SNS-TOOLBOX: A SUITE OF TOOLS FOR THE DESIGN, OPTIMIZATION, AND IMPLEMENTATION OF SYNTHETIC NERVOUS SYSTEMS

by

WILLIAM ROBERT PILLERS NOURSE

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Department of Electrical, Computer, and Systems Engineering

CASE WESTERN RESERVE UNIVERSITY

August, 2024

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

William Robert Pillers Nourse

candidate for the degree of **Doctor of Philosophy**[1].

Committee Chair

M. Cenk Çavuşoğlu

Committee Member

Roger Quinn

Committee Member

Gregory Lee

Committee Member

Michael Lewicki

Committee Member

Nicholas Szczecinski

Date of Defense

May 13, 2024

[1]We also certify that written approval has been obtained for any proprietary material contained therein.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

Firstly, I must thank my family for their unfailing support throughout this endeavor. Thank you especially to my parents for always cheering me on, and to Elizabeth for joining me and helping to get over the finish line. I would not be where I am today without all of you.

Next, I must thank my research mentors, Dr. Quinn and Dr. Szczecinski. Dr. Quinn always helped me navigate the twists and turns as I settled on a research topic, and I always be grateful that he took a chance on an EE and let me discover an area of work I never knew was possible. Dr. Szczecinski has truly been instrumental in the details of this work, and has opened my eyes to the magic of neuroscience and insects. Without his guidance, this work would have never been completed, and I hope someday to merely approach his talents. I must also thank Dr. Çavuşoğlu, Dr. Lee, and Dr. Lewicki for their feedback in preparing this dissertation, as well as agreeing to participate on my committee.

Over the years I have had the pleasure of working with some of the best colleagues I could have asked for in the Biorobotics Lab at CWRU. The collaborative environment on the eighth floor of Glennan has fostered some incredible discussions, which have helped shape both this work as well as my work going forward. Therefore I must thank, in no particular order and not limited to, the following people: Clarus Goldsmith, Shanel Pickard, Ian Adams, Fletcher Young, Marshaun Fitzpatrick, Ken Moses, Nicole Graf, Natasha Rouse, and Shane Riddle. I would also like to especially thank Clayton Jackson and Ben Rubinstein, who greatly assisted with the physics simulation and gradient backpropagation in this work.

SNS-Toolbox: A Suite of Tools for the Design, Optimization, and Implementation of Synthetic Nervous Systems

Abstract

by

WILLIAM ROBERT PILLERS NOURSE

In this dissertation I present SNS-Toolbox, an open-source software package for the design and simulation of networks of biologically inspired neurons and synapses, also known as synthetic nervous systems (SNS). SNS-Toolbox allows SNS networks to be designed using a lightweight Python API, simulated in real-time on consumer computer hardware, and executed onboard physical robotic systems. I also present a companion package to SNS-Toolbox which allows simulation and training of large SNS networks using gradient backpropagation. This software is released under an open-source license with online documentation for ease of use, and has been disseminated to other researchers for their use. As a demonstration, I use SNS-Toolbox to implement a stereo visual motion detector, based on circuitry present within the *Drosophila melanogaster* (fruit fly) optic lobe. This network analyzes local motion at each point within a visual field, and returns an estimate of global motion when subjected to grating stimuli. Finally I showcase the design of FlyWheel, a robotic benchmark for studying models of insect vision and applying SNS networks to physical hardware. This body of work marks the first tool which is capable of simulating SNS networks with hundreds to thousands of neurons and synaptic connections in real-time or faster, optimize networks with chemical reversal potentials using gradient backpropagation, and interface these networks for control of external systems.

Chapter 1

INTRODUCTION

While it may seem that roboticists have different research interests than neuroscientists and biologists, many in fact share a common goal: Understanding how systems of varying intelligence interact with their environment to generate adaptive behavior [24]. In robotics today, robots are being used to collect data on real world interaction in order to generate large scale foundation models of how physical systems behave in interacting settings. Similarly, efforts are underway in neuroscience to map the location and connections of every neuron in connectomic maps of animal nervous systems, and experimentally deduce the functional role of each neuron based on behavioral experiments with electrical or optical recordings. In this light, robotics and neuroscience have the possibility of working together to advance their common goals: By implementing models of nervous systems which interact within a real environment, neuroscientists can validate their models and run experimental studies not possible in animals [83, 134], and roboticists can work towards understanding how intelligent behavior can emerge from the relatively small nervous systems in animals [9].

For this synergy of interests to occur, it is vital to have software which can take a biologically-inspired neural network, also known as a synthetic nervous system (SNS), and run that network in a closed loop with a robotic system. While there is a wide variety of neural simulation software available today [15, 51, 59, 141], most do not offer an easy path to interfacing the neural simulation with external systems such as robots or physics engines. Other frameworks which can interface more easily with external systems exist [42, 91], but either reduce the complexity of neural dynamics available or suffer performance issues when scaling to large networks [28].

One area where artificial neural networks (ANNs) amd modern neural simulators perform simplifications is that of synaptic dynamics. Most large-scale simulators focus on synaptic weights [42, 141], ignoring more complex synaptic dynamics such as chemical and electrical synapses. While many elements of these dynamics can be approximated using weights, using the synaptic dynamics closer to those seen in animal nervous systems allow SNS networks to perform some complex computations with much smaller networks than ANNs, such as multiplication and modulation using chemical synapses [52, 121] or coordinate transformation using electrical synapses [53, 97]. **The primary aim of this dissertation** is to develop software for the design and simulation of networks with these complex bio-plausible dynamics, as well as to interface these networks with other systems for the purposes of control and interaction.

A domain of interest in both robotic systems and neuroscience is visual processing of motion. Estimation of the body's motion based on changes in visual information is a problem known as ego-motion estimation [71], and is a problem which has been solved in the nervous system. In the fruit fly *Drosophila melanogaster* in particular, an extremely detailed connectivity map has been produced which highlights the exact anatomy of the brain region responsible for this behavior [108, 115]. **The secondary aim of this dissertation** is the design of a reduced order implementation of the visual motion processing network in insects, which is capable of discriminating direction of motion and could be implemented on embedded robotic hardware.

Small networks of neurons can be designed and tuned by hand or analytic calculations, and is often done in biological modeling [121]. However, this analysis becomes intractable when extended to very large networks that are presented with complex stimuli such as natural images. While numerous forms of optimization exist for large neural networks, gradient backpropagation is currently the dominant choice for optimizing large networks in machine learning with both feedforward [78]

and recurrent [136] connectivity. While this is generally not viewed as the optimization method used in real nervous systems, the optimization performance for implementation of neural systems using backpropagation is currently the state of the art. Plenty of frameworks exist for optimizing artificial neural networks (ANNs) [95] and even spiking neural networks (SNNs) [42, 58] using gradient descent, but none currently exist which handle the complex dynamics of chemical synapses. While traditionally ignored in the world of ANNs, chemical synapses have been shown to exhibit useful nonlinear behavior which is not possible using synaptic weights alone [52, 121]. Based on this, **the third and final aim of this dissertation** is the design of software which allows for the optimization of large SNS networks using modern gradient-based optimization.

After Chapter 2, every chapter in this dissertation consists of work that has been either published, presented at, or submitted for publication in a peer-reviewed journal, conference tutorial, or conference proceeding. I am the first author of each of these works, and they are reproduced here in an order which is roughly chronological but primarily for logical organization. Chapter 2 provides an overview in the fields of SNS networks, neural simulation, and visual motion processing in both insects and computational approaches.

Chapter 3 introduces SNS-Toolbox, the primary output of this dissertation. SNS-Toolbox is an open-source Python software package which enables the design and simulation of SNS networks using heterogeneous combinations of bio-plausible neural and synaptic dynamics. Networks can be compiled to simulate on either a CPU or GPU, and are easy to interface with external systems. As a demonstration of this, I showcase examples of SNS-Toolbox controlling a robotic system in the robot operating system (ROS) [99] and a musculoskeletal biomechanical model of rat locomotion in the physics simulator MuJoco [127].

Using SNS-Toolbox, in Chapter 4 I present an SNS for estimating local visual motion. This network is inspired by the connectivity information available for the

optic lobe in the fruit fly *Drosophila melanogaster*, and aims to be a minimal neural representation which is suitable for future implementation on a robotic platform. Just as in the insect, visual information is split into two pathways for processing increasing or decreasing change in brightness, and these pathways are nonlinearly summed across neighboring pixels to produce a local estimate of motion. I evaluate this system on a collection of simulated visual gratings, and show that the network is capable of producing sharp directional selectivity.

In order to expand this model to larger sizes, it was necessary to iterate on SNS-Toolbox in order to handle significantly larger layer-based networks in real-time. In Chapter 5 I introduce SNSTorch, an open-source companion package to SNS-Toolbox designed to simulate large populations of non-spiking neurons which can be combined into layered networks using chemical synapses. I demonstrate the performance improvements of SNSTorch over SNS-Toolbox for large networks, and use gradient backpropagation within PyTorch [95] to optimize parameter tuning via regression in a small network as well as a large-scale recurrent SNS for sequence classification. This marks the first time that chemical synapses have been incorporated into a framework compatible with modern machine learning techniques, and opens the door for exploration of their benefits in other domains.

In Chapter 6, I outline the products of initial efforts to develop an SNS for global visual motion estimation in natural environments. I first describe the design of FlyWheel, a mobile robot which uses a stereo visual system to model the visual information available to the fruit fly. I then use FlyWheel to record a collection of image sequences during rotation, and then augment these sequences to generate an open-source dataset for global motion estimation of over 21,000 one-second clips. I then present initial results for adapting the network presented in Chapter 4 for naturalistic settings.

Finally, in Chapter 7 I discuss the novelty of and impact of this work in neural simulation, as well as future work in expanded capabilities of the software as well

as improving the insect-inspired visual motion processing network.

Over the scope of this dissertation, I describe the design and use of SNS-Toolbox, as well as a companion package SNSTorch. These are the first tools which are capable of simulating large-scale synthetic nervous systems of thousands of neurons in real-time or faster, particularly networks which contain a heterogeneous mixture of neural and synaptic dynamics. These networks can be used for the control of any external system, which is also a novel capability for large-scale synthetic nervous system simulators. Additionally, I present the capability to optimize chemical synapses using gradient backpropagation, which is currently unused in machine learning but allows the learning of modulatory connections. With these tools in place, research can begin on the next generation of synthetic nervous systems as well as their application across a wide variety of domains.

Chapter 2

BACKGROUND

2.1 Synthetic Nervous Systems

As mentioned in Chapter 1, a common goal of neuroscientists and roboticists is to understand how animal nervous systems interact with their body and their environment in order to generate adaptive and intelligent behavior [24]. By understanding and modeling aspects of the nervous system, it is hoped that robots will one day be able to exhibit embodied intelligence [9] as well as animal-like robustness and adaptability [124]. One approach is to design Synthetic Nervous Systems (SNS), networks of conductance-based neurons and synapses which can be used to model animal nervous systems [109, 120] and control robots [5, 50, 64]. SNS networks are different from traditional artificial neural networks (ANNs) [32] or deep learning (DL) [78] systems, with the main differences coming from where dynamic behavior is implemented. In ANNs, particularly with recurrence [136], complex dynamics emerge as a network property if the network contains enough static neurons. In contrast, SNS networks are built with neurons which themselves are dynamic systems [10] and then integrated within larger networks. In this way, SNS networks are more similar to spiking neural networks (SNNs) [81], although they may contain either non-spiking neurons, spiking neurons, or a heterogenous mixture of neurons with different dynamics [50]. Additionally, synapses within SNS networks can also have their own dynamics [123], adding an additional layer of dynamic complexity.

2.2 Neural Simulation Software

A wide variety of software exists for simulating conductance-based neural dynamics [15, 51, 59, 141], and these simulators are capable of simulating highly detailed

and biologically accurate neural models. However, since these simulators were originally designed for the purpose of performing digital experiments and collecting data over a long simulation run it can be challenging to interface these simulators with external software and systems [39, 118]. Additionally, these simulators are limited to being run on conventional CPUs, although some have begun to be adapted for simulation on GPUs [2].

In the field of spiking neural networks (SNNs), many simulators have been developed which simulate large networks of spiking neurons and train them with principles from machine learning [42, 58, 89]. In general, training using modern methods is limited to networks which use the leaky integrate and fire model of a neuron. For pure simulation, simulators have also been designed to simulate the Izhikevich neural model [66] at scale [91]. While these simulators are capable of executing at high speed, they do so at the cost of limiting simulations to reduced models of spiking neurons and current-based synapses. Additionally, most do not support hybrid networks of neurons with heterogeneous dynamics, such as a mix of spiking and non-spiking neurons.

Multiple solutions have been developed which combine a neural dynamics simulator with a physics engine. One approach is to combine an existing neural simulator with an existing physics engine using a middleware memory management software [39, 131], allowing users who are comfortable with a specific neural simulator to interface their networks with physics objects at the expense of complicated software dependencies which are difficult to translate to other systems. The first integrated system was AnimatLab [28], which allows networks consisting of either a non-spiking or spiking neural model to control user-definable physics bodies to be simulated in a GUI with an integrated plotting engine. Numerous models have successfully been controlled with AnimatLab, both in simulation [27, 65, 120] and with robotic hardware [50, 64], however the reliance on a GUI and the software implementation makes it difficult to design larger networks [110]. The Neurorobotics

Platform (NRP) is a similar system to Animatlab in that it is large software suite which integrates multiple neural simulators with a physics engine in a cloud-based simulation environment [43], and has been used successfully for multiple neuro-robotic controllers in simulation [22, 84]. However, the NRP comes with significant overhead and is consequently unsuited for real-time control of robotic hardware.

Following developments in neuromorphic hardware which aims to accelerate spiking neural networks [33], software toolkits are beginning to emerge which enable the mapping of neural models to neuromorphic hardware [11, 77]. High-performance robotic controllers have been developed using these frameworks [29, 38], however as of the writing of this dissertation neuromorphic hardware is not widely available or affordable, and the solutions which exist do not support complex synaptic dynamics such as chemical reversal potentials [33].

2.3 Visual Motion Processing

One example of a problem which has potential solutions in both neuroscience and robotics is the processing of visual motion. Stabilizing the yaw motion of a mobile agent is a special case of visual odometry where the motion of a camera through a fixed world is calculated, also known as ego-motion estimation [71].

2.3.1 Drosophila melanogaster

A popular model organism for studying visual motion processing is the fruit fly *Drosophila melanogaster*, as it contains many of the same logical elements as that of the visual system in vertebrate animals [26] while using three orders of magnitude fewer neurons [14, 79]. Combining this reduction in scale with the extensive work in recent years to create a full *Drosophila* brain connectome [108, 137] makes the fruit fly a compelling inspiration for robotic implementation.

