locomotion**. *Proceedings of the National Academy of Sciences*, **106**(25):10081–10085, 2009. 42

[207] M. MORI AND S. HIROSE. **Three-dimensional serpentine motion and lateral rolling by active cord mechanism ACM-R3**. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, **1**, pages 829–834 vol.1, 2002. 43

[208] SHUNTAROU CHIGISAKI, MAKOTO MORI, HIROYA YAMADA, AND SHIGEO HIROSE. **Design and control of amphibious Snake-like Robot ACM-R5**. *Nippon Kikai Gakkai Robotikusu, Mekatoronikusu Koenkai Koen Ronbunshu (CD-ROM)*, 2005. 43

[209] T. OHASHI, H. YAMADA, AND S. HIROSE. **Loop forming snake-like robot ACM-R7 and its Serpenoid Oval control**. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 413–418, Oct 2010. 43

[210] C. WRIGHT, A. BUCHAN, B. BROWN, J. GEIST, M. SCHWERIN, D. ROLLINSON, M. TESCH, AND H. CHOSET. **Design and architecture of the unified modular snake robot**. In *2012 IEEE International Conference on Robotics and Automation*, pages 4347–4354, May 2012. 43, 44

[211] DAVID ROLLINSON, STEVEN FORD, BEN BROWN, AND HOWIE CHOSET. **Design and modeling of a series elastic element for snake robots**. In *ASME 2013 Dynamic Systems and Control Conference*, pages V001T08A002–V001T08A002. American Society of Mechanical Engineers, 2013. 44

[212] M. YIM, D. G. DUFF, AND K. D. ROUFAS. **PolyBot: a modular reconfigurable robot**. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, **1**, pages 514–520 vol.1, 2000. 44

[213] M. YIM, W. M. SHEN, B. SALEMI, D. RUS, M. MOLL, H. LIPSON, E. KLAVINS, AND G. S. CHIRIKJIAN. **Modular Self-Reconfigurable Robot Systems [Grand Challenges of Robotics]**. *IEEE Robotics Automation Magazine*, **14**(1):43–52, March 2007. 44

[214] C. WRIGHT, A. JOHNSON, A. PECK, Z. MCCORD, A. NAAKTGEBOREN, P. GIANFORTONI, M. GONZALEZ-RIVERO, R. HATTON, AND H. CHOSET. **Design of a modular snake robot**. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2609–2614, Oct 2007. 44

[215] SHUGEN MA, Y. OHMAMEUDA, K. INOUE, AND BIN LI. **Control of a 3-dimensional snake-like robot**. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, **2**, pages 2067–2072 vol.2, Sept 2003. 45

[216] L. PFOTZER, S. KLEMM, A. ROENNAU, J.M. ZÖLLNER, AND R. DILLMANN. **Autonomous navigation for reconfigurable snake-like robots in challenging, unknown environments**. *Robotics and Autonomous Systems*, **89**:123 – 135, 2017. 45, 46

[217] M. TANAKA AND K. TANAKA. **Control of a Snake Robot for Ascending and Descending Steps**. *IEEE Transactions on Robotics*, **31**(2):511–520, April 2015. 45

[218] HENRY C. ASTLEY, CHAOHUI GONG, JIN DAI, MATTHEW TRAVERS, MIGUEL M. SERRANO, PATRICIO A. VELA, HOWIE CHOSET, JOSEPH R. MENDELSON, DAVID L. HU, AND DANIEL I. GOLDMAN. **Modulation of orthogonal body waves enables high maneuverability in sidewinding locomotion**. *Proceedings of the National Academy of Sciences*, **112**(19):6200–6205, 2015. 45, 51

[219] U.S. FISH HILLEBRAND STEVE AND VIA WIKIMEDIA COMMONS WILDLIFE SERVICE [PUBLIC DOMAIN]. **Eastern garter snake slithers through a muddy area - Wikipedia**. `https://commons.wikimedia.org/wiki/File:Eastern_garter_snake_slithers_through_a_muddy_area.jpg`. 46

[220] NORZALILAH MOHAMAD NOR AND SHUGEN MA. **Smooth transition for CPG-based body shape control of a snake-like robot**. *Bioinspiration & Biomimetics*, **9**(1):016003, 2014. 46

[221] ROLLINSON DAVID AND CHOSET HOWIE. **Pipe Network Locomotion with a Snake Robot**. *Journal of Field Robotics*, **33**(3):322–336, 2014. 47

[222] WEIKUN ZHEN, C. GONG, AND H. CHOSET. **Modeling rolling gaits of a snake robot**. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015. 47

[223] R. L. HATTON AND H. CHOSET. **Sidewinding on slopes**. In *2010 IEEE International Conference on Robotics and Automation*, pages 691–696, 2010. 47

[224] CHAOHUI GONG, R. L. HATTON, AND H. CHOSET. **Conical sidewinding**. In *2012 IEEE International Conference on Robotics and Automation*, pages 4222–4227, May 2012. 47

