UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Central pattern generator model using spiking neural networks

Author:
Yuriy PRYYMA

Supervisor:
Sergiy YAKOVENKO

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2021

Declaration of Authorship

I, Yuriy PRYYMA, declare that this thesis titled, "Central pattern generator model using spiking neural networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Central pattern generator model using spiking neural networks

by Yuriy PRYYMA

Abstract

Locomotor control involves dynamic mechanical and neural interactions that are essential for survival. The neural locomotor pathways contain the central pattern generator (CPG), a network of neurons embedded into the spinal cord and generating dynamic output for walking and running. Even though there are multiple formulations of the CPG, from coupled oscillators to complex networks of Hodgkin-Huxley neurons, the optimal choice of model implementation depends on its application. The choice of a formulation is often described as the trade-off between complexity and the level of details in the model's function. However, the advantages between different formulations have not been established. Recently, the spiking neural networks (SNN) have gained popularity as a biological analog for neural dynamics that uses methodology developed for artificial neural networks. This formulation uses spiking frequency instead of rate signals to accomplish dynamic computations with the integrate-and-fire neurons.

In this study[1], we aimed to create the framework for comparing a versatile CPG rate model and its implementation with the model build with SNN. We used a neuromorphic software package (Nengo) to develop and validate a bilateral CPG model's structural and functional details based on the half-center oscillators. The spiking model shows similar precision for calculating the empirical phase-duration characteristics of gait in cats as the rate model, and it also reproduces the linear relationship between the CPG input and the empirical limb speed of forward progression. While the phase characteristic was used to optimize neural dynamics, the input relationship with the limb speed is the product of the model structure. Furthermore, the spiking model has increased tolerance to temporal noise, and it can withstand some structural damage. The spiking and rate models require further comparative analysis. Overall, the development of adaptable spiking models could help integrate the biomimetic components within the control systems for assistive robotics and electrical stimulation devices to rehabilitate locomotion after central and peripheral injuries.

[1] https://github.com/YuriyPryyma/cpg_nengo

Acknowledgements

I would like first to express my appreciation to Dr. Sergiy Yakovenko the director of neural engineering laboratory at West Virginia University for providing supervision and consultations and for sharing his experience in the domain. Without him, this work would be impossible. I would also like to thank Ukrainian Catholic University and all teachers and lecturers for creating such a great Master's program. Separately, I want to thank Oleksii Molchanovskyi for the support in my two-year graduate studies. Additionally, I give my gratitude to the Nengo community for providing helpful resources and my friends how reviewed final work.

Contents

List of Figures

List of Abbreviations

CPG	**C**entral **P**attern **g**enerator
SNN	**S**piking **N**eura **N**etwork
GC	**G**ait **c**ycle
NEF	**N**eural **E**ngineering **F**ramework
HCO	**H**alf-**c**enter **o**scillators
ANN	**A**rtificial **N**eural **N**etwork
HR	**H**indmarsh-**R**ose
HH	**H**odgkin–**H**uxley

Chapter 1

Introduction

1.0.1 Motivation

One of the critical functions of animals is the ability to move in a variety of environments. As a result, different types of CPGs have evolved to control rhythmic behaviors. Locomotion is an essential behavior that shaped both neural pathways and animal morphologies. CPG is a neural structure within the lumbar enlargement of mammals, e.g., cats and humans, consisting mainly of interneurons; however, even alpha motoneurons were shown to have intrinsic rhythmogenic properties indicating their contribution to the CPG activity. The CPG is often modeled as a system of differential equations of Hodgkin–Huxley that describes ion flows inside a neuron. These models are often difficult to validate due to high parametric space. Other approaches focus on the formulation of state dynamics using generalizations to reduce complexity. Both models can transform a low dimensional input to generate a pattern of swing and stance phases to achieve locomotion. Although biophysical models accurately describe neuron physics and dynamics, it is hard to extend networks to more functions and structural elements due to high parametric space. Modern approaches like Artificial Neural Network (ANN) showed good performance on various cognitive tasks, yet the formulations are only generally described by biological dynamics. A new type of ANNs called spiking neural networks(SNNs) has been developed to use the integrate-and-fire concept from computational neuroscience and improve biological relevance. This work tries to explore neurological models of animal CPGs using spiking neural networks. Neural Engineering Framework(NEF) used to structure SNNs describes rules for information transition and representation allows organization of SNNs in a more abstract and scalable way. NEF would help build an accurate and scalable CPG model, preserving the benefits of underlying spiking neuron networks. Also, models build on top spiking neural networks produce similar biases as biological systems do (Rasmussen and Eliasmith, 2014). Therefore, solving dynamical tasks using SNNs could help us better understand the neural control of behaviors in humans. In this work, our task is to describe the rhythmogenesis function using a theoretical model that could be further used in clinical applications, for example, for the control of intraspinal stimulation to restore rhythmogenesis after a spinal cord injury. Like models of other neural structures, including single neurons, the mathematical formulation is typically defined by the scientific question. In this study, we examined the findings

based on the dynamics of the rate model using the model with the formulation based on the spiking dynamics. The variation of a mathematical formulation is one of the requirements within the field of multi-scale modeling. The new model that shows equivalent dynamic would allow us to examine further costs and benefits of spiking and rate model formulations. One of the possible motivations for CPG implementation using SNN could be its power efficiency and performance.

1.0.2 Goals

- Provide an overview on work related to CPG model formulated with spiking neuron networks.
- Implement a bilateral half-center oscillators model of a CPG using SNNs in Nengo framework.
- Optimize model to tune to locomotion patterns and test the velocity hypothesis supported by the rate models.

1.0.3 Thesis structure

The thesis presents related work in CPG and SNN domains in chapter 2. In the next chapter 3 we describe the properties of locomotion data used in the study. Then, chapter 4 focused on the baseline CPG model and how we implemented it in Nengo. Next, the process of model optimization and later its validation is described in chapter 5. Finally, in chapter 6 we summarize our results and discuss model limitations and future work.

Chapter 2

Related work

In this chapter, we divide related work into two parts. In the first section, we want to look at the current and common approaches for building CPG models. Then, in the second section, we want to focus on the evolution of a spiking neural network and its pros and cons.

2.1 Central pattern generator (CPG)

Central Pattern Generator (CPG) controls vertebrate locomotion (Dimitrijevic, Gerasimenko, and Pinter, 1998), and it can automatically generate complex control signals to coordinate muscles during rhythmic movements, such as walking, running, swimming, and flying. CPGs are neural networks capable of producing coordinated rhythmic activity patterns without any rhythmic inputs from sensory feedback or higher control centers. In general, locomotion is organized such that CPGs are responsible for converting commands from the descending and sensory feedback pathway to low-level patterns of flexor and extensor movements (Ijspeert, 2008).

The coordination of locomotion is one of the main challenges in moving robots. There are two main approaches for the design of locomotion control systems, such as kinematic and dynamic mathematical models and biologically inspired approaches (Ijspeert, 2008). The first one uses joint speed and positions in advance, based on a mathematical model using observation for the environment and robot dynamics (Fukuoka, Kimura, and Cohen, 2003). The second approach mimics the center of animals' locomotion with a CPG model to produce walking patterns. The first approach relies on a complex model and hard to adapt. Since biologists have made significant advances in understanding animal locomotion, recent robotics models showed promising results with different leg configurations (Espinal et al., 2016, Endo et al., 2008)

There are several approaches to model CPG. The first one is detailed biophysical models, which are usually based on a system of differential equations that describe how ion channels influence membrane potentials and the generation of action potentials (Hellgren, Grillner, and Lansner, 1992, Traven et al., 1993). One of the most popular is Hodgkin–Huxley model (also termed H–H model) developed by Hodgkin and Huxley, 1952. However, it is very complicated and computationally expensive for computer simulations

involving large populations of neurons. Because of it, most models concentrate on the detailed dynamics of small circuits. The second one uses more abstracted versions of neurons. One of the earliest models of an abstracted neuron is the integrate-and-fire model (Ijspeert, 2001, Williams, 1992). These models exploring how a rhythmic activity is generated by network properties (e.g., half-center networks) and how interneuron connections synchronize different oscillatory neural circuits. The disadvantage of this model is that it does not implement time-dependent memory present in natural neuron systems. The other approach is to represent CPG as a dynamical system of coupled, nonlinear oscillators (Yu et al., 2014, Collins and Richmond, 1994, Wang et al., 2019). As opposite to neural oscillators, nonlinear oscillators do not have clear biological meanings. Another option for big networks would be to use relatively simple phenomenological rate models of a neuron (Sterratt et al., 2009) where specifics of neural spiking approximated using the discharge rate of neural spiking.

The Hindmarsh-Rose (HR) neuron model, which simulates spike bursting of the membrane potential observed in experiments, is often used in spiking CPG models. The modified Hindmarsh-Rose model created by Selverston et al., 2000 was able to capture dynamics of lobster stomatogastric ganglion neurons from a central pattern generator. The authors also conclude that their model could be used to construct an analog electronic system that could work in real-time.

2.2 Spiking neural networks (SNN)

The human brain has remarkable properties such as analog computation, low power consumption, fast inference, event-driven processing, online learning, and massive parallelism. SNNs model architecture tries to mimic these properties. Because spike events are sparse and have high information content, we could significantly reduce the power consumption of parts of networks that do not receive signals (Stone, 2018). Similarly, human brains do not use all neurons simultaneously, but only regions needed for current tasks. This same advantage is maintained in hardware (Rueckauer et al., 2017, Pande et al., 2013). Thus, it is possible to create low-energy hardware based on the property that information is sparse in time and concentrated in spikes. It is one of the biggest advantages of SNNs.