Within the fruit fly, the motion vision pathway is an extremely important system for adaptive behavior which aids in estimation of body motion and enabling rapid

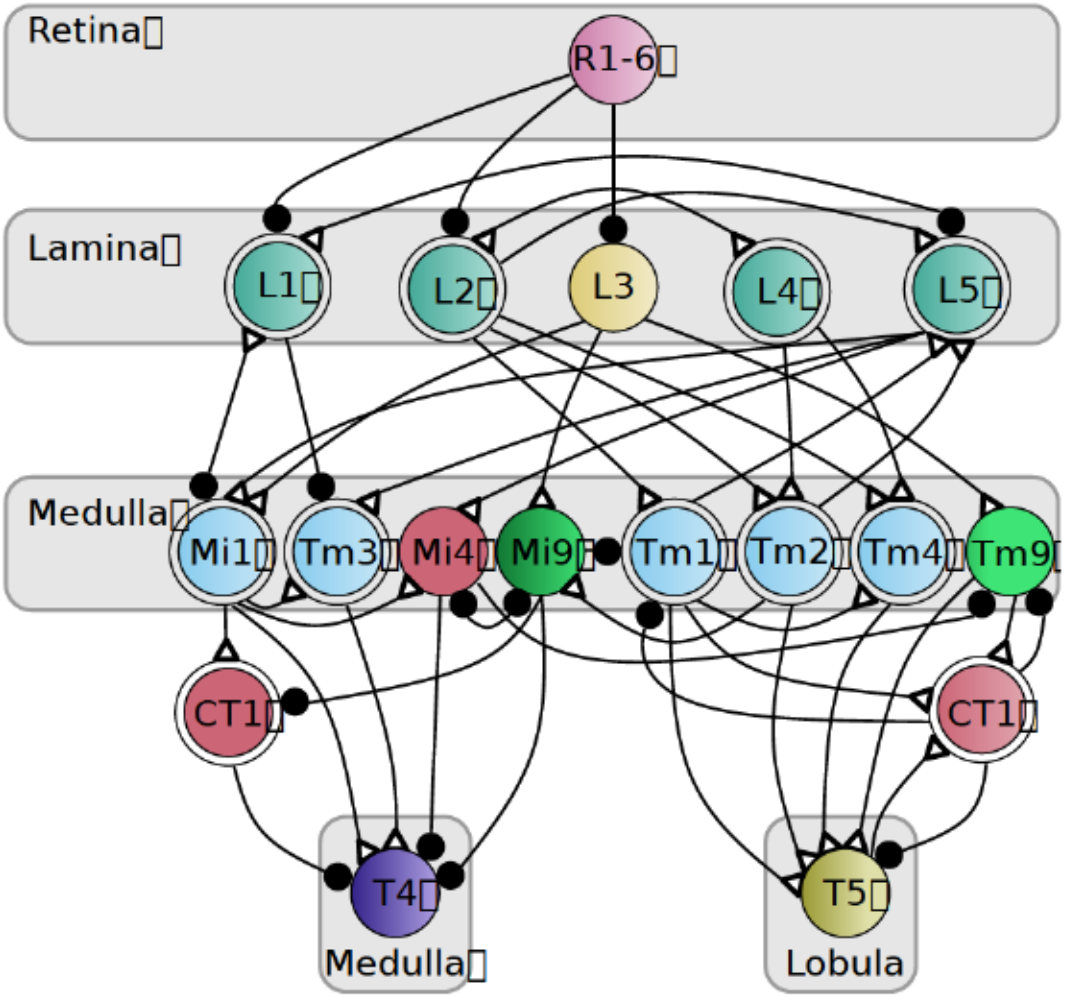


Figure 2.1: Neuroanatomy of the motion vision pathway in the *Drosophila melanogaster* optic lobe. Connectivity based on information presented in Borst et al. 2020 [14] and Braun et al. 2023 [17].

response to oncoming threats [1, 31, 48]. Within the Drosophila optic lobe, retinal and lamina cells convert changes in light intensity using spatiotemporal filters into information used in the rest of the network. Within the lamina, cells L1-L3 separate information flow into two pathways: an On pathway for encoding increases in brightness, and an Off pathway for encoding decreases in brightness [112, 125]. This transformed visual information is then filtered in the medulla by two banks of unique spatiotemporal filters (Mi1, Tm3, Mi4, Mi9 for the On pathway; Tm1, Tm2, Tm4, Tm9 for the Off pathway) [4, 40], which are then combined nonlinearly

Chapter 3

SNS-TOOLBOX: AN OPEN SOURCE TOOL FOR DESIGNING SYNTHETIC NERVOUS SYSTEMS

Material in this chapter has been previously published in

- Nourse, W. R., Szczecinski, N. S., & Quinn, R. D. (2022, July). SNS-Toolbox: A Tool for Efficient Simulation of Synthetic Nervous Systems. In Conference on Biomimetic and Biohybrid Systems (pp. 32-43). Cham: Springer International Publishing.
- Nourse, W. R., Jackson, C., Szczecinski, N. S., & Quinn, R. D. (2023). SNS-Toolbox: An Open Source Tool for Designing Synthetic Nervous Systems and Interfacing Them with Cyber–Physical Systems. Biomimetics, 8(2), 247.

Additionally, some material in this chapter was presented in person as the tutorial "An Introduction to Design and Simulation using SNS-Toolbox and SNSTorch" at the 2024 conference on Neuro Inspired Computational Elements (NICE) on April 26, 2024. Edits have been made to place this material into context with the rest of this dissertation.

3

One developing approach for robotic control is the use of networks of dynamic neurons connected with conductance-based synapses, also known as Synthetic Nervous Systems (SNS). These networks are often developed using cyclic topologies and heterogeneous mixtures of spiking and non-spiking neurons, which is a difficult proposition for existing neural simulation software. Most solutions apply to either one of two extremes, the detailed multi-compartment neural models in small networks, and the large-scale networks of greatly simplified neural models. In this

work, we present our open-source Python package SNS-Toolbox, which is capable of simulating hundreds to thousands of spiking and non-spiking neurons in real-time or faster on consumer-grade computer hardware. We describe the neural and synaptic models supported by SNS-Toolbox, and provide performance on multiple software and hardware backends, including GPUs and embedded computing platforms. We also showcase two examples using the software, one for controlling a simulated limb with muscles in the physics simulator Mujoco, and another for a mobile robot using ROS. We hope that the availability of this software will reduce the barrier to entry when designing SNS networks, and will increase the prevalence of SNS networks in the field of robotic control.

3 [illegible]

A common goal of neuroscientists and roboticists is to understand how animal nervous systems interact with biomechanics and their environment and generate adaptive behavior [24]. By understanding and modeling aspects of the nervous system, it is hoped that robots will be able to exhibit embodied intelligence [9] and exhibit animal-like robustness and adaptability [124]. One approach is to design Synthetic Nervous Systems (SNS), networks of conductance-based neurons and synapses which can be used to model animal nervous systems [109, 120] and control robots [5, 50, 64]. Some strengths of SNS networks include that they can be tuned using analytic design rules [121, 123] and that results obtained controlling robotic hardware can propose neurobiological hypotheses [83, 134].

In order to design SNS networks for robotic control, software tools are needed. Software for simulating SNS networks should support conductance-based modeling of neurons and synapses, as there are elements of neural behavior in conductance-based models which are incompatible with current-based models [100, 121]. Bidirectional synaptic links, such as electrical synapses, should also be supported [53]. Simulators should also support networks with heterogeneous neural models, poten-

tially containing both spiking and non-spiking neurons [50]. While individual spiking neurons can be more computationally powerful than non-spiking neurons [81], non-spiking neurons are capable of capturing much of the dynamics of populations of spiking neurons while being more amenable to real-time simulation [130]. Networks should be able to be constructed in a programmatic way, in order to aid the design of large but formulaic networks [110]. SNS networks should be able to be simulated with faster than real-time performance using CPUs and GPUs, and the same networks should be easily interfaced with physics simulation engines and robotic hardware. Additionally, for accessibility and ease of use in laboratory and educational settings, a simulator software should be cross-platform compatible with the Windows (trademark Microsoft Corporation, Redmond, WA, USA), MacOS (trademark Apple Corporation, Cupertino, CA, USA), and Linux operating systems. A selected survey of available simulation software is presented in Table 3.1.

Software for simulating conductance-based neural dynamics have long been available, with the most popular options being NEURON [59], NEST [49], GENESIS [15], and Brian [51]. These simulators are capable of simulating highly detailed and biologically accurate neural models, however they were originally designed for the purpose of performing digital experiments and collecting data over a long simulation run. As such, interfacing with external software and systems can be challenging [118] and often requires dedicated software for memory management [39]. Additionally, these simulators are limited to being run on conventional CPUs, although some have begun to be adapted for use with GPUs [2].

Other simulators are capable of designing large networks of neurons using principles from machine learning, such as snnTorch [42], SpykeTorch [89], and BindsNET [58]. These simulators are capable of executing at high speed on both CPUs and GPUs, but they do so at the cost of limiting simulations to reduced models of spiking neurons and current-based synapses. Simulators have also been designed to simulate the Izhikevich neuron [66] and other spiking neurons at scale, including

Software	No GUI Required	Real-Time Capable	Non-Spiking & Spiking	Chemical Synapses	Electrical Synapses	CPU/GPU Support	Cross-Platform*
AnimatLab		X	X	X	X		
NRP			X	X†	X†	X†	X
Nengo	X	X	X	X†	X†	X†	X
snnTorch	X	X				X	X
Spyke-Torch	X	X				X	X
BindsNET	X	X				X	X
Brian2	X		X	X	X	X	X
NEURON	X		X	X	X	X‡	X
NEST	X	X			X	X‡	X
ANNarchy	X	X	X	X	X	X‡	
SNS-Toolbox	X	X	X	X	X	X	X

Table 3.1: Comparison of neural simulation software. Note that due to the many simulators available, not all are presented in this table. * To be considered cross-platform compatible, the software must be easily run on Linux, MacOS, and Windows. † To implement some features, custom code must be implemented which is incompatible with the rest of the Nengo ecosystem. ‡ Limited GPU support is currently available.

CARLSim [91], NEMO [45], GeNN [138], CNS [90], and NCS6 [60]; however they typically do not support hybrid networks of spiking and non-spiking neurons.

Multiple solutions have been developed which combine a neural dynamics simulator with a physics engine. One approach is to combine an existing neural simulator with an existing physics engine using a middleware memory management software. This has been used to combine Brian [51] and SOFA [3] using CLONES [131], as well as NEST [49] with Gazebo [73] using MUSIC [39]. This approach allows users who are comfortable with a specific neural simulator to interface their networks with physics objects. However, it leads to complicated software dependencies which are difficult to translate to other systems. The first integrated system

was AnimatLab [28], which allows networks consisting of either a non-spiking or spiking neural model to control user-definable physics bodies. It also comes with an integrated plotting engine, allowing users to run experiments and analyze data within a single application. Numerous models have successfully been controlled with AnimatLab, both in simulation [27, 65, 120] and with robotic hardware [50, 64], however networks cannot be designed in a programmatic way and are difficult to scale to larger networks [110]. The Neurorobotics Platform (NRP) is a large software suite which integrates multiple neural simulators, including Nengo [11] and NEST [49], with Gazebo [73] in a cloud-based simulation environment. The NRP is a comprehensive toolbox which comes with a variety of advanced visualization tools, and has been used successfully for multiple neurorobotic controllers in simulation [22, 84]. However, the NRP comes with much overhead and, as such, is unsuited for real-time control of robotic hardware.

In recent years, high-performance robots have been developed which are controlled using networks of spiking neurons [29, 38]. These networks achieve state-of-the-art performance, but rely on specialized neuromorphic hardware, such as Intel's Loihi processors [33], which are not yet widely available. Lava [77] is a relatively recent and promising software solution for designing spiking networks, but it is primarily designed for use with CPUs and Loihi. Currently, the most widely used software for implementing spiking networks and controlling real hardware is Nengo [11], which has achieved impressive results [38]. However, Nengo is optimized for networks designed using the Neural Engineering Framework [41], and can have reduced performance without the use of neuromorphic hardware. One simulator which can simulate networks with a mixture of spiking and non-spiking neurons in real-time or faster is ANNarchy [130], which does so using a C++ code generation system. However, this code generation system which enables high performance comes at the cost of incompatibility with the Microsoft Windows operating system, which reduces its level of accessibility.

Here, we present SNS-Toolbox, an open-source Python package for the design and simulation of synthetic nervous systems. SNS-Toolbox allows users to design SNS networks with a simple interface and simulate them using established numerical processing libraries on consumer-grade hardware. We focus on simulating a specified set of neural and synaptic dynamics, without dedicated ties to a GUI or a physics simulator. This focus allows the SNS-Toolbox to be easily interfaced with other systems, and for a given network design to be able to be reused without modification in multiple contexts. In previous work [94], we presented an initial version of SNS-Toolbox with reduced functionality. Here we explain in detail the expanded neural and synaptic dynamics supported in the toolbox, and describe the workflow for designing and simulating networks. We provide results which demonstrate comparative performance with other neural simulators, and we showcase the use of SNS-Toolbox in two different applications, motor control of a muscle-actuated biomimetic system in Mujoco [127], and navigation control of a robotic system in simulation using the Robotic Operating System (ROS) [99].

3.3 Materials and Methods

Herein we describe the internal functionality of the SNS-Toolbox, how different neurons and synapses are simulated, designed, and compiled by the user. Section 3.3.1 defines the neural models which are supported in the toolbox, and Section 3.3.2 does the same for connection types. Section 3.4.2 describes the design process using SNS-Toolbox, and how a network is compiled and simulated.

All software described is written in Python [105], which was chosen due to its ease of development and wide compatibility. Unless otherwise specified, the units for all quantities are as follows, current (nA), voltage (mV), conductance (S), capacitance (nF), and time (ms).

3.3.1 Neural Models

SNS-Toolbox is designed to simulate a small selection of neural models, which are variations of a standard leaky integrator. In this section, we present the parameters and dynamics of each neural model which can be simulated using SNS-Toolbox.

Non-Spiking Neuron

The base model for all neurons in SNS-Toolbox is the non-spiking leaky integrator, as has been used in continuous-time recurrent neural networks [10]. This neural model can be used to model non-spiking interneurons, as well as approximate the rate-coding behavior of a population of spiking neurons [121]. The membrane potential behaves according to the differential equation

—— (3.1)

where is the membrane capacitance, the membrane conductance, and is the resting potential of the neuron. is an injected current of constant magnitude, and is any external applied current. is the current induced via synapses from presynaptic neurons, the forms of which are defined in Section 3.3.2.

During simulation, the vector of membrane potentials _ is updated at each step by representing Equation (3.1) in a forward-Euler step:

_ _ _ _ _ _ _ _ _ (3.2)

where denotes the element-wise Hadamard product, and represents the simulation timestep. is the membrane time factor, which is set as ——.