[225] C. GONG, M. J. TRAVERS, XIAOZHOU FU, AND H. CHOSET. **Extended gait equation for sidewinding**. In *2013 IEEE International Conference on Robotics and Automation*, pages 5162–5167, May 2013. 47

[226] CHAOHUI GONG, MATTHEW J. TRAVERS, HENRY C. ASTLEY, LU LI, JOSEPH R. MENDELSON, DANIEL I. GOLDMAN, AND HOWIE CHOSET. **Kinematic gait synthesis for snake robots**. *The International Journal of Robotics Research*, **35**(1-3):100–113, 2016. 47

[227] JAMES GRAY. **The mechanism of locomotion in snakes**. *Journal of experimental biology*, **23**(2):101–120, 1946. 49, 52

[228] S HIROSE. **Biologically Inspired Robots: Snake-Like Locomotors and Manipulators,(1993)**. 49

[229] N. KOPELL AND G.B. ERMENTROUT. **Coupled oscillators and the design of central pattern generators**. *Mathematical Biosciences*, **90**(1):87 – 109, 1988. 61

[230] B. A. PEARLMUTTER. **Gradient calculations for dynamic recurrent neural networks: a survey**. *IEEE Transactions on Neural Networks*, **6**(5):1212–1228, Sep 1995. 61

[231] AUKE JAN IJSPEERT, ALESSANDRO CRESPI, DIMITRI RYCZKO, AND JEAN-MARIE CABELGUEN. **From Swimming to Walking with a Salamander Robot Driven by a Spinal Cord Model**. *Science*, **315**(5817):1416–1420, 2007. 64, 66

[232] NORZALILAH MOHAMAD NOR AND SHUGEN MA. **Smooth transition for CPG-based body shape control of a snake-like robot**. *Bioinspiration & Biomimetics*, **9**(1):016003, 2014. 66

[233] KEEHONG SEO, SOON-JO CHUNG, AND JEAN-JACQUES E. SLOTINE. **CPG-based control of a turtle-like underwater vehicle**. *Autonomous Robots*, **28**(3):247–269, Apr 2010. 66

[234] H. YU, W. GUO, J. DENG, M. LI, AND H. CAI. **A CPG-based locomotion control architecture for hexapod robot**. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5615–5621, Nov 2013. 66

[235] MARC-OLIVER GEWALTIG AND MARKUS DIESMANN. **NEST (NEural Simulation Tool)**. *Scholarpedia*, **2**(4):1430, 2007. 75, 83

[236] NATALIA CAPORALE AND YANG DAN. **Spike Timing–Dependent Plasticity: A Hebbian Learning Rule**. *Annual Review of Neuroscience*, **31**(1):25–46, 2008. PMID: 18275283. 75

[237] HENRY MARKRAM, JOACHIM LÜBKE, MICHAEL FROTSCHER, AND BERT SAKMANN. **Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs**. *Science*, **275**(5297):213–215, 1997. 75

[238] GUO-QIANG BI AND MU-MING POO. **Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type**. *Journal of Neuroscience*, **18**(24):10464–10472, 1998. 75

[239] LARRY F ABBOTT AND SACHA B NELSON. **Synaptic plasticity: taming the beast**. *Nature neuroscience*, **3**(11s):1178, 2000. 76

[240] VINCENT JACOB, DANIEL J. BRASIER, IRINA ERCHOVA, DAN FELDMAN, AND DANIEL E. SHULZ. **Spike Timing-Dependent Synaptic Depression in the In Vivo Barrel Cortex of the Rat**. *Journal of Neuroscience*, **27**(6):1271–1284, 2007. 76

[241] MARCO LEHMANN, HE XU, VASILIKI LIAKONI, MICHAEL HERZOG, WULFRAM GERSTNER, AND KERSTIN PREUSCHOFF. **Evidence for eligibility traces in human learning**. *arXiv preprint arXiv:1707.04192*, 2017. 76

[242] EUGENE M. IZHIKEVICH. **Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling**. *Cerebral Cortex*, **17**(10):2443–2452, 2007. 77

[243] CLARK LEONARD HULL. **Principles of behavior: An introduction to behavior theory.** 1943. 77

[244] CE CARR AND M KONISHI. **A circuit for detection of interaural time differences in the brain stem of the barn owl**. *Journal of Neuroscience*, **10**(10):3227–3246, 1990. 78

[245] SAJJAD SEIFOZZAKERINI, WEI-YUN YAU, BO ZHAO, AND KEZHI MAO. **Event-Based Hough Transform in a Spiking Neural Network for Multiple Line Detection and Tracking Using a Dynamic Vision Sensor.** In *BMVC*, 2016. 80

[246] J. KAISER, J. C. V. TIECK, C. HUBSCHNEIDER, P. WOLF, M. WEBER, M. HOFF, A. FRIEDRICH, K. WOJTASIK, A. ROENNAU, R. KOHLHAAS, R. DILLMANN, AND J. M. ZÖLLNER. **Towards a framework for end-to-end control of a simulated vehicle with spiking neural networks**. In *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 127–134, Dec 2016. 81