Several models of spiking neurons and SNN have developed so far, e.g.: Spike Response Models (Gerstner and Kistler, 2002, Gerstner, 2001); models with simulated Hodgkin–Huxley formulations (Izhikevich, 2004). However, a lot of studies in a field of SNNs has been limited to very simple and shallow network architectures on relatively simple digit recognition datasets like MNIST (LeCun and Cortes, 2010), while only a few works report their performance on more complex standard vision datasets like CIFAR-10 (Krizhevsky, 2009). The multi-layer neural architecture in the primate's brain has inspired researchers to concentrate on the depth of ANNs instead of using shallow networks with many neurons. Theoretical and experimental results show

better performance of deep rather than wide structures, which inspired progressions in the direction of deep SNNs. Despite their recent success in applying SNNs to image processing tasks, the resulting accuracy is still lower than state-of-the-art DNNs models using similar CNN architecture. One could argue that datasets used for evaluation are more suitable for the DNNs model that work on frame-level, unlike the SNNs model that requires preprocessing frames to spike data.

Nevertheless, the biggest problem of deep SNNs is the training process. Because spikes signals are sparse and not differentiable, we can not apply backpropagation to train this model. There exist three ways for SNN learning: 1) unsupervised learning such as spike timing-dependent plasticity (STDP); 2) indirect supervised learning such as ANNs-to-SNNs conversion; 3) direct supervised learning such as gradient descent-based backpropagation. But right now most of them limited to very shallow structures (amount of network layer less than 4) or toy small datasets (e.g., MNIST, Iris). Only a small number of works tries direct training of deep SNNs due to their challenges. There is work in progress to develop practical learning algorithms and efficient programming frameworks (Wu et al., 2018).

Research of CPG models implemented as spiking neural networks(SNNs) in Nengo mainly focused on the robotics domain to create a locomotion model for different robots. The primary motivation for using SNNs is to reduce embedded devices' power consumption to run the robot. Some approaches use Christiansen Grammar Evolution to estimate the spiking neural network's weights and synaptic connections and deploy the FPGA model (Field Programmable Gate Array), which shows excellent performance gains (Rostro-Gonzalez et al., 2015). Reinforcement-based stochastic weight update could also be used to train SNNs that run on a lightweight raspberry pi (Lele et al., 2020). The authors' model converges to the desired bio-observed tripod in 70% of the cases, while in other cases, it converges to suboptimal gaits that can still enable the locomotion. However, some of the research focuses more on engendering and deployment of SNNs on specific hardware, for example SpiNNaker boards, and does not describe the model learning process (Cuevas-Arteaga et al., 2017, Gutierrez-Galan et al., 2020).

Chapter 3

Data description

In this chapter, we describe the nature of the locomotion data and its main characteristics. We are using these characteristics to generate synthetic data for training and validation. Below we provide a description of CPG data properties that helps to understand the main research questions.

3.1 Gait cycle

Animal locomotion is often described in terms of the gait cycle (Winter, 1984). A gait cycle starts when a limb contacts the ground and ends when the same limb touches the ground next time and involves driving the center of mass in the direction of motion. A single gait cycle is also known as a stride. The gait cycle is usually broken into two phases: stance and swing. During the stance phase a limb is in contact with ground, and during the swing phase it is in air. The swing to stance transition happens when a last limb part contacts ground. Running is when both limbs are in swing, which happens when swing phase is longer than stance phase. Also, the stance phase(support) could be father divided into:

- Single stance: only one foot is in contact with the ground.
- Double stance: both limbs are in contact with the ground.

3.2 Locomotion phases

Halbertsma, 1983 study analyzed different patterns of gait cycles of the intact cat. Nine adult male and female cats with varying weights were used to conduct experiments. The motion capture system and electromyograms were utilized to record cat locomotion on a treadmill. There were two belts on a treadmill to impose different speeds on left and right legs. The speed of a treadmill was changing to record locomotion with different speeds and cycle durations to derive statistical analysis of gait phases. The excitation of different muscles correlated with the movement in different joints. Here we present the insights from Halbertsma, 1983 study related to this work:

- All cycle durations and joint angles are adapted to different speeds to maintain coordination and no element from gait cycles remains constant.

- As we increase the speed of a treadmill, stride cycle duration decreases. The duration of a stance phase decreases proportionally more than the swing phase duration in the same stride. There is a linear relationship stance, swing and stride duration.
- During different types of locomotion from a cat's experiments like walking and trotting, strides from opposite limbs appears to be shifted by approximately half of a cycle. Even when two belts of a treadmill were moving with mismatched speed, the limbs still maintained coordination.
- Cats experiments reveal some of the properties of locomotion system which neuron networks and feedback mechanisms must have.

The example of the linear relationship of phase duration is visualized in Figure 3.1. Circles represent duration of stance and swing phases. The swing phase duration does not change much during cycle duration changes, and the most significant portion of change is in the stance phase meaning that animals move faster by mainly decreasing their contact with the ground.

The resulting linear relationship is used to generate synthetic data for phase duration where T_{su} represents the duration of a stance cycle duration, T_{sw} for swing, and T_c for stride cycle duration.

$$T_{su} = -0.168 + 0.9062 * T_c \tag{3.1}$$

$$T_{sw} = 0.168 + 0.0938 * T_c \tag{3.2}$$

Valid stride cycle duration ranges from empirical data is $[.57, 1.91]$ seconds

There is also a relationship between cycle duration and animal velocity from Goslow, Reinking, and Stuart, 1973. Again, we could represent it as an as power function:

$$T_c = 0.5445 * V^{-0.5925}$$

Where V is animal velocity, and T_c is stride duration.

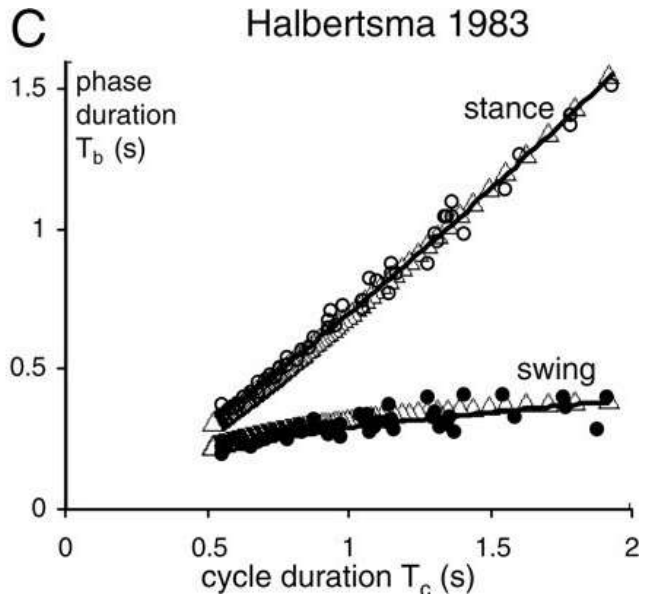


FIGURE 3.1: Simulated and experimental phase duration characteristics during cat locomotion. Filled circles: flexor burst or swing-phase durations, open circles: extensor burst or stance-phase durations. The straight line fitted to the data. Source: Yakovenko et al., 2005

Chapter 4

Methodology

The project goal is to formulate two limb phase transitions during locomotion using SNNs. One of the widely used approaches (Schlichter et al., 2010) is to model this CPG as a coupled half-center oscillators (HCO). HCO usually consists of two neurons or two groups of neurons with strong mutual inhibit connections. These neuron groups are phase-locked, meaning that only one group is active while the other is suppressed. Each group of neurons has no rhythmogenic ability individually, but when combined, could produce rhythmic output patterns. For two limbs dynamics formulation, we then need two coupled half-center oscillators. Studying how coupled half-center oscillators produces coordinated locomotion is an important issue in neuroscience.

For two limbs CPG with spiking neurons, we use dynamics formulated for rate model of CPG with HCOs described in subsection 4.1. The goal of this section is to implement CPG rate model states transitions using SNNs. In subsection 4.2.1 we describe the main model development approaches in Nengo simulation environment. The last subsection describes details of oscillators state representation and transitions in Nengo.

4.1 Rate model of CPG

The reciprocal half-center oscillator(HCO) model implemented by (Yakovenko, 2011, Yakovenko, Sobinov, and Gritsenko, 2018b) is a baseline CPG implementation for this project, and it uses a firing-rate neuron model. The firing-rate model allows the construction of neuron-like network units with outputs consisting of firing rates rather than action potentials. This approach helps with computational and interpretational challenges but lacks biological details. Because we model two limbs, there two pairs of single oscillator models (Prochazka and Yakovenko, 2007) for phase dominance consisting of two firing-rate neurons model. Each group of neurons represents spiking rates of presumed neuron populations and has a single scalar value. Four scalar values x_1, x_2, x_3, x_4 represent our model state(see Fig. 4.1). States values do not represent muscle activities or have any physical meaning but represent leaky integrator saturation value. For each limb, only one integrator is active, and the second one is suppressed. When the value of an integrator reaches the threshold, it resets, and model switches activate integrator. This process

happens independently for each limb. Switching of activate integrator represents locomotion phase transition. The swing phase for the first limb is active when $x_1 > 0$ and similarly for the second limb if $x_3 > 0$. In the same way, x_2, x_4 represents stance phases.