Spiking Neuron

Spiking neurons in SNS-Toolbox are represented as expanded leaky integrate-and-fire neurons [88], with the membrane depolarization dynamics described in Equa-

tion (3.1) and an additional dynamical variable for a firing threshold [123],

$$\text{—} \qquad _0 \tag{3.3}$$

where is a threshold time constant, and $_0$ is the initial threshold voltage. is a proportionality constant which describes how changes in affect the behavior of , with 0 causing to always equal $_0$. When the neuron is subjected to a constant stimulus, 0 results in a firing rate which decreases over time, and 0 causes a firing rate which increases over time. Spikes are represented using a spiking variable ,

$$\begin{array}{l} 1 \\ 0 \quad \text{otherwise} \end{array} \tag{3.4}$$

which also triggers the membrane potential to return to rest:

$$\text{if} \quad 1 \qquad . \tag{3.5}$$

The vector of firing thresholds _ is updated as

$$\text{–} \quad \text{–} \quad \text{–} \quad \text{–} \quad \text{–}_0 \ \text{—} \quad \text{—} \quad \text{—} \tag{3.6}$$

where — is the threshold time factor. Based on the threshold states, the spiking states are updated as

$$\text{–} \qquad 0 \ \text{–} \quad \text{–} \tag{3.7}$$

Note that for simplified implementation, all spikes with SNS-Toolbox are internally represented as impulses of magnitude 1. Using these spike states, the membrane potential of each neuron which spiked is reset to $_{\text{rest}}$

$$\text{—} \quad \text{—} \quad \text{—} \quad \text{–} \quad 1 \quad \text{—} \tag{3.8}$$

Neuron with Voltage-Gated Ion Channels

The other neural model available within SNS-Toolbox is a non-spiking neuron with additional Hodgkin–Huxley [61] style voltage-gated ion channels. The membrane dynamics are similar to Equation (3.1), with the addition of an ionic current

[122]:

— (3.9)

This ionic current is the sum of multiple voltage-gated ion channels, all obeying the following structure:

(3.10)

Any neuron within a network can have any number of ion channels. is the maximum ionic conductance of the j ion channel, and is the ionic reversal potential. and are dynamical gating variables, and have the following dynamics

— ———— (3.11)

where functions of the form are a voltage-dependent steady-state

$$\frac{1}{1 \quad \exp} \tag{3.12}$$

and is a voltage-dependent time constant

$$\frac{}{\exp} \tag{3.13}$$

denotes an exponent, and is the gate reversal potential. and are parameters which shape the and functions. is the maximum value of . Note that depending on the desired ion channel, the exponent for various sections can be set to 0 in order to effectively remove it from Equation (3.10). One particular example of this is a neuron with a persistent sodium current, which is also available as a preset in SNS-Toolbox,

(3.14)

which is the same as Equation (3.10) with one dynamic variable disabled and some variable renaming.

3.3.2 Connection Models

Within SNS-Toolbox, neurons are connected using connection objects. These can either define links between individual neurons, or structures of connectivity between neural populations (see Section 3.3.2).

Non-Spiking Chemical Synapse

When connecting non-spiking neurons, non-spiking chemical synapses are typically used. The amount of synaptic current to post-synaptic neuron from presynaptic neuron is

(3.15)

where is the synaptic reversal potential. is the instantaneous synaptic conductance, which is a function of the presynaptic voltage :

0 ——— (3.16)

is the maximum synaptic conductance, and voltages and define the range of presynaptic voltages where the synaptic conductance depends linearly on the presynaptic neuron's voltage.

When simulating, Equation (3.15) is expanded to use matrices of synaptic parameters (denoted in bold),

_ $\mathbf{G_{syn}}$ E _ $\mathbf{G_{syn}}$ (3.17)

and each term is summed column-wise to generate the presynaptic current for each neuron. Synaptic parameter matrices have an NxN structure, with the columns corresponding to the presynaptic neurons and the rows corresponding to the postsynaptic neurons. Equation (3.16) is also expanded to use parameter matrices,

$\mathbf{G_{non}}$ 0 $\mathbf{G_{max\ non}}$ $= \dfrac{\mathbf{E_{lo}}}{\mathbf{E_{hi}}\ \mathbf{E_{lo}}}\ \mathbf{G_{max\ non}}$ (3.18)

Spiking Chemical Synapse

Spiking chemical synapses produce a similar synaptic current as non-spiking chemical synapses (Equation (3.15)), but a key difference is that is a dynamical variable defined as

——— (3.19)

if 1 . (3.20)

The conductance is reset to , the maximum value, whenever the presynaptic neuron spikes. Otherwise, it decays to zero with a time constant of . When simulated, these dynamics are represented as

$$\mathbf{G_{spike}} \quad \mathbf{G_{spike}} \quad 1 \quad 1 \quad \mathbf{T_{syn}} \tag{3.21}$$

where $\mathbf{G_{spike}}$ is the matrix of spiking synaptic conductances, and $\mathbf{T_{syn}}$ is the synaptic time factor matrix.

An additional feature available with spiking synapses is a synaptic propagation delay. This sets the number of simulation steps it takes for a spike to travel from one neuron to another using a specific synapse, a feature which is useful for performing some aspects of temporal computation [110]. If the synapse between neurons and has a delay of timesteps, the delayed spike is represented as

(3.22)

For simulation, this propagation delay is implemented using a buffer matrix $_{\text{buffer}}$ with columns and rows, where is the longest delay within the network. The rows of $_{\text{buffer}}$ are shifted down at each timestep, and the first row is replaced with current spike state vector _ . $_{\text{buffer}}$ is then transformed into a matrix of delayed spikes $_{\text{delay}}$ by rearranging based on the delay of each synapse in the network. $_{\text{delay}}$ is then used to simulate the synaptic reset dynamics from Equation (3.20),

$$\mathbf{G_{spike}} \quad \mathbf{G_{spike}} \quad \mathbf{G_{max\ spike}} \tag{3.23}$$

Electrical Synapses

Electrical synapses, or gap junctions, are resistive connections between neurons that do not use synaptic neurotransmitters. As a result, the neurons exchange current proportional to the difference between their voltage values. Their current is defined as

(3.24)

where is the synaptic conductance. Electrical synapses are simulated in SNS-Toolbox using a similar formulation as Equation (3.17),

$$\_ \quad \mathbf{G_{elec}} \quad \_ \quad \_ \quad \mathbf{G_{elec}} \tag{3.25}$$

SNS-Toolbox simulates electrical synapses as bidirectional by default, where current can flow in either direction between the connected neurons. Rectified connections are also supported, where current only flows from the presynaptic to the postsynaptic neuron and only if . When simulating with rectified electrical synapses, a binary mask $\mathbf{M}$ is generated,

$$\mathbf{M} \quad \_ \quad \_ \tag{3.26}$$

where denotes outer subtraction. The voltage of each neuron is subtracted in a pairwise fashion, with the result processed by the heaviside step function . This generates a matrix where each element is 1 if current is allowed to flow in that direction, and 0 otherwise. This binary mask is then applied to a synaptic conductivity matrix $\mathbf{G_{rec}}$ to obtain the masked conductance $\mathbf{M_G}$,

$$\mathbf{M_G} \quad \mathbf{M} \quad \mathbf{G_{rec}} \tag{3.27}$$

To generate the opposite current flow in rectified synapses, the masked conductance is then added to its transpose with the diagonal entries removed,

$$\mathbf{M_D} \quad \mathbf{M_G} \quad \mathbf{M_G} \quad \text{diag} \quad \mathbf{M_G} \tag{3.28}$$

This final masked, transformed conductance matrix is substituted for $\mathbf{G_{elec}}$ in Equation (3.25)

$$\_ \quad \mathbf{M_D} \quad \_ \quad \_ \quad \mathbf{M_D} \tag{3.29}$$

Matrix and Pattern Connections

In their base form, each of the preceding connection models defines the connection between two individual neurons. However, the connections' behavior can be extended to defining connections between populations of neurons. Following the model presented in [123], in the simplest form of population-to-population connection all neurons become fully connected and the synaptic conductance is automatically scaled such that the total conductance into each postsynaptic neuron is the same as the original synapse.

For more complex desired behavior, more types of population-to-population connections are available. Matrix connections allow the user to specify the exact matrices for each synaptic parameter, and one-to-one connections result in each presynaptic neuron to be connected to exactly one postsynaptic neuron, with all synapses sharing the same properties. Pattern connections are also available, modeled after convolutional connections in ANNs [78]. In pattern connections, a kernel matrix $\mathbf{K}$ can be given,

$$\mathbf{K} \tag{3.30}$$

where the indices are values for a single synaptic parameter ( ). If $\mathbf{K}$ describes the connection pattern between two 3 3 neural populations, then the

resulting synaptic parameter matrix **P** will present the following structure:

$$\mathbf{P} = \begin{bmatrix} & 0 & & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 & 0 \\ 0 & & 0 & & 0 & 0 & 0 \\ & 0 & & 0 & & & 0 \\ 0 & & 0 & & 0 & & \\ 0 & 0 & 0 & & 0 & & 0 \\ 0 & 0 & 0 & & & & \\ 0 & 0 & 0 & 0 & & 0 & \end{bmatrix} \tag{3.31}$$

3.3.3 Inputs and Outputs

In order for an SNS to interact with external systems, it must be capable of receiving inputs and sending outputs. For applying external stimulus to a network, input sources can be added. These sources can be either individual elements or a one-dimensional vector, and are applied to the network via _ ,

$$\_ \qquad \mathbf{C_{in}} \ \_ \tag{3.32}$$

where $\mathbf{C_{in}}$ is an LxN binary masking matrix which routes each input to the correct target neuron. L is the number of input elements, and N is the number of neurons in the network. This external input vector is varied from step to step and could come from any source (e.g., static data, real-time sensors).

Output monitors can also be added, both for sending signals to other systems and for observing neural states during simulation. These outputs are assigned one-to-one to each desired neuron, meaning one output applied to a population of five neurons results in five individual outputs. Output monitors can be voltage-based or spike-based, where the output is the direct voltage or spiking state of the source

neuron. During simulation, the output vector is computed as

$$\_\_\_ \quad \mathbf{C}_{\text{out voltage}} \_ \quad \mathbf{C}_{\text{out spike}} \_ \tag{3.33}$$

where $\mathbf{C}_{\text{out voltage}}$ and $\mathbf{C}_{\text{out spike}}$ are connectivity matrices for the voltage and spike-based monitors, respectively.

3.3.4 Software Design and Workflow

Using SNS-Toolbox, the design and implementation of an SNS is split across three phases, a design phase, a compilation phase, and a simulation phase.

Design

To design a network, users first define neuron and connection types. These describe the parameter values of the various neural and synaptic models in the network, which can be subsequently reused. Once the neuron and connection presets are defined, they can be incorporated into a Network object (for a complete inventory of the different elements which can be added to a Network, refer to Sections 3.3.1–3.3.3). First, the user can add populations of neurons by giving the neuron type, the size or shape of the population, and a name to refer to the population. When simulated, all populations will be flattened into a one-dimensional vector, but during the design process they can be represented as a two-dimensional matrix for ease of interpretation (e.g., working with two-dimensional image data). After populations are defined and labeled, the user can add synapses or patterns of synapses between neurons/populations, giving an index or character-string corresponding to the source and destination neurons or populations and the connection preset.

Once a network is designed, it can also be used as an element within another network. In this way, a large network can be designed using large collections of predefined subnetworks, in a methodology referred to as the Functional Subnetwork Approach (FSA). Available within the SNS-Toolbox is a collection of subnetworks

which perform simple arithmetic and dynamic functions. For a complete explanation of these networks, as well as the FSA, please refer to [121].

Compilation

While it describes the full structure of an SNS, a Network object is merely a dictionary which contains all of the network parameters. In order to be simulated, it must be compiled into an executable state. Given a Network, the SNS-Toolbox can compile a model capable of being simulated using one of the four software backends, NumPy [56], PyTorch [95], a PyTorch-based sparse matrix library (torch.sparse), and an iterative evaluator which evaluates each synapse individually. These backends are all built on well-established numerical processing libraries, with PyTorch bringing native and simple GPU support. Each backend has different strengths and weaknesses, which are illustrated in Section 3.4.2. Although each is different, all backends are compiled following the general procedure described in Algorithm 1. Once a network is compiled, it can either be immediately used for simulation or saved to disk for later use.

Algorithm 1 General compilation procedure.

```
function COMPILE(net,    )
    Get network parameters
    Initialize state and parameter vectors and matrices
    Set parameter values of each neuron in each population
    Set input mapping and connectivity parameter values
    Set connection synaptic parameter values
    Calculate time factors
    Initialize propagation delay buffer
    Set output mapping and connectivity parameter values
    return model
end function
```

Simulation

Since the SNS-Toolbox focuses on smaller networks which are connected with varying levels of feedback loops [10, 50, 63, 64, 96] instead of multiple massively

connected layers [46], we optimize our computations by representing all networks as single-layer, fully-connected recurrent networks. During simulation, the neural dynamics are evaluated by unfolding the network through time. This is similar to the method developed by Werbos et al., for training recurrent ANNs [136]. See Figure 3.1 for a visual representation of this strategy. At each timestep, every neuron can receive input from any neuron at the previous step (including itself via an autapse [113]). Although the SNS-Toolbox only acts as a neural dynamics simulator, it is extensible to interact with other systems for controlling robot (Section 3.4.4) or musculoskeletal (Section 3.4.5) dynamics.

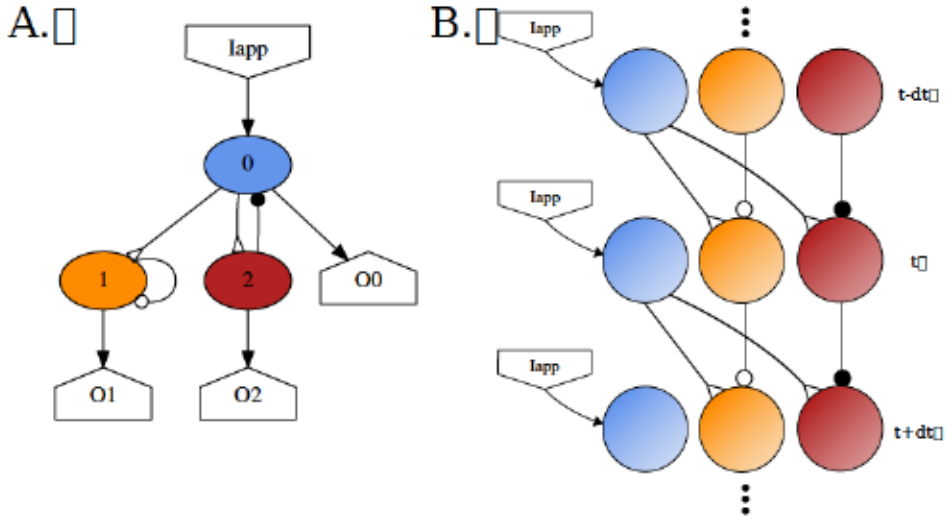


Figure 3.1: Simulation method for a small example network using the SNS-Toolbox. (**A**) Overall network diagram generated within the toolbox. (**B**) Diagram of the general computational flow when simulating the network. The network is unfolded in time, and neural voltages are propagated in feedforward layers from one time-step to another.

3.4 Results

In this Section, we provide results showcasing the capabilities of SNS-Toolbox. We first provide quantitative benchmarks which characterize the performance of the

software, and conclude with two application examples.

3.4.1 Specifications

Unless otherwise specified, all of the following results were obtained using the software and hardware presented in Table 3.2.

Table 3.2: Software and hardware specifications.