[247] ANDREW P DAVISON, DANIEL BRÜDERLE, JOCHEN M EPPLER, JENS KREMKOW, EILIF MULLER, DEJAN PECEVSKI, LAURENT PERRINET, AND PIERRE YGER. **PyNN: a common interface for neuronal network simulators**. *Frontiers in neuroinformatics*, **2**:11, 2009. 83

[248] ERWIN COUMANS. **Bullet physics engine**. *Open Source Software: http://bulletphysics. org*, **1**:3, 2010. 84

[249] 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*, **3**, page 5. Kobe, Japan, 2009. 84

[250] AARON CLAUSET, CRISTOPHER MOORE, AND MARK EJ NEWMAN. **Hierarchical structure and the prediction of missing links in networks**. *Nature*, **453**(7191):98, 2008. 87

[251] J. S. ALBUS, R. LUMIA, AND H. MCCAIN. **Hierarchical control of intelligent machines applied to space station telerobots**. *IEEE Transactions on Aerospace and Electronic Systems*, **24**(5):535–541, Sep 1988. 87

[252] MALTE SCHILLING, THIERRY HOINVILLE, JOSEF SCHMITZ, AND HOLK CRUSE. **Walknet, a bio-inspired controller for hexapod walking**. *Biological Cybernetics*, **107**(4):397–419, Aug 2013. 87

[253] S. NAGATA, M. SEKIGUCHI, AND K. ASAKAWA. **Mobile robot control by a structured hierarchical neural network**. *IEEE Control Systems Magazine*, **10**(3):69–76, April 1990. 87

[254] ANDRES ESPINAL, HORACIO ROSTRO-GONZALEZ, MARTIN CARPIO, ERICK I. GUERRA-HERNANDEZ, MANUEL ORNELAS-RODRIGUEZ, AND MARCO SOTELO-FIGUEROA. **Design of Spiking Central Pattern Generators for Multiple Locomotion Gaits in Hexapod Robots by Christiansen Grammar Evolution**. *Frontiers in Neurorobotics*, **10**:6, 2016. 88

[255] PIERRE A. GUERTIN. **The mammalian central pattern generator for locomotion**. *Brain Research Reviews*, **62**(1):45 – 56, 2009. 88

[256] JOHN NASSOUR, PATRICK HÉNAFF, FETHI BENOUEZDOU, AND GORDON CHENG. **Multi-layered multi-pattern CPG for adaptive locomotion of humanoid robots**. *Biological cybernetics*, **108**(3):291–303, 2014. 88

[257] T. REIL AND P. HUSBANDS. **Evolution of central pattern generators for bipedal walking in a real-time physics environment**. *IEEE Transactions on Evolutionary Computation*, **6**(2):159–168, April 2002. 88

[258] ACHIM J LILIENTHAL AND TOM DUCKETT. **Experimental analysis of smelling Braitenberg vehicles**. In *IEEE international conference on advanced robotics (ICAR 2003), Coimbra, Portugal, June 30-July 3, 2003*, **1**, pages 375–380. Coimbra, University, 2003. 103

[259] SEHOON HA, JOOHYUNG KIM, AND KATSU YAMANE. **Automated Deep Reinforcement Learning Environment for Hardware of a Modular Legged Robot**. In *International Conference on Ubiquitous Robots*, 2018. 126

[260] 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 *OSDI*, **16**, pages 265–283, 2016. 127

[261] RAMI AL-RFOU, GUILLAUME ALAIN, AMJAD ALMAHAIRI, CHRISTOF ANGERMUELLER, DZMITRY BAHDANAU, NICOLAS BALLAS, FRÉDÉRIC BASTIEN, JUSTIN BAYER, ANATOLY BELIKOV, ALEXANDER BELOPOLSKY, ET AL. **Theano: A Python framework for fast computation of mathematical expressions**. *arXiv preprint*, 2016. 127

[262] RONAN COLLOBERT, KORAY KAVUKCUOGLU, AND CLÉMENT FARABET. **Torch7: A matlab-like environment for machine learning**. In *BigLearn, NIPS workshop*, number EPFL-CONF-192376, 2011. 127

[263] TIMOTHY P LILLICRAP, JONATHAN J HUNT, ALEXANDER PRITZEL, NICOLAS HEESS, TOM EREZ, YUVAL TASSA, DAVID SILVER, AND DAAN WIERSTRA. **Continuous control with deep reinforcement learning**. *arXiv preprint arXiv:1509.02971*, 2015. 127

[264] FLORIAN WALTER, FLORIAN RÖHRBEIN, AND ALOIS KNOLL. **Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks**. *Neural Networks*, **72**:152 – 167, 2015. Neurobiologically Inspired Robotics: Enhanced Autonomy through Neuromorphic Cognition. 127