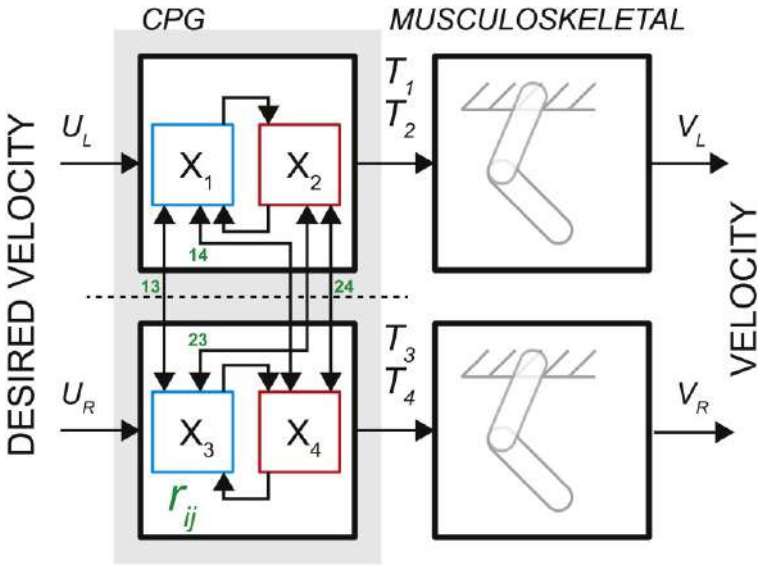


FIGURE 4.1: Schematic representation of CPG model. The model receives low-dimensional input actuating two limbs (U_L and U_R). The output pattern corresponds to the forward progressions with turning. Source: Yakovenko, Sobinov, and Gritsenko, 2018a

The dynamics of integrator values are represented as a system of differential equations shown below.

$$\dot{x} = x_0 + G_u u + G_x^{\mathrm{UL}} x + G_x^{\mathrm{BL}}(1 - x)\Big|_{x>0} \tag{4.1}$$

$$G_x^{\mathrm{UL}} = I r_{\mathrm{leak}} \tag{4.2}$$

$$G_x^{\mathrm{BL}} = \begin{bmatrix} 0 & 0 & r_{13} & r_{14} \\ 0 & 0 & r_{23} & r_{44} \\ r_{13} & r_{14} & 0 & 0 \\ r_{23} & r_{24} & 0 & 0 \end{bmatrix} \tag{4.3}$$

u is a model input parameter for desired limb speed. G_u matrix represent influence of the u parameter on neuron exitation, G_x matrix shows strength of connection between neurons with respect to neuron state x and bias x_0, I is the identity matrix, r_{leak} is the constant that determines intrinsic state-dependent feedback. x represent two pairs of integrators for swing and stance phases $(x_1, x_2, x_3, x_4)^T$. $[x_1, x_3]$ responsible for swing phase and $[x_2, x_4]$ for stance. Neurons' states could take positive values starting from 0 and going to 1, at which it resents back to 0. As a result, the model has nine parameters describing dynamics.

The model of a CPG was simulated for 80 seconds, and its state update was approximated using Runge–Kutta (fourth-order) method. Error function consists of 3 parts: phase-duration loss, cycle ranges loss and limb symmetry loss. Phase-duration loss compute duration for swing and stance phase for two limbs for different speeds and then calculated how close it is to empirical data (Halbertsma, 1983). Cycle ranges loss checks that minimum and maximum stride cycle duration are in the range of empirical data (Halbertsma, 1983). Finally, we compute the activation peaks of model spikes by measuring the distance between the maximum excitement of neurons. Root mean square value (RMS) was calculated for distances between ground truth phase duration characteristic and received from the model. Limbs symmetry loss checks that swing phases are not intersecting, and the stance phase of one limb fully contains the swing phase of the opposite leg. Error minimization was treated as backbox optimization, and model parameters searched using Nelder–Mead nonlinear error minimization.

Resulting model could accurately reproduce experimental results (R^2_{swing} and R^2_{stance} are 0.915 and 0.999, respectively).

4.2 Background information

4.2.1 Nengo simulation environment

There are many approaches for implementing cognitive processes like vision, speech generation, including cognitive architectures (Anderson et al., 2004) and machine learning (Hinton, 2006). Nengo relies on a Neural Engineering Framework(NEF) proposed by Eliasmith, 2005. The NEF is the approach of building large-scale artificial neuro networks with cognitive behavior that utilize single neuron models. There are several successful implementations using NEF and Nengo of complex cognitive systems like memory, decision making, and list memory. The most prominent model based on NEF and Nengo called spawn (Eliasmith, 2013) could perform up to eight cognitive tasks without any retraining. For example, it could memorize a list of input numbers and then use its motor system to draw them. Fascinatingly models based on NEF produce results that well align with experimental data like behavioral errors, age-related cognitive decline (Rasmussen and Eliasmith, 2014), and single-cell activity (Stewart, Bekolay, and Eliasmith, 2012)

Neural Engineering Framework principles:

- Representation: Vector of real values is represented by the population of neurons using encoding to and decoding from inner population spiking patterns. Manipulation neurons in terms of number vectors allow convenient use of mathematics to solve problems. During encoding, the input signal excites neurons based on their threshold value and then fires at a rate proportional to the input signal. The neuron threshold value and fire pattern represent the neuron tuning curve. The combination of many tuning curves allows an accurate approximation of the input signal. During the decoding process, neurons' fire pattern first

filtered using an exponentially decaying filter due to the postsynaptic current of neurons. Then the weighted sum of the neuron's activation is minimized to match the desired output, and spikes decoding weights are calculated by solving a least-squares minimization problem.

- Transformation: Biological neurons communicate current thought synapses using neurotransmitters. Many factors influence the power of connection. Nengo neurons communicate information from source to target neurons using decoding and encoding weights from respecting population and controlling the power of a connection using scalar factor. As mentioned in the representation part, least-squares minimization used to compute decoding weights. By default, each connection learns an identity function, meaning that it reconstructs the input signal. However, decoding weights could be trained to represent any function. For example, let us take input signal $sin(x)$ for a group of neurons of 100 neurons. The first signal will be encoded using 100 tuning curves and represented as spikes of neurons for function $sin(x)$. Then if we want to learn function x^2, we sample points from a range of values representing the neurons population and evaluate the function x^2. The model uses resulting points to train decoding weights. As a result, we trained the output ensemble to represent function $sin(x)^2$

- Dynamics: Dynamics could play a fundamental role in human-like cognitive behaviors as humans work with the stream of input information and changing states. Vector of real values described in representation principle could be thought of as state variables in a dynamical system. Nengo recurrent connection based on principle two could implement dynamics. One of the building blocks for a dynamical system is an integrator. An integrator in control applications is a component that integrates input signals over time and outputs its current value. The basic need for this element is to accumulate input signals. In applications, we could use this property as a memory. NEF implementation of integrator requires the population of neurons and recurrent connection. Integrator update equation(see 4.4) for classic control-theory integrator.

$$\dot{x} = f(x) + g(u) \tag{4.4}$$

would translate to two NEF transforms(see 4.5, 4.6) implemented as separate connections. NEF integrator example for input signal $sin(t)$ see Figure 4.2 in relationship to classical implementation.

$$f' = \tau f(x) + x \tag{4.5}$$

$$g' = \tau g(u) \tag{4.6}$$

Where τ is is target ensemble synapse value. Synapse applies filter and time delay to input current across the synapse, and the output is the current that will be induced in the postsynaptic neuron.

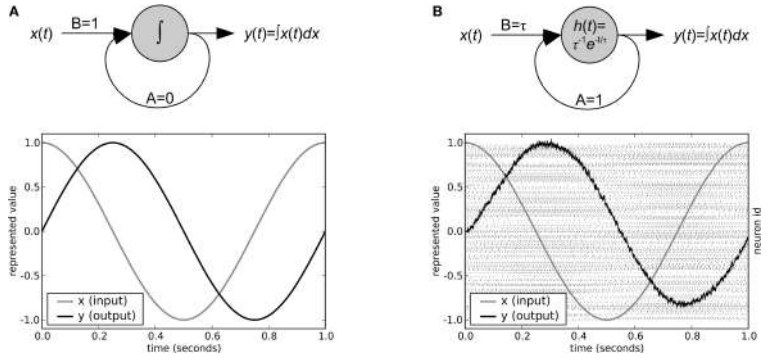


FIGURE 4.2: Visualisation for classic control-theory integrator and NEF integrator implemented with 300 LIF neurons for intput signal $x(t) = sin(t)$ Source: Stewart, 2009

4.3 Model development

4.3.1 Representing states

Rate CPG model described in section 4.1, consists of two oscillators, and each has two states. In total we have 4 states x_1, x_2, x_3, x_4. Similarly to the rate model, states do not have physical meaning but represent a saturation value of leaky integrator, which should reset after threshold. Our first task is to represent the state using SNNs and work on dynamics later. According to the Nengo representation principle (see 4.2.1), we could use population-based encoding and transform real values vector to spikes of neurons. This formulation uses spiking frequency instead of rate signals to accomplish dynamic computations with the integrate-and-fire neurons Nengo ensemble class implements this functionality. An ensemble is a building block in Nengo as it could represent any real vector using a population of neurons. Theoretically, one ensemble could represent all four values, and it would help us limit duplication of state, but one ensemble would also reduce the ability to manipulate individual state variables. Furthermore, because we do not know which neurons in the ensemble represent specific scalar in multi-dimensional vector, we can not reset it without changing all other vector values. A better approach would be to split each of the state values into different neuron ensembles. Because each pair represent a different locomotion phase, we call them swing1, stance1, swing2, stance2(see 4.3). Each half-center represented by 2 neuron ensembles, each representing neuron integrator.