Item	CPU	GPU	RAM	Python	NumPy	PyTorch	CUDA
Specification	AMD Ryzen 9 3900X	NVIDIA RTX2060	32 GB DDR4 2400 MHz	3.8.10	1.21.1	1.9.0	11.5

3.4.2 Performance Benchmarking

For evaluating the performance of the SNS-Toolbox, we present benchmarking results for varying network size, structure, and type. In these benchmarks, networks consist entirely of either spiking or non-spiking neurons, and are either densely or sparsely connected. In densely connected networks, every neuron is synaptically connected to every other neuron. For sparse networks, the neurons are connected with the following structure; 8% of neurons receive external input, 12% of neurons are recorded for output, and the number of neurons and synapses is equal. This structure is based on general principles observed in previous large-scale synthetic nervous systems [50, 64].

Maximum Network Size

Networks were constructed following the structure described in Section 3.4.2, and increased in size until one of the following two termination conditions were met, either the network parameter matrices could not fit in memory or network synthesis took an excessive amount of time ($\geq$10 h). These experimental results are shown in Table 3.3. The limiting factors of whether a network can successfully be synthesized are the synaptic parameter matrices, as these increase in size quadratically as the

size of the network increases. CPU-based backends are able to achieve the highest network sizes, which is expected due to the increased volume of memory available to the CPU. The iterative backend is able to achieve the highest sizes of network, since its neural and synaptic dynamics are computed by iterating over one-dimensional arrays instead of vector and matrix operations on two-dimensional arrays. All of the sparse networks took significantly longer to synthesize, resulting in termination of their testing before running out of memory.

Table 3.3: Maximum network size.

Backend	Iterative	NumPy	Torch (CPU)	Torch (GPU)	Sparse (CPU)	Sparse (GPU)
Max Dense	11,010	20,010	22,000	7865	151*	2510*
Max 1:1	158,010	23,010	24,010	7639	17,510*	11,120*

* Larger networks can be simulated, but compilation takes excessive time.

Backend Performance

We show that SNS-Toolbox is capable of simulating thousands of non-spiking neurons in real-time or faster, with slower performance when simulating spiking neurons. In total, 100 networks, which varied in size from 10 to 5000 neurons in a logarithmic spacing, were generated and simulated for 1000 steps in each backend. A simulation step of $\Delta t = 0.1$ ms was used. The elapsed time to simulate each step was recorded, and the results are shown in Figure 3.2. Each of the available backends exhibit different strengths and weaknesses. For networks with less than 100 neurons, the Numpy [56] backend runs the fastest, followed by the PyTorch [95] backend running on the CPU. Once networks increase in size beyond 200–300 neurons, the PyTorch backend running on a GPU becomes the fastest. While this backend is the fastest, the exchange of data between the CPU and GPU results in a higher degree of temporal variability than the CPU-based backends. Further investigation is needed to reduce this variability in performance.

The exact threshold for what could be considered real-time performance is dependent on the simulation step size, which, in turn, is dependent on the membrane properties of neurons within the network. While all networks in this test were simulated with the same step size for consistency, accurately simulating spiking networks will generally require a finer simulation step than non-spiking networks. In this test, an elapsed time of 0.1 ms per step is considered real-time for the spiking networks. A step size of 1 ms would suffice for the non-spiking networks tested in this section, so their real-time limit is 1 ms. For non-spiking networks, this means that networks up to about 3000 neurons can be simulated in real-time, and for spiking networks the threshold is about 100–200 neurons.

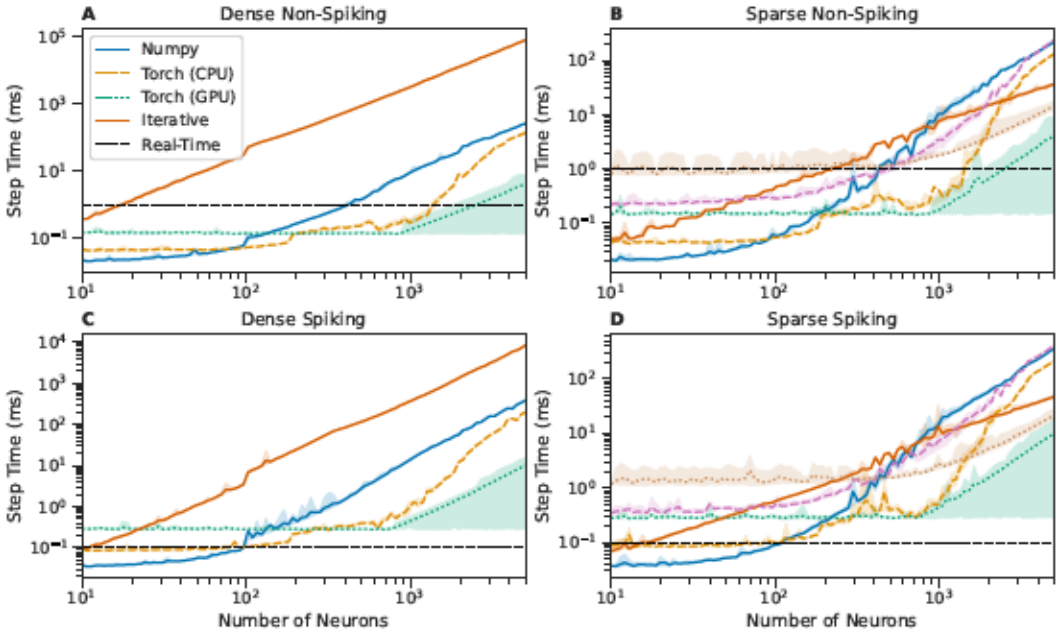


Figure 3.2: Comparison of wall-clock times to simulate a network for one simulation time-step over varying network sizes, using the six software backends provided in SNS-Toolbox. **(A,B)**: Networks of non-spiking neurons, **(C,D)**: networks of spiking neurons. **Left**: Fully-connected networks, **Right**: Sparsely connected networks, following the structure described in Section 3.4.2. Lines denote the mean over 1000 steps, shaded region denotes the area between the fifth and ninety-fifth percentiles. The real-time limit is denoted with a horizontal dashed black line.

The exact threshold for what could be considered real-time performance is dependent on the simulation step size, which, in turn, is dependent on the membrane properties of neurons within the network. While all networks in this test were simulated with the same step size for consistency, accurately simulating spiking networks will generally require a finer simulation step than non-spiking networks. In this test, an elapsed time of 0.1 ms per step is considered real-time for the spiking networks. A step size of 1 ms would suffice for the non-spiking networks tested in this section, so their real-time limit is 1 ms. For non-spiking networks, this means that networks up to about 3000 neurons can be simulated in real-time, and for spiking networks the threshold is about 100–200 neurons.

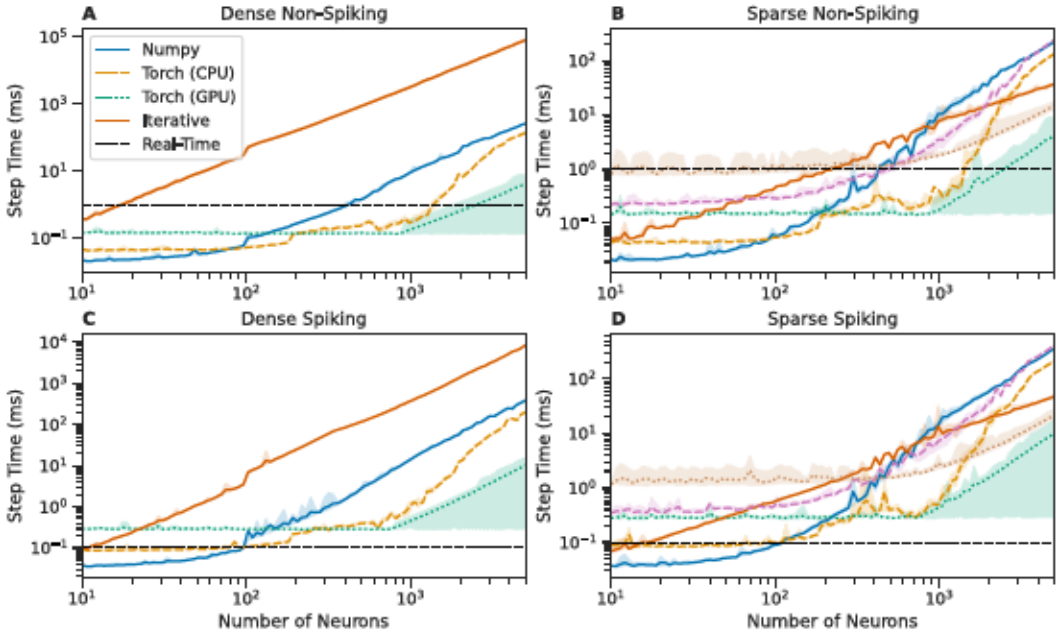


Figure 3.2: Comparison of wall-clock times to simulate a network for one simulation time-step over varying network sizes, using the six software backends provided in SNS-Toolbox. **(A,B)**: Networks of non-spiking neurons, **(C,D)**: networks of spiking neurons. **Left**: Fully-connected networks, **Right**: Sparsely connected networks, following the structure described in Section 3.4.2. Lines denote the mean over 1000 steps, shaded region denotes the area between the fifth and ninety-fifth percentiles. The real-time limit is denoted with a horizontal dashed black line.

Benchmarking Alternative Software

The SNS-Toolbox is faster than the majority of similar neural simulators. We perform the same testing procedure presented in Section 3.4.2, and compare against the behavior of similar simulators, namely Brian2 [51], Nengo [11], and ANNarchy [130]. For these other simulators, the neural and synaptic dynamics for basic spiking and non-spiking neurons within SNS-Toolbox (see Sections 3.3.1 and 3.3.2) were implemented and verified to match the behavior in SNS-Toolbox. In Brian2 and ANNarchy, this was completed via their built-in interfaces for interpreting custom behavioral strings. This process was less straightforward in Nengo, requiring a custom Nengo process object which re-implemented the equations as performed in SNS-Toolbox. As such, while the networks are able to successfully run in Nengo, they are not fully compatible with the rest of the Nengo ecosystem. Since these benchmarks are not being compared against biological recordings, validation is completed by comparing the behavior of the neural models across simulators and verifying that the simulation recordings are identical.

Results are shown in Figure 3.3. For clarity, simulators with multiple backends or variants are condensed to show the best performing version for each network size. The variants tested in Brian2 are the normal version on CPU, and the GPU-accelerated Brian2CUDA [2], and ANNarchy was compiled using the CPU and GPU paradigms. All SNS-Toolbox backends were tested. Across all network sizes and structures, SNS-Toolbox is faster or within performance variance of Brian2 and Nengo. SNS-Toolbox is faster than ANNarchy for some densely-connected non-spiking networks, but, in general, is slower but competitive across the test suite. Suggestions for improving this speed discrepancy will be explored in the Discussion.

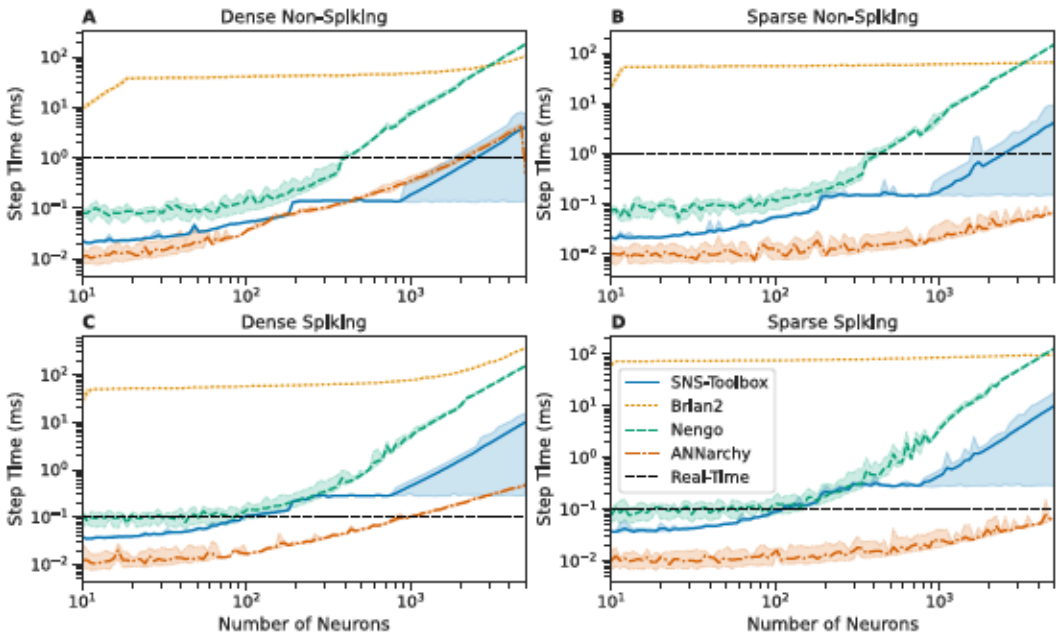


Figure 3.3: Comparison of wall-clock times for SNS-Toolbox to simulate a network for one simulation time-step over varying network sizes, using SNS-Toolbox and three other neural simulators (Brian2 [51], Nengo [11], and ANNarchy [130]). For the following simulators, the time data presented are chosen as the best-performing backend variant, Brian2, standard Brian2, and the GPU-accelerated Brian2CUDA; SNS-Toolbox, all available variants; and ANNarchy, CPU-based compilation, and GPU-based compilation. (**A**,**B**): Networks of non-spiking neurons, (**C**,**D**): networks of spiking neurons. **Left**: Fully-connected networks, **Right**: Sparsely connected networks, following the structure described in Section 3.4.2. Lines denote the mean over 1000 steps, shaded region denotes the area between the fifth and ninety-fifth percentiles. The real-time limit is denoted with a horizontal dashed black line.

Performance on Embedded Hardware

The testing procedure presented in Section 3.4.2 is again repeated, testing the performance of SNS-Toolbox on various embedded computing platforms. These included a Raspberry Pi Model 3B (trademark Raspberry Pi Limited, Cambridge, UK), Jetson Nano 4GB (trademark NVIDIA Corporation, Santa Clara, CA, USA), and an Intel NUC SWNUC11PHKi7c00 (trademark Intel Corporation, Santa Clara, CA, USA) with 32 GB of RAM. Due to the reduced available memory available on the Raspberry Pi and Jetson, network size is varied logarithmically from 10 to 1000

neurons, instead of the 10–5000 neurons in Sections 3.4.2 and 3.4.2. Results are shown in Figure 3.4; for clarity, all backends are condensed for each device such that the best performing solution at each network size is presented. The Raspberry Pi performs comparably with a Jetson Nano, with the Jetson exhibiting slightly better performance across all network sizes. The amount of memory available on the Raspberry Pi is the smallest of the three devices, so it is unable to simulate densely-connected networks over approximately 900 neurons in size. The Intel NUC is a significantly more powerful computing platform than the Raspberry Pi or the Jetson Nano, and accordingly behaves more closely to desktop-level performance.