4.3.2 Defining transitions

SNNs CPG model implements the same dynamics as rate CPG defined by differential equations (see equation 4.1). Because state variables are represented as separate ensembles, and we can only build connections between

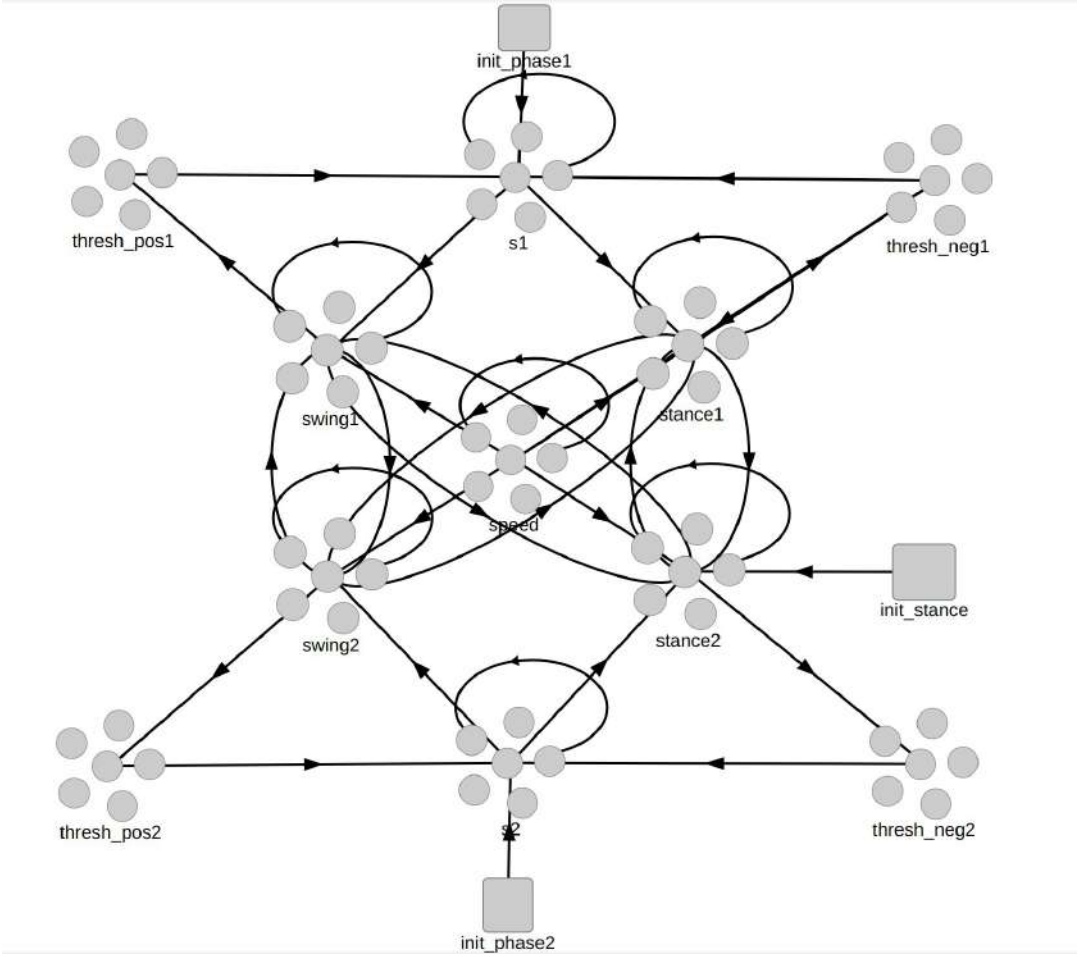


FIGURE 4.3: CPG for two limbs implemented using SNN. Visualized using NengoGUI

two ensembles, it is impossible to access all state values in one place. Therefore, we need to decouple dynamics equations into separate connections applied only to ensemble pairs. Resulting equations:

$$swing'_{1,2} = init\_swing + speed\_swing * u_{1,2} + inner\_inhibit * swing_{1,2} \quad (4.7)$$

$$\dot{stance}'_{1,2} = init\_stance + speed\_stance * u_{1,2} + inner\_inhibit * stance_{1,2} \quad (4.8)$$

$$\dot{swing}'_{1,2} = (1 - swing_{2,1}) * swing\_swing \quad (4.9)$$

$$\dot{swing}'_{1,2} = (1 - stance_{2,1}) * swing\_stance \quad (4.10)$$

$$\dot{stance}'_{1,2} = (1 - swing_{2,1}) * stance\_swing \quad (4.11)$$

$$\dot{stance}'_{1,2} = (1 - stance_{2,1}) * stance\_stance \tag{4.12}$$

Baseline model (x_1, x_2, x_3, x_4) values correspond to $(swing_1, stance_1, swing_2, stance_2)$. Equations 4.9 and 4.7 controls integration speed of a population based on current integrated value and system desired speed of locomotion. Parameters *init_swing* and *init_stance* represent base speed of integration for swing and stance for two limbs. *speed_swing* and *speed_swing* controls influence of input speed on integration. *inner_inhibit* is the same for swing and stance, and control influence of a state variable on itself. Equations 4.9, 4.10, 4.11, 4.12 connects integrators with each other and mainly responsible for symmetry and system self-regulation. Parameters *swing_swing*, *swing_stance*, *stance_stance*, *stance_swing* controls how different phases for opposite limbs influence each other. In total, there are 9 parameters that formulate dynamics.

Each update rule could then be translated to NEF neuron connections using dynamics principles(see 4.2.1) by multiplying function updates by required synapse value. In addition, if there are multiple connections to one ensemble, Nengo will automatically sum up their stimulus. The combination of all connections implements the same dynamics as in equation 4.1. Example of the Nengo implementation for swing phase dynamics.

```
tau = 0.1
speed = 1
def swing_feedback(x):
    dX = init_swing + speed_swing * speed + inner_inhibit * x
    return dX * tau + x

swing = nengo.Ensemble(500)
nengo.Connection(swing, swing,
                 function=swing_feedback,
                 synapse=tau)
```

Each of the Nengo connections could be represented as a python function. Above there is an example of such function "swing_feedback" that uses dynamics parameters from equasion 4.7. While building the network, Nengo sample points from the ensemble input dimension and propagate them through each connection function. The resulting points would be used for decoding weights optimization. Connection function used only during network creation and later only learned encoding and decoding weights used. So high-level dynamics parameters only take part in connection building.

By default, all initial values of an ensemble are zero, but the phase for two limbs should be shifted already from the stater. For this reason, we added initialization for the second limb stance. Its value controlled by "init_stance_position" parameter.

Visualization of all neurons connections could be found on final model visualization (see Figure 4.3).

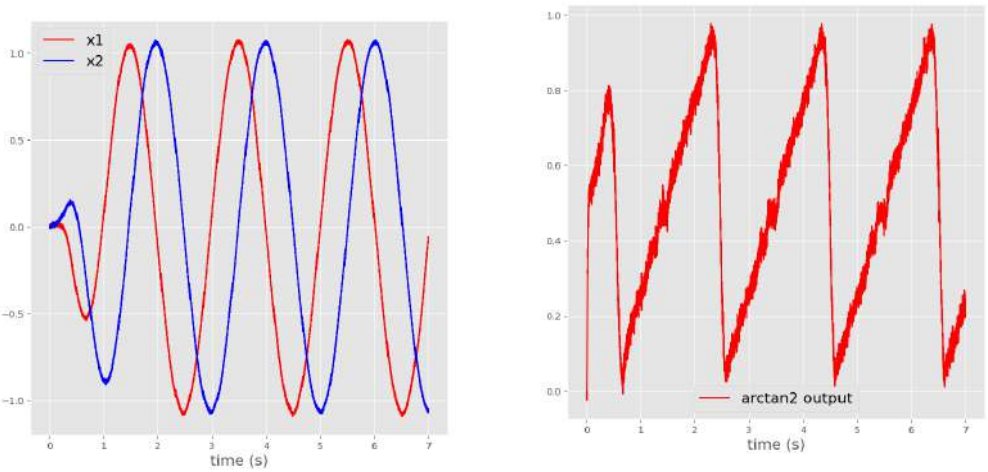
4.3.3 State switching

In section 4.3.2 we defined all integrators and their interconnections, but we still lack switching of state. In baseline model(see 4.1) each "phase" neuron resets it's value when achieve threshold. After reset to 0, limbs switched to opposite phase. This functionality allows for periodic switching of swing and stance phases. However, the state represented by the population of spikes cannot instantly change and can only gradually update its value. There are three different ways to implement this functionality:

- First approach is to incorporate reset functionality into state recurrent update(see equation 4.7). We need to set a huge negative update for the neuron state if its value reaches 1. Example of implementation for transition function.

```
tau = 0.1
speed = 1
def swing_feedback(x):
    if x >= 1:
        dX = init_swing + speed_swing * speed + \\
                inner_inhibit * x
    else:
        dx = -100
    return dX * tau + x
```

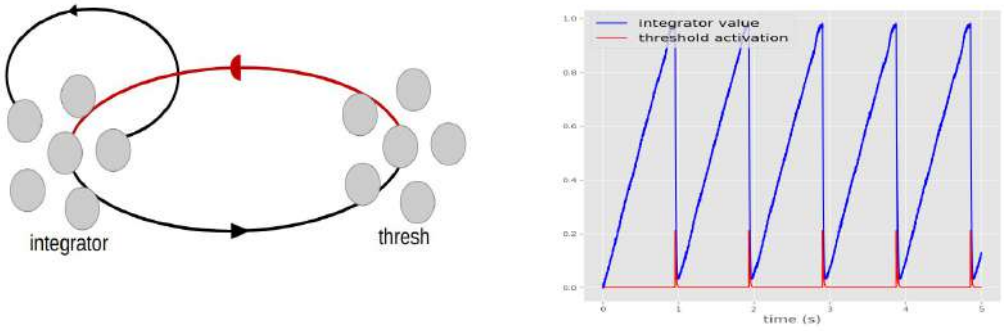
  Unfortunately, this function is too challenging to implement in one single neural ensemble. Furthermore, because the differential of a function at value 1 is undefined, points for training decoders evaluated using our function will be hugely different for two sides of 1, and the resulting function would be smoothed out.



(A) Points on unit circle of a neural oscillator (B) Output of arctan2 function for unit circle points

FIGURE 4.4: Step function implemented using oscillator

- The second option is to start from a self-sustaining neural oscillator[1]. Use the output of the neural oscillator(see Figure 4.4a), and compute the transition function from that. For example, if the oscillator provides the x, y coordinates of a point moving around a unit circle, we can use the arctan2 function to convert the point to an angle and later use it as output for the step signal(Figure 4.4b). However, this approach makes controlling speed integration much more complicated.
- The last option is to use a separate element that resets the NEF integrator. The idea is to turn off the neuron ensemble when it is not in the active phase. Turn off functionality implemented in nengo using inhibition procedure. It works by setting encoding weights of an ensemble to zero by connecting off signal to population encoding weights. Zeroing weights will automatically stop all spikes and therefore stop integration of input signal. Finally, we need an element that could detect integrator threshold. For this, we could rely on ensemble tuning curves that control when neurons start spiking. We set neuron intercepts parameter to our target threshold and when threshold ensemble spikes inhibit target integrator(figure 4.5a). Visualization of integrator output on figure 4.5b, the red line represents the time of threshold activation.



(A) Visualization of step function implementation using neuron inhibition.

(B) Output of neuron integrator and threshold neuron

FIGURE 4.5: Step function implemented using inhibition

Because the third option of step function implementation allows simple control of phase changes, we chose this option. So for each "phase" neuron, we need one step threshold and one state ensemble to remember the current phase. Visualisation for full model implementation with step ensembles("thresh_pos1", "thresh_neg1", ...) in Figure 4.3.

4.3.4 Input speed control

Our model input parameter is desired speed of locomotion. For each speed value, we should produce a swing and stance phase pattern that would be

[1]https://www.nengo.ai/nengo/examples/dynamics/oscillator.html

similar to the pattern in actual data. For evaluation and training, we need to change input speed during model simulation. We created a distinct ensemble for "Speed" representation that implements NEF integrator and constantly increment speed value. Then speed value is copied to each "phase" neuron that is used in recurrent connection. For this, we need to extend the "phase" neuron vector to two values to include speed value. For training, we use the speed for two limbs and will test different speeds during evaluation. In our simulation, speed would change its value from 0 to 1 that should correspond to stride cycle ranges $[.57, 1.91]$ seconds from empirical data.

4.3.5 Nengo parameters

The biggest question is deciding model parameters like the number of neurons to represent the state, radius of a representation, value of synapse filter. For deciding the number of neurons, we need to balance the accuracy of trained transition and simulation time. We will use 4000 neurons to represent the state and later run experiments to find an optimal number. Nengo ensemble could represent values in a unit cycle scaled by radius factor. So, for example, some huge number equal to radius are going to be "saved" as one on the unit circle and later scaled by radius. Because of this, we lose precision by increasing radius and having the same number of neurons. For our case phases, threshold values are one and speed range is [0, 1], but because we need to represent a vector with two values, they will not fit the unit cycle, so we should increase the radius to $\sqrt{1^2 + 1^2}$.

Synapse controls the size of a filter for spikes post-processing and decoding back to the numerical value. So basically influence how fast the system could react to changes, but there is a drawback with small window size, there will be much variance in output signals due to the stochastic nature of underlining spikes. So because of the fast nature of the motor control system and the instantaneous phase transition requirements, we chose a 0.01 synapse value.

Chapter 5

Solution

We implemented dynamics of a rate CPG model with SNNs, and the last problem is to find parameters that satisfy the properties of empirical data. Our dynamics parameters are *init_swing* , *init_stance*, *speed_swing*, *speed_swing*, *inner_inhibit*, *swing_swing*, *swing_stance*, *stance_stance*, *stance_swing* and "init_stance_position". In total there are 10 parameters. They are used to initialize ensemble connections. This section aims to define the error cost function based on locomotion characteristics of cats that evaluates the performance of simulated CPG created using dynamics parameters that should show the same characteristics. Similarly to the robotic problems, we also want to learn the dynamics of a CPG, but in our case, we already have predefined dynamics that we want to learn instead of robotics, where learning dynamics require exploration. Therefore, we consider this optimization task as a supervised learning problem. Our next step will be decisions on the optimization algorithm. After we find the parameter for the model, we should check its validity.

5.1 Error cost function

For error function creation we are guided by empirical data properties presented in chapter 3 derived from study by Halbertsma, 1983 and Goslow, Reinking, and Stuart, 1973. Because we want simultaneously optimize for different properties or error function consist if several parts:

- Phase duration error: calculates how model swing and stance phases are close to cats swing and stance for the same speed. We calculated error at the end of simulation when state changes recorded all range of input speeds. We split all time ranges into (swing, stance) pairs and calculated their combined stride duration. Then using equations 3.1 and 3.2 we compute expected swing, stance duration. These data points then used to calculate the root-mean-square error, which is our phase duration error

- Cycle ranges error: restrains stride duration from going out of range of cats data cycle duration. We do not directly optimize for power law for speed and cycle duration because we want to test it later. Instead, we calculate the root-mean-square error for the first stride and maximum range for stride duration and the last stride(recorded for max speed)

and minimum stride duration. This error will ensure that all cycle duration is in one range. We do the same calculation for two limbs and average result.

- Balance error: forces model to synchronize two limbs and balance locomotion. This error focused on interconnections between limbs. For balanced locomotion, double and single support for two limbs should go one after each other, and swing phases should not intersect as this would result in running, which we do not consider in this study. We compute the intersection of the swing phase of a limb and the stance phase of an opposite limb for balance error calculation and normalize by swing phase duration. This number shows what percentage of swing phase contained in the stance phase of the opposite limb. We compute this value for two limbs, and ideally, we want this percentage to be 100. For control of phase synchronization, we added the additional parameter "init_stance_position".

- Symmetry error: tries to position swing phase in the center of opposite limb stance phase. This error was added later in the study to overcome some cases when the swing phase starts simultaneously as the opposite leg's stance phase, which does not correspond to empirical data. However, the swing phase should not be strictly in the middle but lean towards it.

The final error is a sum of the above errors with coefficients.

$$error = 1.5 * phase\_error + balance\_error$$
$$+0.5 * range\_error + 0.2 * symmetry\_error$$

Coefficients represent the importance of each error for the final solution.

5.2 Optimisation

Due to the stochasticity of a simulation and the nature of the Nengo framework, we can not directly calculate gradient for underlying parameters. One of the reasons is that we use model parameters to train spiking neuron network transitions during the building procedure and later in simulation rely only on learned connections. Another reason is that we calculate error at the end of a simulation, accumulated across all phase periods. For this reason, we treat this problem as black-box optimization of a function that receives parameters, builds models, simulates it for 95 seconds, and compute the error. The time duration of a simulation was based on experiments in the baseline model. We used several algorithms for model optimization.

5.2.1 Nelder–Mead

Nelder–Mead(Gao and Han, 2010) widely popular numerical optimisation method used by default for function optimisation in MATLAB[1] and python SciPy[2] library. It could search for the minimum or maximum of a function in a multidimensional space. This method is often applied to nonlinear optimization problems for which we cannot calculate derivatives. Also, this algorithm showed good results in the baseline model, so we started from it. As starting parameters for an algorithm, we used parameters from the baseline model. However, this approach did not bring us good results. We think one reason is that Nelder–Mead converged to local optimum due to its heuristic search method. Starting parameters could be optimal to the baseline model but lead to the not optimal solution for Nengo port of CPG model. Another big problem is training duration. Nelder–Mead implementations in python do not utilize a multi-core system and run only in one thread.

5.2.2 Hyperout

For the next step, we wanted to scale the optimization procedure to multiple nodes and take into account long simulation time. The Nengo model optimization is a very similar problem to hyper-parameters searching for deep neural networks, as they also take a long time to evaluate. One of the frameworks to solve this problem is Hyperout [3]. It implements "Tree-Parzen Estimators"(Bergstra et al., 2011) algorithm. We also used "Tune: Scalable Hyperparameter Tuning"[4] framework to paralyze hyper-parameters search. One of the benefits of using this type of algorithm that we do not need to supply initial parameters for optimization. However, we should provide ranges in which to search for optimal parameters. We could estimate parameters ranges knowing locomotion properties of cats movements. Search space: [ht] The first optimization jobs archived a small error, but there was a problem with the symmetry of swing phases, so we decided to add a new error term that tries to account for this(see Chapter 5.1). The most successful optimization routine converged to a combined error value 0.17 with 500 simulations. For best params see Table 5.2. For visualization of training error check Figure 5.1. Hyperout trials are stochastic in time, so we sorted errors for better visualization.

5.2.3 HEBO

The third algorithm we tried is Heteroscedastic Evolutionary Bayesian Optimisation(HEBO)[5] created by Cowen-Rivers et al., 2020. It is Bayesian optimization family of algorithm which also adds evolutionary optimizers. It

[1] https://www.mathworks.com/help/optim/ug/fminsearch-algorithm.html
[2] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html
[3] http://hyperopt.github.io/hyperopt
[4] https://docs.ray.io/en/master/tune/index.html
[5] https://github.com/huawei-noah/noah-research/tree/master/HEBO

TABLE 5.1: Search space

Param	min	max
init_stance	0	2
init_stance_position	0	1
init_swing	3	6
speed_stance	2	4
speed_swing	2	5
inner_inhibit	-1	1
swing_swing	-1	1
stance_swing	-1	1
swing_stance	-1	1
stance_stance	-1	1

TABLE 5.2: Best params

init_stance	init_stance_position	init_swing	speed_stance	speed_swing
0.54503	0.63792	5.0592	3.6923	3.2043
inner_inhibit	stance_stance	stance_swing	swing_stance	swing_swing
-0.45334	0.80108	-0.85135	-0.64631	0.10515

won black-box optimizers challenge[6] for hyperparameters search for deep learning. We used the same parameters space from table 5.1. Although we could achieve comparable results as with Hyperout, HEBO takes more function evaluation in our case to coverage to a meaningful result.