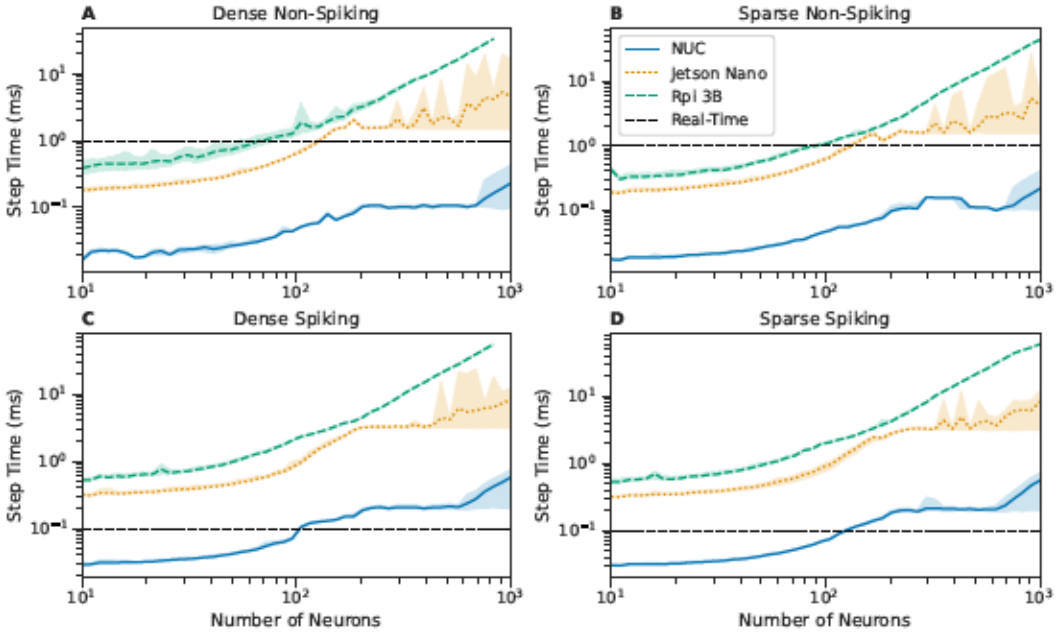


Figure 3.4: Comparison of wall-clock times to simulate a network for one simulation time-step over varying network sizes, using SNS-Toolbox on three different embedded computing platforms (Intel NUC, Raspberry Pi version 3b, and an NVIDIA Jetson Nano). The time data presented are chosen as the best-performing backend variant at each network size, with GPU-based backends excluded on the Raspberry Pi. (**A**,**B**): Networks of non-spiking neurons, (**C**,**D**): networks of spiking neurons. **Left**: Fully-connected networks, **Right**: Sparsely connected networks, following the structure described in Section 3.4.2. Lines denote the mean over 1000 steps, shaded region denotes the area between the fifth and ninety-fifth percentiles. The real-time limit is denoted with a horizontal dashed black line.

3.4.3 Basic Demonstration

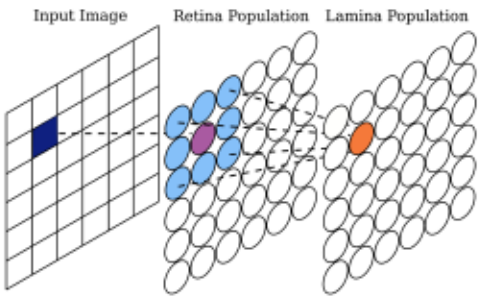
Using the SNS-Toolbox, we implemented a network which models an anatomical circuit and performs a useful function, but would be complicated and tedious to design by hand. In the *Drosophila melanogaster* nervous system, the optic lobe contains circuits for processing motion in the visual field[12, 85]. The first two layers of this visual system are the retina and the lamina, which perform two distinct visual processing operations. Retinal neurons R1-R6 encode incoming photons as changing neural activity[26], and the lamina primarily consists of two pathways: the L2 neurons have an antagonistic center-surround receptive field, and L1 neurons have a traditional center-surround receptive field[47]. Based on this structure, modeling the retina and L1 cells results in a circuit which performs high-pass filtering of an input image. Previous work has created a synthetic nervous system model of the *Drosophila* optic lobe motion circuitry[110] using the Animatlab software[28]. This model was constrained to one-dimensional images, in part due to the difficulty of implementing the model. In this previous approach neurons and synapses have to be placed and routed by hand, which is time-intensive and tedious to produce a simplified model with 510 neurons and 1574 synapses for one-dimensional images[110]. To demonstrate the effectiveness of the SNS-Toolbox, we created a model of the retina and L1 cells of the optic lobe capable of processing two-dimensional images, consisting of 2048 neurons and 8476 synapses and generated in only 18 lines of Python code (see Figure 3.5A).

To ensure the network responds to input in a realistic (within simulation) time, we examine the temporal properties of the *Drosophila* optic lobe. Based on flicker fusion studies, the response rate of the fruit fly to visual stimuli is in the range of 60-100 Hz[87]. Approximating the settling time of each neuron to be 4 , the maximum time constant of the entire visual processing system is 40-66.67 ms. Since our two-layer circuit is not the entire visual system, we will reduce these estimates by an order of magnitude and set the membrane time constant of each neuron in the

A.

```
# General network
R = 20.0    # range of network activity (mV)
neuron_type = NonSpikingNeuron()    # generic neuron type
net = Network(name='Visual Network')    # create an empty network

# Retina
# add a 2d population the same size as the scaled image
net.add_population(neuron_type,shape,name='Retina')
# add a vector input for the flattened image
net.add_input('Retina', size=flat_size,name='Image')
# add a vector output from the retina
net.add_output('Retina',name='Retina Output')

# Lamina
net.add_population(neuron_type,shape,name='Lamina')

e_ex = 160.0    # excitatory reversal potential
e_in = -80.0    # inhibitory reversal potential
k_ex = 1.0  # excitatory gain
k_in = -1.0/9.0 # inhibitory gain
# calculate excitatory conductance
g_max_ex = (k_ex*R)/(del_e_ex-k_ex*R)
# calculate inhibitory conductance
g_max_in = (k_in*R)/(del_e_in-k_in*R)
# kernel matrix of synaptic conductances
g_max_kernel = np.array([[g_max_in, g_max_in, g_max_in],
                         [g_max_in, g_max_ex, g_max_in],
                         [g_max_in, g_max_in, g_max_in]])
# kernel matrix of synaptic reversal potentials
e_kernel = np.array([[e_in, e_in, e_in],
                     [e_in, e_ex, e_in],
                     [e_in, e_in, e_in]])
# pattern connection (acts as high pass filter)
connection_hpf = NonSpikingPatternConnection(g_max_kernel,e_kernel)
# connect the retina to the lamina
net.add_connection(connection_hpf,'Retina','Lamina',name='HPF')
# add a vector output from the lamina
net.add_output('Lamina',name='Lamina Output')

net.render_graph(view=True) # view the network diagram
```

B.



C.

Figure 3.5: Using SNS-Toolbox to design a two-layer visual processing system. **A.** Python code to generate the desired network. Image preprocessing and output plotting are omitted. **B.** Network visual representation. An input image is converted to stimulus current for a population of neurons, representing the insect retina. From the retina, a 33 kernel of inhibitory (light blue) and excitatory (purple) synapses is applied to create a high-pass filtering effect in the next layer, representing the L1 insect lamina neurons. **C.** Output of retina and lamina neurons, respectively. Voltages are mapped to grayscale intensities.

network as 5 ms.

We assume that input images are grayscale, because the retina neurons used within the motion vision pathway only respond to a single light color (green)[25]. Since a single *Drosophila* eye consists of around 800 ommatidia arranged in 32-34 columns[75], we will also design for input images which are 32x32 pixels. As an input transformation to the retina layer, pixel values are linearly scaled from their original 0-255 to 0-R nA.

After creating two 32x32 populations of neurons and attaching an input source, the last step is to define the connection between the retina and the lamina. Our lamina model only consists of L1 neurons, so each neuron has center-on surround-off receptive field. Since each L1 cell receives input from the directly adjacent

ommatidia, we can implement these receptive fields as a 3x3 connection kernel:

K

where and are the desired gains for inhibitory and excitatory synapses, respectively. For desired behavior, these gains were chosen as $\frac{1}{9}$ and 1. These synaptic gains can be transformed into synaptic conductances and relative reversal potentials using the method described in[121],

——— (3.34)

where all excitatory synapses use an of 160 mV and all inhibitory synapses use an of -80 mV.

The results of simulating the network are shown in Figure 3.5. The output of the lamina layer is as expected, and correctly implements a rudimentary high-pass filter or edge-detector.

3.4.4 Mobile Robot Control

As a toy application example, we use SNS-Toolbox to control a simulated mobile robot. A skid-steer Jackal robot (trademark Clearpath Robotics, Kitchener, ON, USA) is placed in a navigational course resembling a figure-eight in the Gazebo physics simulator [73] (Figure 3.6B), with the goal being to drive the robot around the course without colliding with any of the walls or barriers. The simulated robot is equipped with a planar LiDAR unit, and is controlled and operated using the ROS software ecosystem [99]. We implement the neural control system as a ROS node which subscribes to the angular distance readings from the laser scanner, and publishes to the velocity controller onboard the robot.

The control network, shown in Figure 3.6A, implements a Braitenberg-inspired [16] steering algorithm. The laser scan sends distance measurements for 720 points in a

270 arc around the front of the robot, and each neuron in a population of 720 non-spiking neurons receives external current from a corresponding directional distance scan. These currents are scaled and mapped by the following relationship,

$$\frac{1 \quad 1}{1 \quad 1} \tag{3.35}$$

such that each neuron has a steady-state voltage of 0 when the sensor distance is at its maximum value , and increases to 1 when the distance is at its minimum . This population then excites two heading control neurons, with the left 360 neurons exciting the clockwise rotation neuron, and the right 360 exciting the counter-clockwise rotation neuron. All synapses between the sensory and heading control neurons share the same synaptic conductance. The difference between the potentials of these two neurons is taken and scaled to generate the desired angular velocity of the robot,

(3.36)

As the robot approaches a barrier, the system generates stronger rotational commands to move away from the obstacle. All 720 sensory neurons also inhibit a speed control neuron, which scales the linear velocity of the robot as

(3.37)

The speed control neuron also has a constant applied bias current of 1 nA. This has the effect of dynamically slowing the robot as it becomes closer to obstacles, allowing the rotational commands to correctly orient the robot. This controller results in successful navigation of the driving course in 133.24 s, with minimal tuning. Currently the velocity is updated with every neural step, however for improved speed performance the velocity can be updated after multiple neural steps. This allows the neural states to converge to a steady-state for each scan distance, and reduces the amount of communication traffic.

Braitenberg-inspired [16] networks have been widely used for steering and lane-keeping tasks in the past [70, 132, 135] to great success. The network designed in

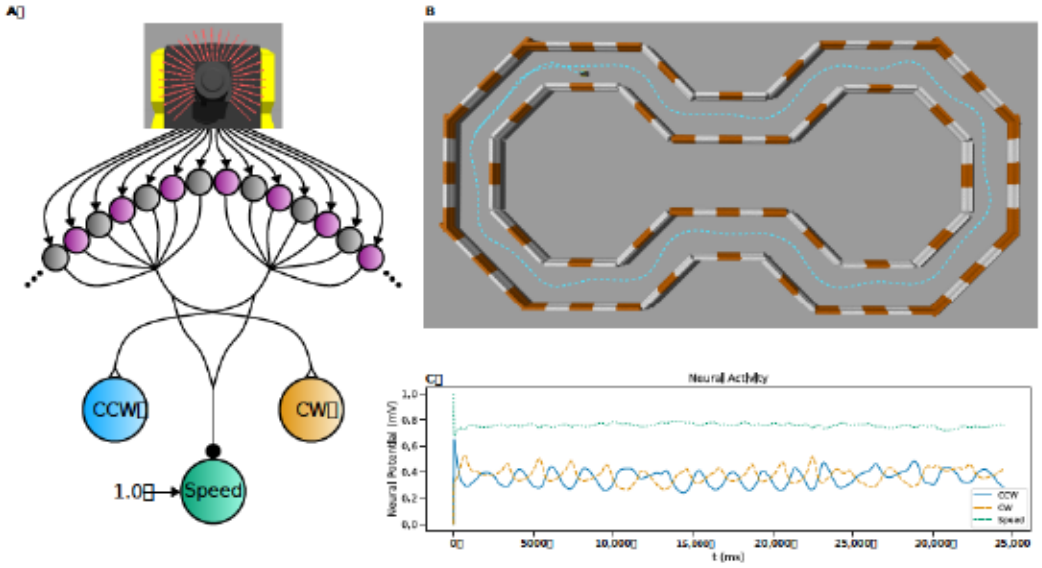


Figure 3.6: LiDAR-based steering algorithm for a simulated mobile robot using ROS. (**A**): Network diagram of the control network. Each distance measurement angle of a simulated LiDAR is inverted, scaled, and mapped as the input to a single input processing neuron. These are then summed onto directional neurons corresponding to clockwise or counter-clockwise rotation, depending on which half of the scanning field the neuron represents. All sensory neurons also connect to a speed control neuron. The difference between the directional neurons is taken as the rotational velocity, and the speed control neuron is scaled by the maximum speed to control the linear velocity. (**B**): Overhead view of the simulation environment in Gazebo [73]. Orange and white barriers act as boundaries of the course, the robot trajectory is superimposed on top with a dashed blue line. (**C**): Neural activity of the three command neurons during the generation of the trajectory shown above.

this section is intended as a proof of concept to showcase the ability to interface SNS-Toolbox with ROS simulations, not as a state-of-the-art steering algorithm.

3.4.5 Musculoskeletal Dynamics

In Deng et al. [35], an SNS was designed to control a biomechanical simulation of rat hindlimbs, with the network and body dynamics simulated using AnimatLab [28]. Here we reimplement this SNS using SNS-Toolbox and interface it with a new biomechanical model implemented in the physics simulator Mujoco [127].

Neural Model

An overall network diagram can be found in Figure 3.7A. The general network structure consists of a two-layer CPG with separate rhythm generation (RG) and pattern formation (PF) layers [106], with each layer comprising of half-center (HC) oscillators [19]. The RG network has two HC neurons which contain voltage-gated ion channels (Equation (3.14)), which mutually inhibit one another via two non-spiking interneurons (Equation (3.1)). This network generates the overall rhythmic activity of the legs, and the global speed can be controlled via the level of mutual synaptic inhibition [122].

The HC neurons of the RG network excite the corresponding HC neurons in the PF networks, which are constructed in a similar manner to the RG network. Each PF network shapes the phase from the RG network into the appropriate joint position for a specific joint, with the knee and ankle joints sharing a PF network. The PF networks are also presynaptic to a motor control network for each joint [63, 68], where motoneurons (MN) drive the flexor and extensor muscles for each joint and are adjusted by Ia and Ib feedback from the muscles.

Biomechanical Model

In Deng et al. [35], the rat hindlimbs were modeled in AnimatLab [28] using simplified box geometry and a pair of linear Hill muscles for each joint. We replicate this model in Mujoco [127] using a three-dimensional model of bone geometry [69] and non-linear Hill muscles (shown in Figure 3.7C). Mujoco was chosen due to its open-source availability, and its robust internal support for complex muscle-based actuation with a lower computational overhead than OpenSim [34].