5.3 Neurons count optimisation

As mentioned in section 4.3.5 we used 4000 neurons to represent state values during training. However, it is possibly not the optimal number, so we could run several experiments to test this. In the first experiment, we evaluated model phase characteristics five times for different neuron numbers and recorded deviations in phase parameters. Recorded standard deviations of phases $[0.5, 0.8, 0.02, 0.03, 0.04, 0.04, 0.001]$ for $[1000, 1500, 2000, 2500, 3000, 3500, 4000]$ of neurons. From this data, we could see a big gap in std from 1500 and 2000 neurons. For the next experiment, we evaluated best-tuned parameters from table 5.2 with a different number of neurons, and the resulted plot in Figure 5.2. The final experiment also shows that 2000 neurons have the same error as the model with 4000, but it stops working when it goes below 2000. We

[6] https://bbochallenge.com/leaderboard

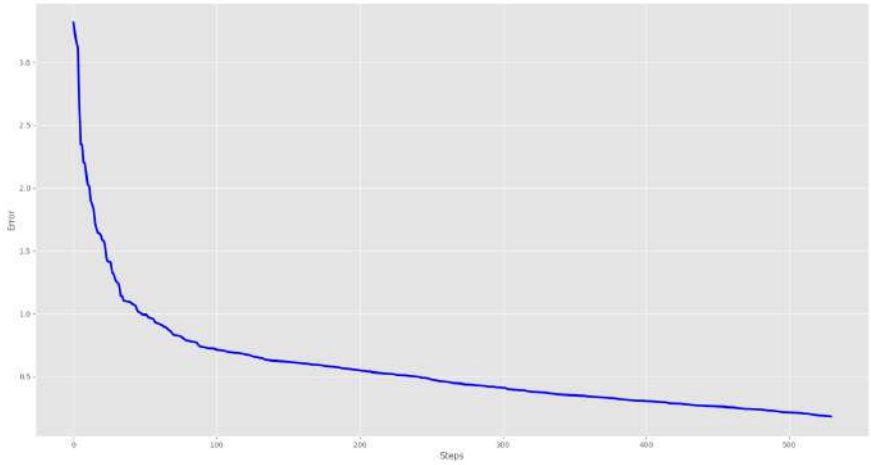


FIGURE 5.1: Error values for Hyperout optimisation sorted in descending order

could conclude that 2000 is an optimal number for integrator value representation using a Nengo ensemble from these two experiments. Father research is needed to investigate why the model disintegrates with values below 2000.

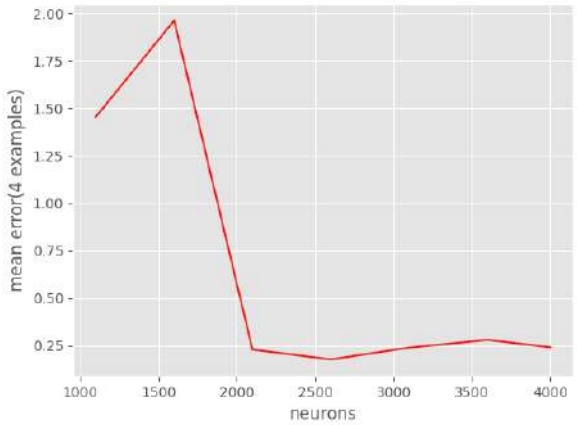


FIGURE 5.2: Evaluation of best-fit parameters with different number of state neurons

5.4 Validation

For validation, we would use the best-trained parameters from Table 5.2. This section will check how the model behaviors at different speeds and if it satisfies power low for cycle duration and checks if it is stable to damage.

5.4.1 Phase duration characteristics

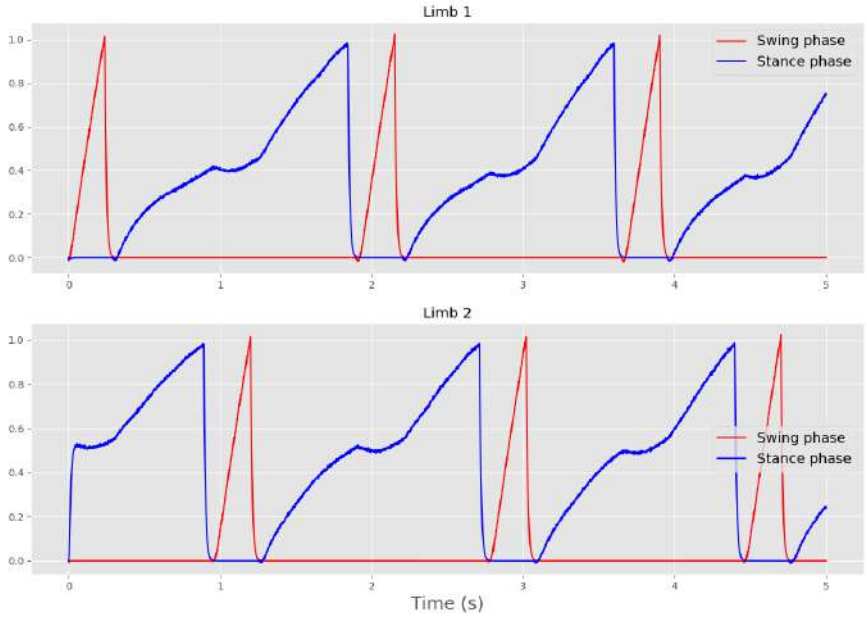
First, we want to check how the model perforce at different speeds(Figure 5.3a and 5.3b). The swing and stance phases go one after the other, and the two limbs are half-cycle apart. Swing phases(red line) are in the center of stance phases(blue line). At high speeds model also stays stable with a much-shorted cycle duration. In this section, we want to estimate how generated cycles are close to empirical data and calculate R^2 metric. For this, we merge cycle duration for two limbs and, using equations 3.1 and 3.2 get ground truth data. The final relationship for one of the limbs on Figure 5.4, dotted blue line, and dotted red line show expected linear relation. Calculated metric $R^2_{swing_1}$ and $R^2_{stance_1}$ for one limb are 0.903 and 0.998. These results are a bit worse but comparable with the baseline model.

5.4.2 Power law

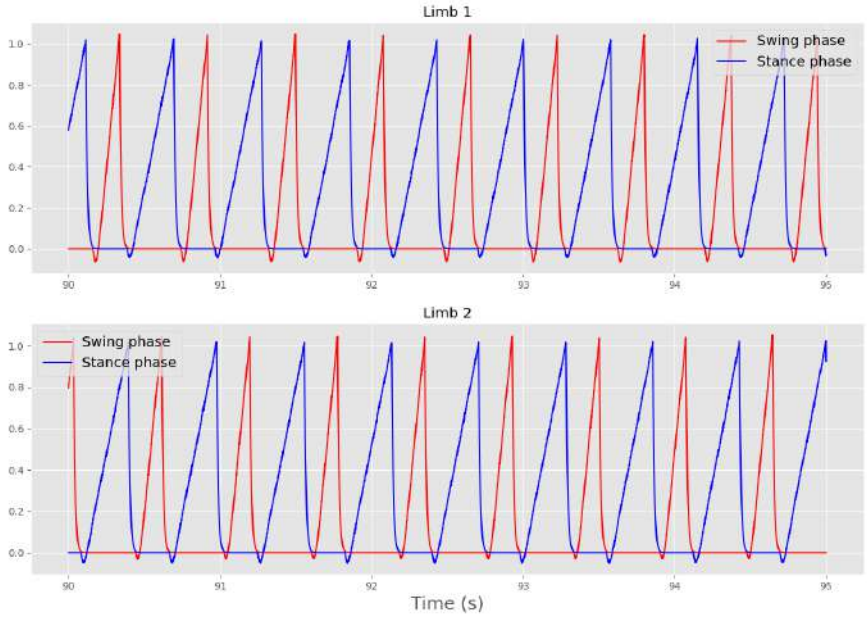
One of the biggest questions in motor control is to identify what input signal drive CPG output. It could be range from high-order variables like position, velocity, force, or some low-level features. If low-level features are represented at high levels of the neural hierarchy, then the musculoskeletal motor organization's complexity and interactions with the environment are not solved in the low-level neural hierarchy. This the statement is unlikely in absolute terms; therefore, the modality of inputs converging on the rhythm generating networks has a global context, for example, the speed of forward progression(Yakovenko, 2011). For testing this idea, we could represent the output of a CPG model in terms of its input and check their relationship. So we chose forward progressions velocity as an output of our model. For calculation of velocity, we could rely on the empirical power relationship between stride cycle duration and velocity(Goslow, Reinking, and Stuart, 1973). The first step would be to calculate the relationship of our input parameters to cycle duration(see Figure 5.5a), and they do represent the power relationship. The next step would be to transform cycle duration(T_c) using reversed equation $V = (T_c/0.5445)^{1/-0.592}$. Final relationship between input CPG speed and output forward progressions velocity is presented in Figure 5.5b and as it turns out this relationship is highly linear($R^2 = 0.990$). It shows that input of the CPG model represented as a desired speed of locomotion adjusts output phases, so they correspond to target velocity. Results support the idea of speed input as a modality of rhythm generating networks.