All muscles in the model share the same sigmoidal activation curve which converts motoneuron activity to a muscle activation between 0 and 1. This is

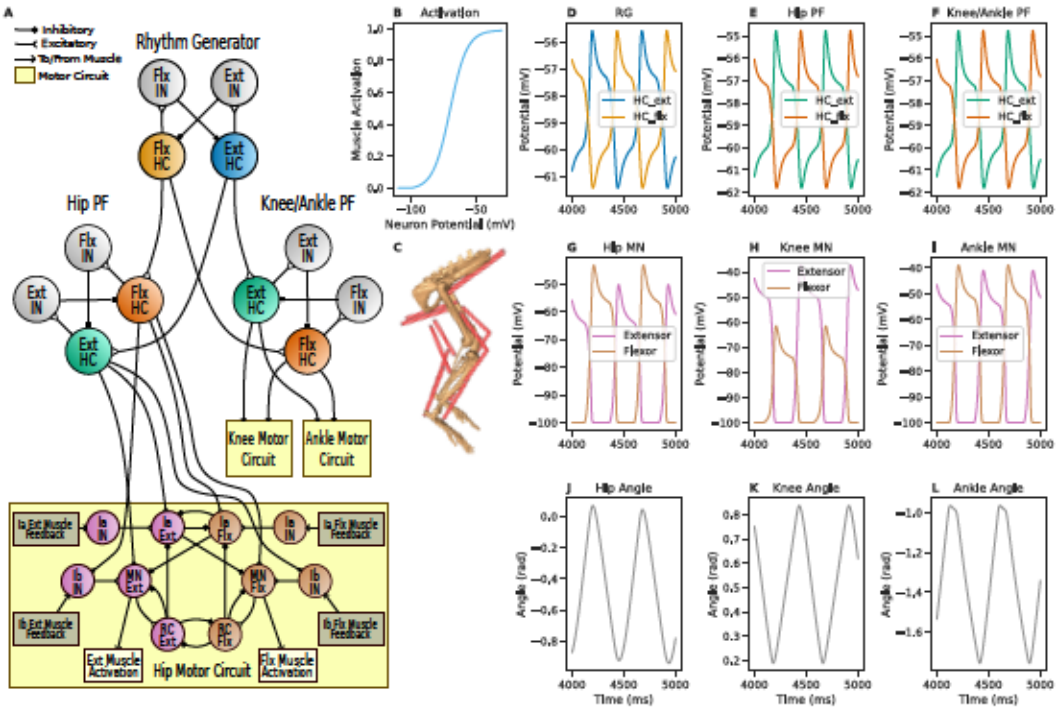


Figure 3.7: SNS-Toolbox controls a musculoskeletal model of a rat hindlimb. **(A)**: Diagram of the neural control network. **(B)**: Relationship between motor neuron voltage and muscle activation. **(C)**: The musculoskeletal model used in Mujoco [127]. **(D)**: Neural activity from the half-center neurons in the central rhythm generator. **(E,F)**: Neural activity from the hip and knee/ankle pattern formation circuits. **(G–I)**: Motor neuron activity in the motor circuits for the hip, knee, and ankle. **(J–L)**: Joint angles of the hip, knee, and ankle. All recordings are shown for a period of 1000 ms, after the model has finished initialization. Pictured are recordings from the elements within the left leg.

calculated in the same manner as [139], with the activation sigmoid defined as

$$\frac{1}{1} \tag{3.38}$$

where is the steepness of the sigmoid and is the motoneuron potential. The exact curve is shown in Figure 3.7B.

Simulation Results

The network and mechanical model are simulated for 5000 ms, with data shown in Figure 3.7. On each step, muscle tensions from Mujoco are first formatted as Ia and

Ib feedback for the SNS, then the outputs of the SNS are mapped via Equation (3.38) into muscle activations for the Mujoco model. The network parameters are exactly the same in this work as in Deng et al. [35], and result in oscillatory motor behavior. The overall leg oscillation occurs at half the frequency of the original model, and the joints exhibit scaled and shifted trajectories. The hip joint oscillates in the range of 0 94 0 07 radians instead of 0 09 0 17 radians in the original model, with the knee in 0 19 0 84 radians instead of 0 47 0 09 radians and the ankle in 1 76 0 97 radians instead of 0 92 0 26 radians. Further investigation is needed to determine the sources of these discrepancies in behavior, however a difference is to be expected given the difference in muscle modeling.

3.5 Discussion

In this work, we present SNS-Toolbox, an open-source Python package for simulating synthetic nervous systems. We focus on simulating a specific subset of neural and synaptic models, which allows for improved performance over many existing neural simulators. To the best of our knowledge, the SNS-Toolbox is the only neural simulator available which meets all of the desired functionality for designing synthetic nervous systems. The SNS-Toolbox is not tied to a dedicated graphical user interface, allowing networks to be designed and simulated on embedded systems. Heterogeneous networks of both spiking and non-spiking neurons, as well as chemical and electrical synapses, can also be simulated in real-time on both CPU and GPU hardware. All of these capabilities are also fully available across all major operating systems, including Windows (trademark Microsoft Corporation), MacOS (trademark Apple Corporation), and Linux-based systems.

We find that SNS-Toolbox can simulate networks with hundreds to thousands of neurons in real-time on desktop hardware, and low hundreds of neurons on embedded hardware. The performance is also competitive with other popular neural simulators. Through a simple programming interface, it is relatively simple to combine networks

made in SNS-Toolbox with other software. Using ROS [99], we implemented a Braitenberg-inspired [16] neural steering algorithm and controlled navigation of a simulated mobile robot through an environment in Gazebo [73]. We also take an existing SNS network which controlled a musculoskeletal model [35] implemented in AnimatLab [28], and achieved cyclical limb motion after reimplementing the network in SNS-Toolbox and interfacing with Mujoco [127].

One decision we made early in the design process was to provide a simplified design and compilation interface, as well as to build SNS-Toolbox on top of widely used Python numerical processing libraries in order to facilitate use across all computing platforms. This has allowed multiple researchers with varying degrees of programming experience within our laboratories to begin using SNS-Toolbox successfully, as well as an instructional tool in pilot classes on neurorobotics. While other tools, such as ANNarchy [130], achieve higher performance by direct code generation in C++, they do so at the expense of easy cross-platform support. Future work may explore adding additional build systems for different operating systems in order to achieve comparable performance.

In order to allow network simulation on GPUs, multiple backends in SNS-Toolbox are built on top of PyTorch [95]. However, PyTorch has a large infrastructure of features which are currently not supported by the structure of the SNS-Toolbox backend, such as layer-based organization of networks and gradient-based optimization using automatic differentiation. Additionally, models built using the formal PyTorch style are able to be compiled into the C++-adjacent Torchscript, allowing improved simulation performance. Work is currently underway to restructure the PyTorch backend within SNS-Toolbox to allow these benefits.

Previous SNS models have often been made using the software AnimatLab [28, 50, 64, 120], which uses a different workflow than SNS-Toolbox. Within AnimatLab, users have an integrated GUI that contains a rigid-body modeler, canvas for dragging and dropping neurons and synapses into a network, and a plotting window for

viewing simulation results. The SNS-Toolbox is designed to focus on the design and simulation of the neural and synaptic dynamics, with the physics simulation and plotting being relegated to external libraries. While this may be less convenient for a user who is either a beginner or is migrating from AnimatLab, we feel that this separation is beneficial as it allows networks made using the SNS-Toolbox to be more extensible to interfacing with other systems. When transitioning from AnimatLab, the primary difference in workflow is that networks in the SNS-Toolbox are described via code instead of drawn. If the transitioning user is familiar with writing code in Python or another similar language, this change is easily managed. The other difficulty when converting from AnimatLab to SNS-Toolbox is when using MuJoCo [127] for physics simulation. As the native muscle model within MuJoCo is different than AnimatLab, we show in Section 3.4.5 that the same network with the same parameter values will exhibit different behavior if not tuned to use the new muscle model.

Throughout the design process of SNS-Toolbox, we chose to focus on implementing specific sets of neural and synaptic dynamics. This brings enhanced performance, however it does mean that there is no method for a user to add a new neural or synaptic model to a network which has not been previously defined, without editing the source code for the toolbox itself. One workaround to this issue is to create more complicated models by treating individual non-spiking or spiking neurons as compartments which are connected together as a multi-compartment model, however, in general, we find that this is a limitation for the SNS-Toolbox at this time.

The SNS-Toolbox is currently available as open-source software on GitHub (trademark GitHub incorporated), and has an extensive suite of documentation freely available online. In addition to installing from source, the SNS-Toolbox is also available to install from the Python Package Index (PyPi). All of the features within the toolbox are built on top of standard, widely used Python libraries. As long

as these libraries maintain backwards compatibility as they update, or the current versions remain available, the functionality of SNS-Toolbox should remain into the future.

Examples were shown using the SNS-Toolbox to interface with external software systems, particularly ROS [99] and Mujoco [127]. While the primary goal of SNS-Toolbox is a simplified interface which focuses on neural dynamics, other users may find our interface mappings between SNS-Toolbox and other software useful. As such, we intend to release supplementary Python packages which contain helper logic to interface the SNS-Toolbox with other software as we develop them.

Many robots have been built which use SNS networks for control, although these are usually tethered to an off-board computer [50, 64] or require non-traditional computer hardware [5] to operate. With the release of SNS-Toolbox, we have two forward-looking hopes. Firstly, that more researchers design and implement synthetic nervous systems for robotic control, and that members of the robotics community will find value in neural simulators which are capable of simulating heterogeneous networks of dynamic neurons.

Chapter 4

A SYNTHETIC NERVOUS SYSTEM FOR ON AND OFF MOTION DETECTION INSPIRED BY THE *DROSOPHILA MELANOGASTER* OPTIC LOBE

Material in this chapter has been previously published in

- Nourse, W. R., Szczecinski, N. S., & Quinn, R. D. (2023, July). A Synthetic Nervous System for on and Off Motion Detection Inspired by the *Drosophila melanogaster* Optic Lobe. In Conference on Biomimetic and Biohybrid Systems (pp. 364-380). Cham: Springer Nature Switzerland.

Edits have been made to place this material into context with the rest of this dissertation.

4.1 Abstract

In this work, we design and simulate a synthetic nervous system which is capable of computing optic flow throughout a visual field, inspired by recent advances in the neural anatomy of *Drosophila melanogaster* found through connectomics. We present methods for tuning the network for desired stimuli, and benchmark its temporal properties and capability for directional selectivity. This network acts as a stepping point towards visual locomotion control in a hexapod robot inspired by the anatomy of *Drosophila*.

4.2 Introduction

A continuing goal in the robotics community is to develop robots with the dynamic capabilities and resilience of animals. A particular focus is on adding the influence of visual information to improve the adaptability of robotic systems [7, 140]. A

promising approach is to design robotic controllers using neuromorphic networks of neurons with biologically inspired dynamics [6], also known as synthetic nervous systems (SNS) [50, 64, 121].

Much is known about the circuitry within the *Drosophila melanogaster* optic lobe, making it a convenient inspiration for robotic vision systems. For visual motion processing in particular, the *Drosophila* nervous system contains many of the same logical elements as that of mammals and vertebrates [26], but does so with three orders of magnitude fewer neurons in the visual system [13, 80]. An additional advantage of *Drosophila* over other model organisms is that extensive work has been done in recent years to create a full connectome of their brains [107, 137], and the visual system in particular has been extensively studied [112, 115, 125].

The motion vision pathway is extremely important for adaptive behavior in *Drosophila*, aiding in estimation of body motion and enabling rapid response to oncoming threats [1, 31, 48]. The structure is well documented, see Fig.4.1 for a visual representation and refer to [13] for a more thorough review. Within the *Drosophila* optic lobe, retinal and lamina cells convert changes in light intensity into information used in the rest of the network. Of particular relevance to the motion vision system are cells L1-L3, which perform spatiotemporal filtering of input stimuli and separate information flow into two pathways: an On pathway for encoding increases in brightness, and an Off pathway for encoding decreases in brightness [112, 125]. From there, the transformed visual information is further filtered in the medulla into a bank of unique filters (Mi1, Tm3, Mi4, Mi9 for the On pathway; Tm1, Tm2, Tm4, Tm9 for the Off pathway), each with slightly different spatiotemporal characteristics [4, 40]. These are then combined nonlinearly (along with the wide amacrine cell CT1 [86]) onto the elementary motion detector (EMD) cells T4 and T5, and the resulting combination generates directional selectivity for each point in the visual field [115] which can then be spatially integrated for more complex behavior [13].

Previous work has adapted this circuitry to robotics [7] and SNS networks [110], but these studies were performed before the wide breadth and depth of connectivity information from connectomic analysis for *Drosophila* became available. In this work, we design an SNS network which measures optic flow for both rising and falling brightness levels, using inspiration from the current body of knowledge about connectivity and activity within the *Drosophila* optic lobe [13, 115]. As there is less known about the exact operation of the Off EMD, we make some design-based decisions in its construction. Using the capabilities of this network, we plan future visual control of motion onboard the bio-inspired robot *Drosophibot* [50].

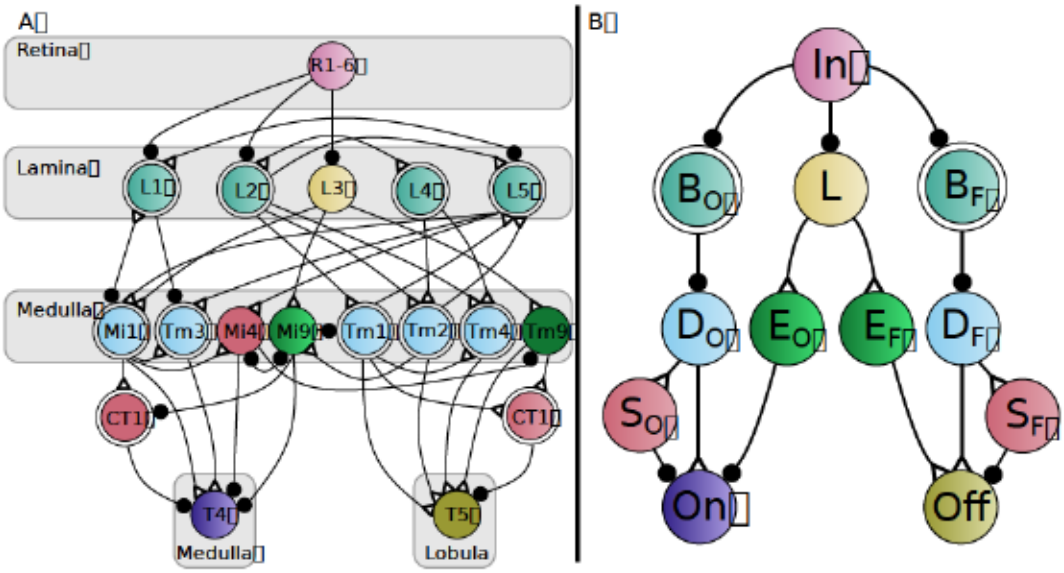


Figure 4.1: **A:** A circuit diagram of a single column within the *Drosophila* motion vision pathway, adapted from [13, 115]. **B:** Reduced diagram used in this work. Node colors in both diagrams are chosen to highlight their common functional roles. Single circles designate neurons which behave as a low-pass filter, double circles indicate a band-pass filter. Dark closed circles indicate inhibitory synapses, open triangles indicate excitatory synapses. In panel **B**, *D* and *S* neurons approximate band-pass behavior by filtering the responses of the *B* neurons, for reduced computational complexity.