5.4.3 Damage simulation

One of the possible benefits of stochastic spiking models is bigger tolerance for noise and damage to elements. Nengo represents its values and computes transitions using a population of neurons, so losing some neurons should not play a significant role in some expend. In this section, we want to test



(A) Simulation from 0 to 5 seconds



(B) Simulation from 90 to 95 seconds

FIGURE 5.3: Plots of swing/stance phase changes for two limbs and within different time periods

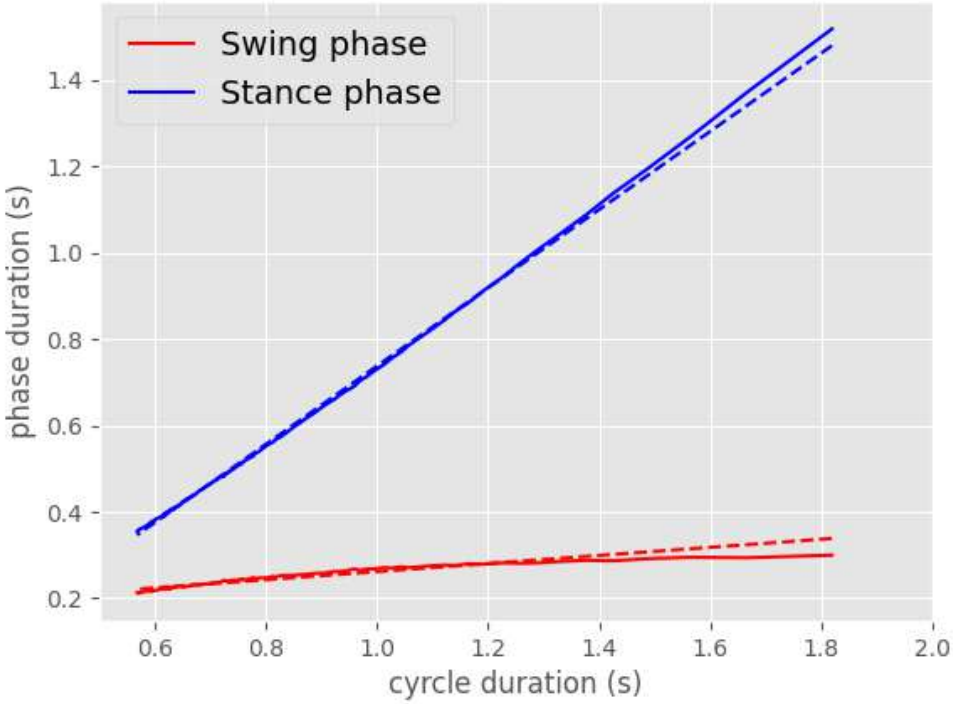
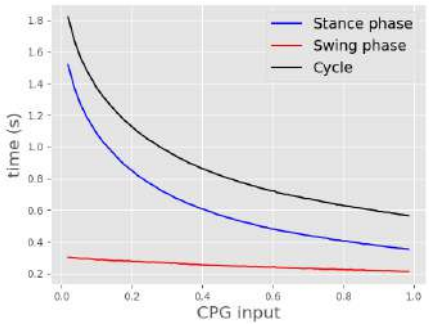


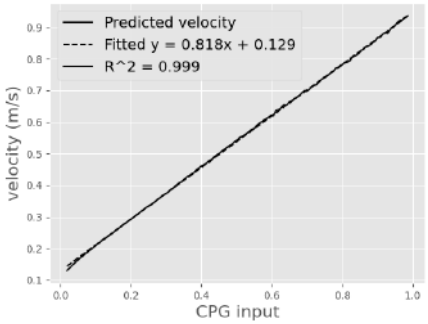
FIGURE 5.4: The bilateral pattern of locomotor phase modulation with optimized CPG model parameters. The relationship between phase and cycle duration for each integrator is plotted together with the desired Halbertsma, 1983 best-fit linear model.

with how much-damaged neurons CPG model could still function. We apply damage to the system by turning off some of the neurons during stimulation. We evaluate the functionality of a model by calculating its error in the same way we did for optimization. If error increase significantly, we could conclude that model is no longer functional. We simulate the model for the same 95 seconds and test for 0 to 900 damaged neurons out of 5000. Simulation results are in Figure 5.6. Model maintained moderately good error(0.69) up to 150 damaged neurons and remained functional up to 600 damaged neurons. These results prove that model could withstand some damage.

Another test we would do is tolerance to noise. Ensemble class in Nengo supports additional noise parameter which alters the output of a neuron population. For this experiment, we would use white noise with a mean 0 and standard deviation from 0 to 0.15. In the same way, as with damage experiments, we would run simulations for 95 seconds with a different number of noise parameters and investigate resulted error curve. Resulted plot in Figure 5.7. The system is stable up to 0.06 standard deviation Gaussian noise.



(A) Power relationship between CPG input speed parameter and model cycle durations



(B) Relationship of input CPG speed and locomotion velocity produced from model stride cycle duration. Relationship is highly linear($R^2 = 0.990$).

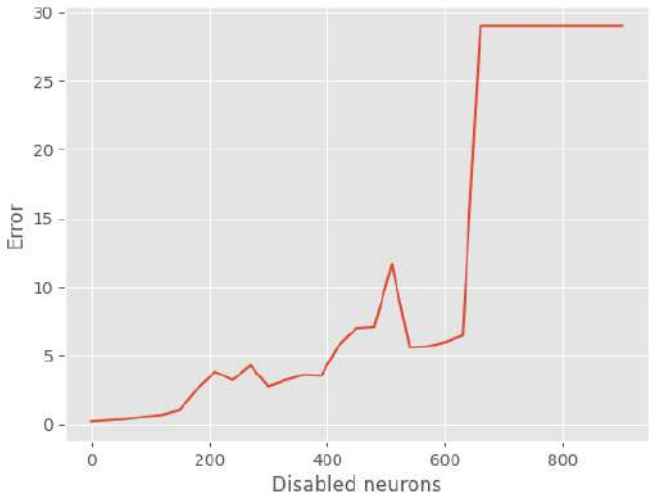


FIGURE 5.6: Progression of error in Nengo CPG model when we remove neurons from state Ensemble

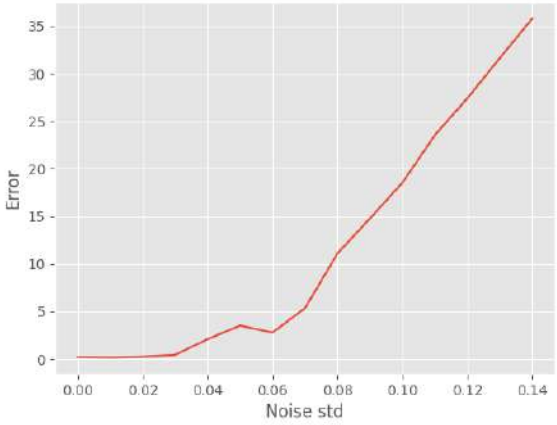


FIGURE 5.7: Progression of error in Nengo CPG model when we add noise to state Ensemble

Chapter 6

Conclusions

6.0.1 Results Summary

Overall results of the project are successful as we answered original research questions. First of all, we implemented a bilateral half-center oscillators CPG model using SNNs, and it proved the validity of the Nengo framework for solving these types of problems. The resulted spiking model has similar behavior at different speeds as the rate model. Also, it reproduces a linear relationship between model speed parameter and velocity of forward progression. The resulting accuracy for phase characteristics is close to the rate model. Transitioning model to Nengo allows to run it on SpiNNaker hardware which is real-time by design and save energy due to sparse signals which could be essential for embedded system. In addition, we showed some noise and damage tolerance present in our CPG model.

6.0.2 Future work

One of the most significant limitations of an existing model is the tedious learning procedure. Error function consists of many moving parts, and black-box optimization may not be the best choice. The better option would be to unitize Nengo learning tools [1] and transform the problem into a reinforcement learning task. The use of build in learning methods would benefit constant learning and error adjustment during the simulation and not limit connections to the base set of differential equations used during model building. Another possible area of research is to check for similar patterns of the Nengo CPG model and biological CPGs.

[1] https://www.nengo.ai/nengo/examples/learning/learn-communication-channel.html

Bibliography

Anderson, John R. et al. (2004). "An Integrated Theory of the Mind." In: *Psychological Review* 111.4, pp. 1036–1060. DOI: 10.1037/0033-295x.111.4.1036. URL: https://doi.org/10.1037/0033-295x.111.4.1036.

Bergstra, James et al. (2011). "Algorithms for Hyper-Parameter Optimization". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2546–2554. ISBN: 9781618395993.

Collins, J. J. and S. A. Richmond (Sept. 1994). "Hard-wired central pattern generators for quadrupedal locomotion". In: *Biological Cybernetics* 71.5, pp. 375–385. DOI: 10.1007/bf00198915. URL: https://doi.org/10.1007/bf00198915.

Cowen-Rivers, Alexander I et al. (2020). "HEBO: Heteroscedastic Evolutionary Bayesian Optimisation". In: *arXiv preprint arXiv:2012.03826*. winning submission to the NeurIPS 2020 Black Box Optimisation Challenge.

Cuevas-Arteaga, Brayan et al. (2017). "A SpiNNaker Application: Design, Implementation and Validation of SCPGs". In: *Advances in Computational Intelligence*. Springer International Publishing, pp. 548–559. DOI: 10.1007/978-3-319-59153-7_47. URL: https://doi.org/10.1007/978-3-319-59153-7_47.

Dimitrijevic, Milan R., Yuri Gerasimenko, and Michaela M. Pinter (Nov. 1998). "Evidence for a Spinal Central Pattern Generator in Humansa". In: *Annals of the New York Academy of Sciences* 860.1 NEURONAL MECH, pp. 360–376. DOI: 10.1111/j.1749-6632.1998.tb09062.x. URL: https://doi.org/10.1111/j.1749-6632.1998.tb09062.x.

Eliasmith, C. (2013). *How to build a brain: a neural architecture for biological cognition*. Oxford University Press.

Eliasmith, Chris (June 2005). "A Unified Approach to Building and Controlling Spiking Attractor Networks". In: *Neural Computation* 17.6, pp. 1276–1314. DOI: 10.1162/0899766053630332. URL: https://doi.org/10.1162/0899766053630332.

Endo, Gen et al. (Feb. 2008). "Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot". In: *The International Journal of Robotics Research* 27.2, pp. 213–228. DOI: 10.1177/0278364907084980. URL: https://doi.org/10.1177/0278364907084980.

Espinal, A. et al. (2016). "Quadrupedal Robot Locomotion: A Biologically Inspired Approach and Its Hardware Implementation". In: *Computational Intelligence and Neuroscience* 2016, pp. 1–13. DOI: 10.1155/2016/5615618. URL: https://doi.org/10.1155/2016/5615618.

Fukuoka, Yasuhiro, Hiroshi Kimura, and Avis H. Cohen (Mar. 2003). "Adaptive Dynamic Walking of a Quadruped Robot on Irregular Terrain Based

on Biological Concepts". In: *The International Journal of Robotics Research* 22.3-4, pp. 187–202. DOI: 10.1177/027836490302003004. URL: https://doi.org/10.1177/027836490302003004.