4.3 Network Components

4.3.1 Neural and Synaptic Models

As this work is primarily focused on designing general network behavior instead of exactly reproducing neural recordings, all neurons in the network are simulated as non-spiking leaky integrators following [121], where the neural state is updated as

(4.1)

where is the neural time constant, is any external input, and is a constant bias term. is the synaptic input from any presynaptic neurons in the network,

(4.2)

with denoting a presynaptic neuron, and denoting the synaptic reversal potential. In this work, all excitatory synapses have 5 , 2 for inhibitory synapses, and specific modulatory synapses have a reversal potential of 0 1 , where is the primary range of neural activity in the network. For numerical simplicity, 1 in this work so that most neurons communicate when their state is between 0 and 1, with the exception of the synapse between neurons and . has an effect roughly analogous to cholinergic synapses in *Drosophila*, to GABAergic synapses, and to glutamatergic synapses. is a monotonic function which describes the incoming synaptic conductance such that 0 where is the upper bound. In this work, we define as

0 ——— 1 (4.3)

where and are the lower and upper threshold states of synaptic activity. For most synapses in the network, we set 0, and . For the synapse between neurons and , we set , and 2 .

Steady State Formulation

For some design sections, we required solving for the steady state of the neuron given steady inputs. As in [121], the steady-state response is

$$\frac{-}{1 \quad -} \tag{4.4}$$

Synaptic Pathway Designs

When connecting components of our network, different pathways are tuned analytically based on their functional role. One common example is signal transmission, where the desired steady-state value (see eq. 4.4) of the postsynaptic neuron is the steady-state voltage of the presynaptic neuron multiplied by a transmission gain . From [121], this can be solved as

$$\frac{}{} \tag{4.5}$$

where is for excitatory synapses, and for inhibitory synapses.

Another formulation which is used throughout this work comes from setting a target state of the postsynaptic neuron, given a presynaptic steady-state and the presence of other external or synaptic currents to the postsynaptic neuron. This is derived from eq. 4.4 and written as

$$\frac{}{} \tag{4.6}$$

Finally, in some instances it is desirable for a presynaptic neuron to modulate the sensitivity of a postsynaptic neuron to external and synaptic inputs. For this we follow the derivation in [121], and use the modulatory reversal potential and set the synaptic conductance as

$$1 \tag{4.7}$$

where the desired behavior is such that is divided by when . This form of synapse is used within the On pathway between and *On*.

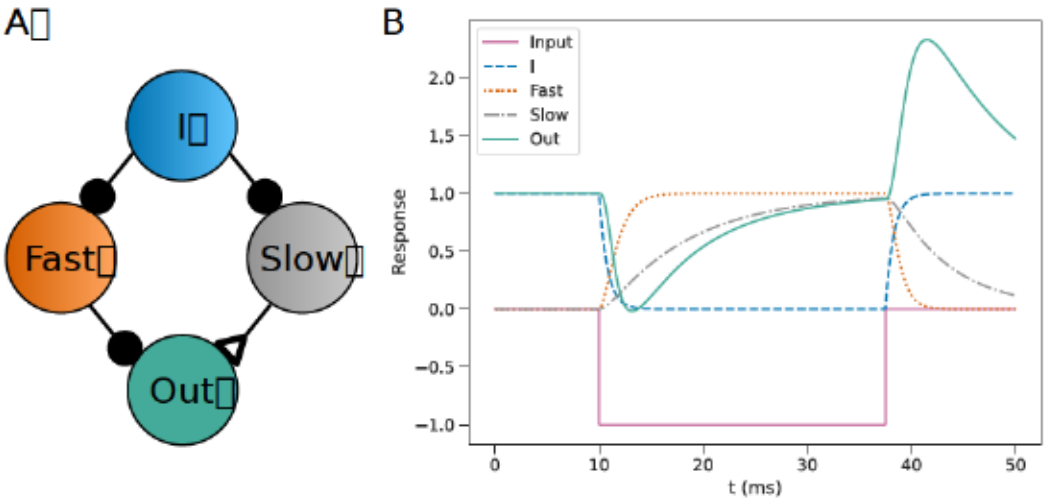


Figure 4.2: **A:** Circuit diagram of a band-pass subnetwork. Two neurons are tuned as low-pass filters with different cutoff frequencies, and are subtracted to produce a band-pass response. **B:** Response of each neuron within the subnetwork when subjected to a time-varying input.

4.3.2 Neural Filters

Most neurons within the *Drosophila* motion vision pathway behave temporally as either low or band-pass filters [13], and here we describe our methodology in designing our network to behave accordingly. The process to tune our neurons as low-pass filters is straightforward, as the leaky integrator is itself a low-pass filter with a cutoff frequency (where the gain is -3dB) defined as $f_c = \frac{1}{2\pi\tau}$.

Common methods for implementing a neuron with band-pass temporal behavior typically involve adding a second dynamic variable to the neuron model [66], such as a voltage-gated ion channel [122] or adaptive spiking threshold [123, 126]. Inspired by the differentiation network in [121], we implement band-pass filters in our network using a subnetwork of four non-spiking leaky integrators instead of adding a new, more complex neural model. While this adds more neurons to the network, we do this to reduce the computational complexity of our system in anticipation of running it on embedded hardware. For a visual representation, please

see Fig. 4.2.

This network is designed as follows. Inputs to the subnetwork enter as a synaptic input for neuron *I*, which then inhibits neurons *Fast* and *Slow* using inhibitory transmission synapses (eq. 4.5) with reversal potential and a gain 1, to arrive at a maximum synaptic conductance of

— (4.8)

Neurons *I*, *Slow*, and output neuron *Out* all have the same time constant, which approximately acts as the upper bound of the filter's passband. Neuron *Fast* has a larger time constant and lower cutoff frequency than the other neurons, and the corresponding cutoff frequency results in setting the lower bound of the passband. Neuron *Fast* inhibits neuron *Out* with a gain-controlled inhibitory synapse (eq. 4.5), and neuron *Slow* excites *Out* with a synapse designed to mirror , where

——————— (4.9)

This results in the state of neuron *Out* being the difference between the original signal in *I* being processed via two different low-pass filters, resulting in a band-pass effect. In practice, the value of the transmission gain from *Fast* to *Out* is found using the Brent method for scalar minimization [18] in SciPy [129] such that the change in magnitude during a step input is -1. All neurons in this subnetwork additionally have a constant bias input of , since the bandpass filters in our network need to hyperpolarize during rising luminance levels, but omitting these bias terms and swapping the inhibitory and excitatory synapses would result in a more traditional band-pass filter [121].

4.4 Network Design

For a circuit diagram of the network described in this section, as well as the comparative structure present in the *Drosophila* optic lobe, please refer to Fig. 4.1.

4.4.1 Input Processing

When presented with a visual stimulus, each input node (denoted *In* in Fig. 4.1B) acts as a temporal low-pass filter, performing an analogous operation to a *Drosophila* photoreceptor cell [26]. As this initial stage sets an upper bound on the frequency response for the rest of the circuit, we set the time constant for this filter such that no frequencies in our desired input range are filtered out and our network dynamics remain stable. In this work, we set this as 10 based on our simulation timestep .

4.4.2 Initial Filter Stage

Similar to the cells present in the *Drosophila* lamina, we apply temporal filters to the output of the initial input filtering stage. While the primary lamina cells in the *Drosophila* motion pathway have slight differences in temporal behavior [40], for analytic simplicity both and have the same properties in this work. For reduced analytic complexity, all spatial receptive fields are condensed into single columns.

We apply a band-pass filter which hyperpolarizes to stimuli of increasing brightness within each pathway to the output of the input stage, analogous to the behavior of the L1 and L2 cells [13], and we refer to them as and . These are constructed in the manner described in Section 4.3.2, and the time constant of the fast side is set to . For the slow side, we choose so that for the fastest input stimulus, the response has time to settle to baseline over the course of a single input period. Approximating the settling period for a leaky integrator as 5 , we write the time constraint as

$$5 \qquad \frac{1}{2} \qquad (4.10)$$

where is the spatial wavelength and is the fastest spatial velocity of the input stimulus.

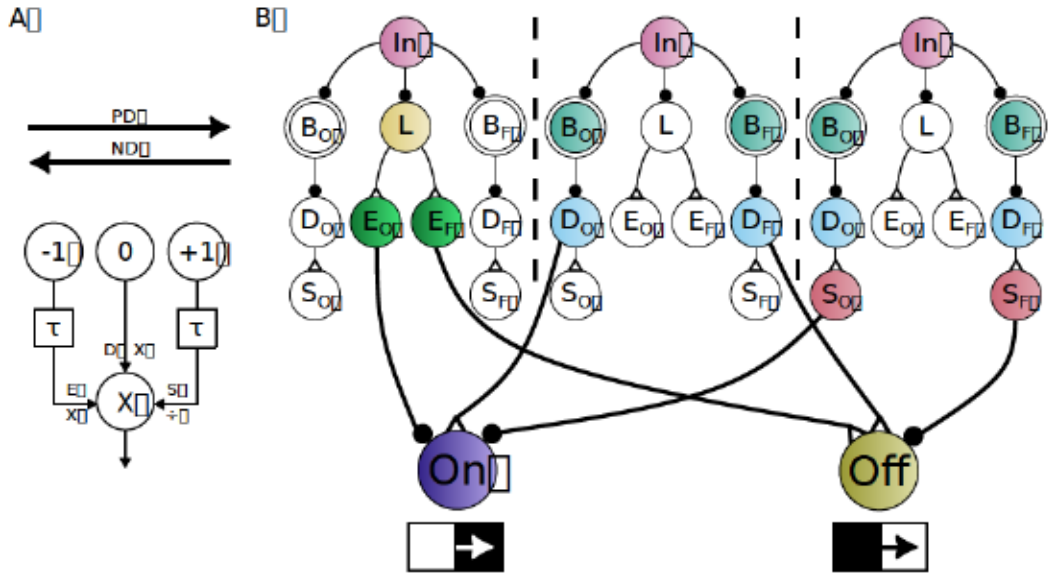


Figure 4.3: **A:** Diagram of a three-arm Haag-Borst HR/BL EMD circuit [54]. **B:** Schematic of the three-arm motion detectors in this work, for both On and Off stimuli. PD denotes the preferred direction, ND denotes the not preferred (null) direction. Nodes without color do not contribute to behavior in this direction of motion.

Similar to the L3 neuron in *Drosophila* [13], we include an additional low-pass filter (denoted as L in Fig. 4.1B) which is shared across both the On and Off pathways. In order to preserve the range of temporal information available for later processing, we set the time constant of L to τ_{fast}, causing this node to act as a delayed and inverted copy of the input stimulus.

4.4.3 Motion Detectors

For the design of the elementary motion detectors (EMD) in the On and Off pathways, we take inspiration from the Haag-Borst HR/BL three-arm EMD [54](Fig. 4.3A). This structure has been shown to be capable of reproducing recordings from T4 [54] and T5 [55] cells, and connectomic analysis has found candidate cells for each input arm of the model [115]. In this model, the output neuron of each EMD (*On* for the On pathway, *Off* for the off pathway) receives input from three separate elements: a

direct input from the cell in the same column, an enhancement input which enhances stimuli coming from the preferred direction, and a suppressor input which suppresses stimuli coming from the null direction. In this work we choose to model the EMD as a three-arm circuit instead of older models which used two arms to achieve either preferred-direction enhancement [57] or null-direction suppression [8], as the three-arm model generates finer directional selectivity and is less susceptible to noise [54].

In *Drosophila* the inputs in each arm can come from multiple neurons, which combine to create varied spatio-temporal properties [4, 13, 40]. For simplified analysis and reduced computational complexity, each arm is represented as a single neuron in this work. Additionally, while the cells in the medulla act as band-pass and low-pass filters with a variety of time constants, for simplicity we represent all of them as low-pass filters and reshape the activity from the higher-level filters (, , and). In this work, neurons which perform enhancement are named , direct stimulation , and suppression . Unless otherwise specified, all neurons can be assumed to have a time constant of . This is only changed as needed for behavioral reasons, which will be explained below.

On Pathway

Recent studies have focused on the behavior of T4 cells and found potential mechanisms which generate motion-detection and direction selectivity using the cells which contribute to the On pathway [115], particularly in the work of Groschner et al. [52]. In their work, they modeled the T4 pathway and found that multiplicative behavior can occur during a period of low inhibition that creates a "window of opportunity" [37]. We adapt a similar mechanism here based on our previous work [121], with a circuit diagram in Fig. 4.3B and behavior shown in Fig. 4.4. Neuron mimics the behavior of Mi1 and Tm3, the response of CT1, and is analogous to Mi9.

Direct: Neuron receives input from via an inhibitory transmission synapse (eq. 4.5), and the gain is tuned via Brent's method [18] such that the peak during a step input is 1. It excites *On* with an excitatory transmission synapse, with the gain tuned again via Brent's method for an isolated peak response of 1.

Suppression: Unlike the other arms of the On EMD, receives input from instead of the filtering stage. This is similar to CT1 receiving indirect input from Mi1 [86, 115]. This creates a slight delay in the response, which improves the ability of to suppress stimuli in the null direction. The connection between and *On* is tuned as an inhibitory target synapse (eq. 4.6) which aims to bring the state of *On* to zero when is signaling with peak strength.

Enhancement: In our model, is responsible for the majority of the stimulus-dependent behavior of the On EMD. Starting with the temporal response, we set the neural time constant so that the state of settles during the time it takes the signal to travel from one column to the next at the slowest desired velocity , assuming that the neuron settles after a time period of 5 (eq. 4.10). This is found with

$$\frac{\qquad}{5} \tag{4.11}$$

where is the spatial resolution of the model (5 in this work). stimulates *On* using a modulatory synapse with a division factor of 10 (eq. 6.7), and is stimulated by using an excitatory transmission synapse (eq. 4.5) with unity gain.

Off Pathway

While studies have found the presynaptic neurons which generate direction selectivity within the Off motion detector circuit [86, 115], current studies which model the Off pathway either omit the role of CT1 in suppression [74] or do not model chemical reversal potentials [76]. As such, we make some base assumptions based

on the connectivity in order to produce direction selectivity. Neuron implements the role of Tm1, Tm2, and Tm4, is analogous to CT1, and mimics Tm9.

Direct: receives stimulation from via an excitatory transmission synapse (eq. 4.5) with a gain tuned such that the postsynaptic peak is 1 for a decreasing step response in brightness, and and of and 2 respectively. It stimulates *Off* with an excitatory target synapse of target (eq. 4.6), with the conductance multiplied by , where is the percentage of direct stimulation. In this work, 0 5. We found that the peak of is a primary factor in the magnitude of *Off*, so we select the time constant so that the peak magnitude starts decreasing at our slowest input velocity . We first find the frequency of our slowest input as ——, and scale that to get 10 . A scaling factor of 10 is chosen because the gain of leaky integrators begins to decrease approximately 1 decade below the cutoff frequency on a logarithmic scale.