Gao, Fuchang and Lixing Han (May 2010). "Implementing the Nelder-Mead simplex algorithm with adaptive parameters". In: *Computational Optimization and Applications* 51.1, pp. 259–277. DOI: 10.1007/s10589-010-9329-3. URL: https://doi.org/10.1007/s10589-010-9329-3.

Gerstner, W. (2001). "Chapter 12 A framework for spiking neuron models: The spike response model". In: *Neuro-Informatics and Neural Modelling*. Elsevier, pp. 469–516. DOI: 10.1016/s1383-8121(01)80015-4. URL: https://doi.org/10.1016/s1383-8121(01)80015-4.

Gerstner, Wulfram and Werner M. Kistler (Aug. 2002). *Spiking Neuron Models*. Cambridge University Press. DOI: 10.1017/cbo9780511815706. URL: https://doi.org/10.1017/cbo9780511815706.

Goslow, George E., Robert M. Reinking, and Douglas G. Stuart (Sept. 1973). "The cat step cycle: Hind limb joint angles and muscle lengths during unrestrained locomotion". In: *Journal of Morphology* 141.1, pp. 1–41. DOI: 10.1002/jmor.1051410102. URL: https://doi.org/10.1002/jmor.1051410102.

Gutierrez-Galan, Daniel et al. (Mar. 2020). "Neuropod: A real-time neuromorphic spiking CPG applied to robotics". In: *Neurocomputing* 381, pp. 10–19. DOI: 10.1016/j.neucom.2019.11.007. URL: https://doi.org/10.1016/j.neucom.2019.11.007.

Halbertsma, J. M. (1983). "The stride cycle of the cat: the modelling of locomotion by computerized analysis of automatic recordings". In: *Acta Physiol Scand Suppl* 521, pp. 1–75.

Hellgren, J., S. Grillner, and A. Lansner (Nov. 1992). "Computer simulation of the segmental neural network generating locomotion in lamprey by using populations of network interneurons". In: *Biological Cybernetics* 68.1, pp. 1–13. DOI: 10.1007/bf00203132. URL: https://doi.org/10.1007/bf00203132.

Hinton, G. E. (July 2006). "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786, pp. 504–507. DOI: 10.1126/science.1127647. URL: https://doi.org/10.1126/science.1127647.

Hodgkin, A. L. and A. F. Huxley (Aug. 1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117.4, pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764. URL: https://doi.org/10.1113/jphysiol.1952.sp004764.

Ijspeert, Auke Jan (Apr. 2001). "A connectionist central pattern generator for the aquatic and terrestrial gaits of a simulated salamander". In: *Biological Cybernetics* 84.5, pp. 331–348. DOI: 10.1007/s004220000211. URL: https://doi.org/10.1007/s004220000211.

— (May 2008). "Central pattern generators for locomotion control in animals and robots: A review". In: *Neural Networks* 21.4, pp. 642–653. DOI: 10.1016/j.neunet.2008.03.014. URL: https://doi.org/10.1016/j.neunet.2008.03.014.

Izhikevich, E.M. (Sept. 2004). "Which Model to Use for Cortical Spiking Neurons?" In: *IEEE Transactions on Neural Networks* 15.5, pp. 1063–1070. DOI: 10.1109/tnn.2004.832719. URL: https://doi.org/10.1109/tnn.2004.832719.

Krizhevsky, Alex (2009). *Learning multiple layers of features from tiny images*. Tech. rep.

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: URL: http://yann.lecun.com/exdb/mnist/.

Lele, Ashwin Sanjay et al. (2020). *Learning to Walk: Spike Based Reinforcement Learning for Hexapod Robot Central Pattern Generation*. arXiv: 2003.10026 [cs.NE].

Pande, Sandeep et al. (Jan. 2013). "Modular Neural Tile Architecture for Compact Embedded Hardware Spiking Neural Network". In: *Neural Processing Letters* 38.2, pp. 131–153. DOI: 10.1007/s11063-012-9274-5. URL: https://doi.org/10.1007/s11063-012-9274-5.

Prochazka, Arthur and Sergiy Yakovenko (2007). "The neuromechanical tuning hypothesis". In: *Progress in Brain Research*. Elsevier, pp. 255–265. DOI: 10.1016/s0079-6123(06)65016-4. URL: https://doi.org/10.1016/s0079-6123(06)65016-4.

Rasmussen, Daniel and Chris Eliasmith (Jan. 2014). "A spiking neural model applied to the study of human performance and cognitive decline on Raven's Advanced Progressive Matrices". In: *Intelligence* 42, pp. 53–82. DOI: 10.1016/j.intell.2013.10.003. URL: https://doi.org/10.1016/j.intell.2013.10.003.

Rostro-Gonzalez, H. et al. (Dec. 2015). "A CPG system based on spiking neurons for hexapod robot locomotion". In: *Neurocomputing* 170, pp. 47–54. DOI: 10.1016/j.neucom.2015.03.090. URL: https://doi.org/10.1016/j.neucom.2015.03.090.

Rueckauer, Bodo et al. (Dec. 2017). "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification". In: *Frontiers in Neuroscience* 11. DOI: 10.3389/fnins.2017.00682. URL: https://doi.org/10.3389/fnins.2017.00682.

Schlichter, Tamara J et al. (July 2010). "Phase response properties of an idealized half-center oscillator". In: *BMC Neuroscience* 11.S1. DOI: 10.1186/1471-2202-11-s1-p3. URL: https://doi.org/10.1186/1471-2202-11-s1-p3.

Selverston, Allen I. et al. (Dec. 2000). "Reliable circuits from irregular neurons: A dynamical approach to understanding central pattern generators". In: *Journal of Physiology-Paris* 94.5-6, pp. 357–374. DOI: 10.1016/s0928-4257(00)01101-3. URL: https://doi.org/10.1016/s0928-4257(00)01101-3.

Sterratt, David et al. (2009). *Principles of Computational Modelling in Neuroscience*. Cambridge University Press. DOI: 10.1017/cbo9780511975899. URL: https://doi.org/10.1017/cbo9780511975899.

Stewart, Terrence (2009). *Python scripting in the Nengo simulator*. DOI: 10.3389/neuro.11.007.2009. URL: https://www.frontiersin.org/files/Articles/359/fninf-03-007/image_n/fninf-03-007-g003.gif.

Stewart, Terrence C., Trevor Bekolay, and Chris Eliasmith (2012). "Learning to Select Actions with Spiking Neurons in the Basal Ganglia". In: *Frontiers in Neuroscience* 6. DOI: 10.3389/fnins.2012.00002. URL: https://doi.org/10.3389/fnins.2012.00002.

Stone, James V. (2018). *Principles of Neural Information Theory: Computational Neuroscience and Metabolic Efficiency*. 1st. Sebtel Press. ISBN: 0993367925.

Traven, H. G. et al. (Aug. 1993). "Computer simulations of NMDA and non-NMDA receptor-mediated synaptic drive: sensory and supraspinal modulation of neurons and small networks". In: *Journal of Neurophysiology* 70.2, pp. 695–709. DOI: 10.1152/jn.1993.70.2.695. URL: https://doi.org/10.1152/jn.1993.70.2.695.

Wang, Wei et al. (Oct. 2019). "Voltage-control oscillator based on Pt/C/NbOx/TiN device with highly improved threshold switching performances". In: *Science China Physics, Mechanics & Astronomy* 62.12. DOI: 10.1007/s11433-019-1463-y. URL: https://doi.org/10.1007/s11433-019-1463-y.

Williams, T. (Oct. 1992). "Phase coupling by synaptic spread in chains of coupled neuronal oscillators". In: *Science* 258.5082, pp. 662–665. DOI: 10.1126/science.1411575. URL: https://doi.org/10.1126/science.1411575.

Winter, David A. (Mar. 1984). "Kinematic and kinetic patterns in human gait: Variability and compensating effects". In: *Human Movement Science* 3.1-2, pp. 51–76. DOI: 10.1016/0167-9457(84)90005-8. URL: https://doi.org/10.1016/0167-9457(84)90005-8.

Wu, Yujie et al. (2018). *Direct Training for Spiking Neural Networks: Faster, Larger, Better*. arXiv: 1809.05793 [cs.NE].

Yakovenko, S. et al. (Aug. 2005). *Control of Locomotor Cycle Durations*. DOI: 10.1152/jn.00991.2004. URL: https://journals.physiology.org/na101/home/literatum/publisher/physio/journals/content/jn/2005/jn.2005.94.issue-2/jn.00991.2004/production/images/large/z9k0080547710008.jpeg.

Yakovenko, Sergiy (2011). "A hierarchical perspective on rhythm generation for locomotor control". In: *Progress in Brain Research*. Elsevier, pp. 151–166. DOI: 10.1016/b978-0-444-53825-3.00015-2. URL: https://doi.org/10.1016/b978-0-444-53825-3.00015-2.

Yakovenko, Sergiy, Anton Sobinov, and Valeriya Gritsenko (Oct. 2018a). *Analytical CPG model driven by limb velocity input generates accurate temporal locomotor dynamics*. DOI: 10.7717/peerj.5849. URL: https://pubmed.ncbi.nlm.nih.gov/30425886/#&gid=article-figures&pid=figure-1-uid-0.

— (Aug. 2018b). "Analytical CPG model driven by single-limb velocity input generates accurate temporal locomotor dynamics". In: DOI: 10.7287/peerj.preprints.26734v2. URL: https://doi.org/10.7287/peerj.preprints.26734v2.

Yu, Junzhi et al. (Mar. 2014). "A Survey on CPG-Inspired Control Models and System Implementation". In: *IEEE Transactions on Neural Networks and*

Learning Systems 25.3, pp. 441–456. DOI: 10.1109/tnnls.2013.2280596. URL: https://doi.org/10.1109/tnnls.2013.2280596.