Suppression: is tuned in a similar manner to , receiving an excitatory transmission (eq. 4.5) input from that is optimized using Brent's method [18] for a peak magnitude of 1. inhibits *Off* via an inhibitory target synapse with the same properties as the synapse between and *On*.

Enhancement: is tuned with the exact input and neural properties of for simplicity. It stimulates *Off* with an excitatory target synapse (eq. 4.6) with target , and the conductance scaled by where is the percentage of enhancement stimulation and is constrained so 1.

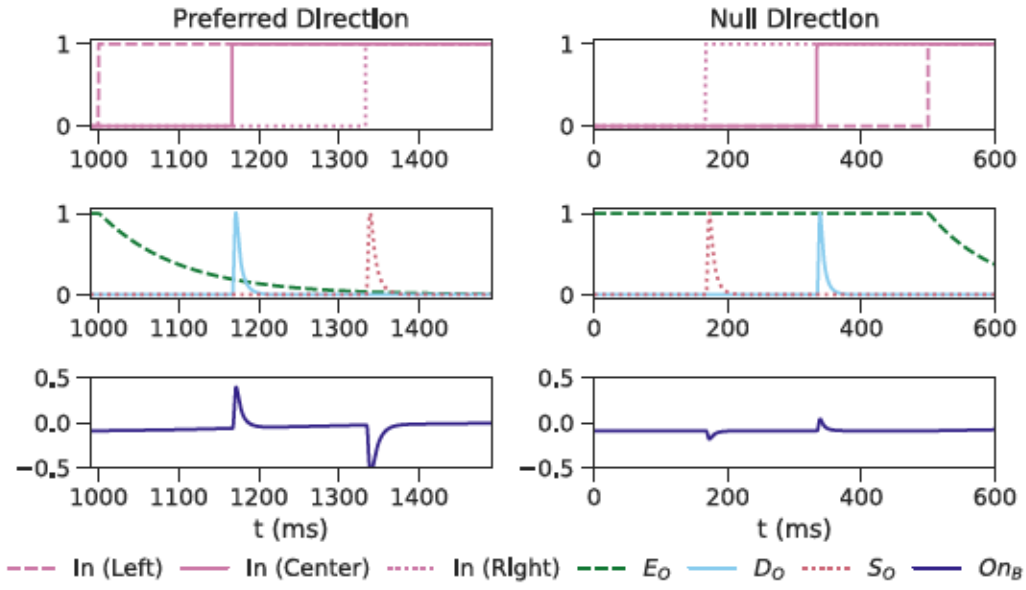


Figure 4.4: Simulation of elements within the On pathway during a stimulus moving in the preferred (*Left*) or null (*Right*) directions. Dashed green traces correspond to Enhancement ( ) signals, solid blue to Direct ( ) signals, dotted pink to Suppression ( ) signals, and solid indigo for the On EMD (*On*). *Top:* Traces of visual stimuli to the Enhancement, Direct, and Suppression columns of the motion detector; *Middle:* Traces of the Enhancement, Direct, and Suppression neurons which are presynaptic to the EMD neuron; *Bottom:* Trace of the final motion detector, which depolarizes for stimuli traveling from left to right (*On*).

4.5 Results

4.5.1 Simulation Setup

All simulations are done using SNS-Toolbox [94], a Python package for designing and simulating synthetic nervous systems (https://github.com/wnourse05/SNS-Toolbox). A of 0.1 ms is used as the simulation step. In all simulations, the network was tuned for sensitivity to images with a spatial wavelength of 30 and a velocity between 10 and 180 across the visual field. Code to simulate the network and generate all of the figures presented here is available at https://github.com/wnourse05/Motion-Vision-SNS.

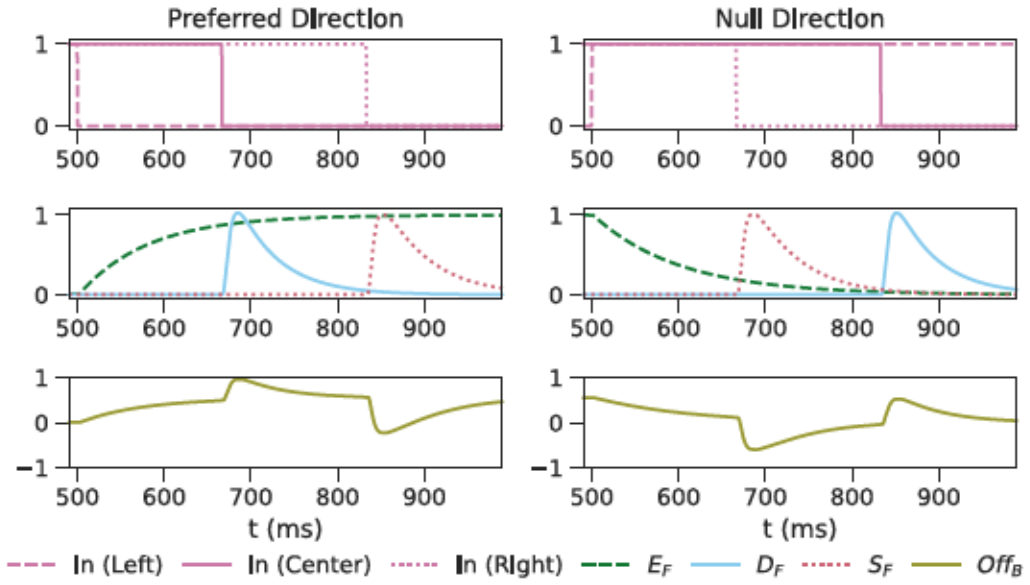


Figure 4.5: Simulation of elements within the Off pathway during a stimulus moving in the preferred (*Left*) or null (*Right*) directions. Dashed green traces correspond to Enhancement ( ) signals, solid blue to Direct ( ) signals, dotted pink to Suppression ( ) signals, and solid olive for the Off EMD (*Off*). For further description refer to Fig. 4.4.

4.5.2 Individual EMD Stimulation

To verify the basic behavior of the EMD circuits, we applied square wave gratings with a spatial wavelength of 30 and a velocity of 30 to 3 adjacent columns. We focus on the behavior of the B channel neurons, which are tuned for sensitivity in motion traveling from left to right.

Shown in Fig. 4.4, we examine the behavior of the On pathway. When stimuli of increasing brightness move across in the preferred direction, receives the stimulus change first and starts to hyperpolarize. In the time it takes for the stimulus to continue to the central column, has decreased. This allows the direct stimulus from to excite *On* with reduced inhibition. As the stimulus continues to the right column, exhibits a bout of further inhibition. The timing relationship between and creates the speed-dependent behavior; as stimuli move more rapidly, the

offset in time between these columns decreases and more inhibition is applied to *On*, causing a decrease in the activity caused by . When stimuli move in the opposite direction, and are both activated while is strongly depolarized, resulting in a significant reduction of peak magnitude in *On*.

Repeating the experiment for stimuli with decreasing brightness, as the off-edge stimulus moves in the preferred direction in the Off pathway begins to depolarize. This is accentuated by a later pulse from , followed by strong inhibition from . As stimuli move in the opposite direction, is either at rest or hyperpolarizing towards rest, depending on the timing of prior stimuli. The off-edge first arrives at which strongly inhibits *Off*, followed by a pulse in excitation from and then a separate increase in excitation from . While the specifics of the mechanism are different between the On and Off pathways, the net behavioral result is the same: stimuli traveling in the preferred direction are enhanced to some degree, while stimuli in the opposite direction do not have as strong a response.

4.5.3 Velocity Response

Square-wave stimuli with 30 are applied to a network which consisted of 49 columns, arranged in a 7x7 grid. The velocity of stimuli is varied from 10 to 360 , and data is recorded from the central EMD. As shown in Fig. 4.6, the peak magnitude of both the On and Off pathways decreases as the velocity approaches the maximum tuning range (180). The On pathway has a high dynamic range, varying smoothly from 1 to near zero, and with peaks in the preferred direction always greater than the null direction. Changes in the magnitude of the Off pathway are more gradual as it approaches the desired maximum velocity, with a slow increase slightly before this point. The ratio between the preferred and null directions is always greater than 1, but to a lesser degree in the Off than the On pathway. Behavior in the *Drosophila* T4 and T5 cells are more similar to the On results shown in Fig. 4.6 than the Off results, with a peak response that decreases as the

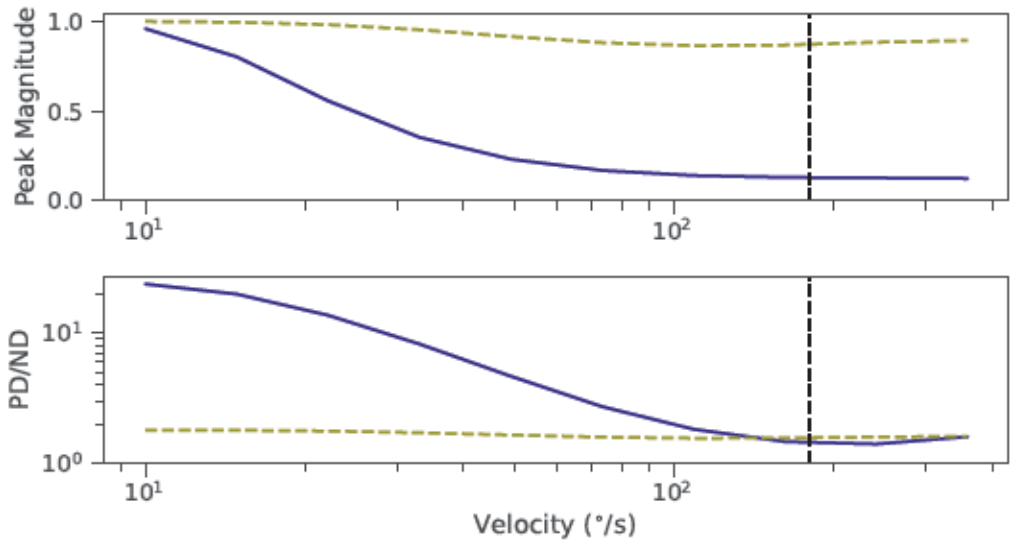


Figure 4.6: Output behavior of the On (solid indigo) and Off (dashed olive) motion detectors when subjected to a square wave, translating from 10 to 360 per second. Target maximum velocity (180) shown with a vertical dashed line. *Top:* Peak magnitude of the motion detector in the preferred direction; *Bottom:* Ratio between the motion detector in the preferred direction and the null direction.

input velocity increases [82]. However, in *Drosophila* this decrease occurs as input velocity is both increased and decreased from a peak velocity.

4.5.4 Directional Selectivity

Stimuli of a consistent wavelength and velocity are applied to the same network described in Section 4.5.3 while the direction of travel is varied from 0 360 in 45 increments, with results shown in Fig. 4.7. The EMD for each cardinal direction exhibits enhanced sensitivity to stimuli in the preferred direction, and reduced sensitivity to the other directions. The On pathway is able to generate a finer level of directional sensitivity than the Off pathway, due to its multiplicative window of reduced inhibition. Further work is necessary to find a similar multiplication mechanism for the Off pathway.

As the networks for each cardinal direction are mirrored versions of each other,

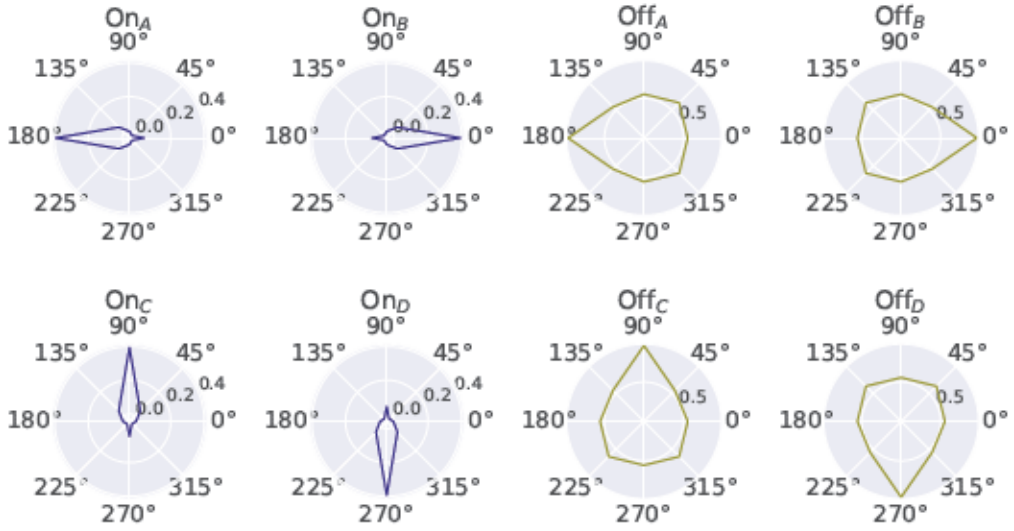


Figure 4.7: Peak response of each motion detector in the On (*Left*) and Off (*Right*) pathways to a square wave grating with 30 and 30 . Preferred direction of each sub-type: *A*: right to left; *B*: left to right *C*: bottom to top; *D*: top to bottom.

the resulting responses are identical except for their orientation. This is different than the tuning found in *Drosophila*, where the sensitivity of each cardinal direction is slightly different [82]. The general shape of our *On* and *Off* responses most closely matches the behavior of the T4b and T5b neurons in the animal, consisting of a sharp triangular point in the preferred direction and a slight bump in the null direction, however T5b is much more similar to T4b than our *Off* neurons are to the *On* neurons.

4.6 Discussion and Future Work

In this work, we implement an SNS network which is a reduced model of the *Drosophila* motion vision system. The network performs optic flow measurement at each point in the visual field, and can be tuned for different ranges of input stimuli in a parametric manner. While some parameters are found via numerical optimization, most are chosen by hand via analytic rules. With further optimization,

we expect that the performance of the network could be tuned to detect particular stimuli.

Compared to the circuit found in *Drosophila*, the model presented here is far reduced in complexity. In particular, the animal uses more neurons as inputs to the EMD cells, which allows for better temporal response and additional adaptation to factors such as changing input contrast [40]. Adding more neurons into the motion detection area in our network may be promising for future development. Another simplification in our model is that the initial filter stage only receives visual input within its own column. This is not the case for the lamina neurons in *Drosophila*, which perform spatial filtering over a 15 20 radius for each column [13]. Future work will extend our analysis to generate directional selectivity in the presence of wider spatial receptive fields.

While our implementation of the On pathway is derived from detailed biological models [52], less recordings and detail were available for the Off pathway. Our model attempts to model direction selectivity using current information about the structure of this system, but showcases some current gaps in understanding. In particular, the neuron in our model which is intended to act analogously to Tm9 ( ) does not provide a significant role in motion detection based on its connectivity. This differs strongly from biological experiments, where the effect of Tm9 in Off motion detection is greater than many of the other neurons combined [112]. As such Tm9 may have additional functionality and roles, as discussed in [115].

Much work has been done to study the effect of the visual system on walking control in *Drosophila* [31, 48]. We aim to continue development of the network described in this work, so that it may be used to assist in the control of legged motion onboard our *Drosophila*-inspired robot, *Drosophibot* [50].

Chapter 5

SNSTORCH: SIMULATION OF LARGE-SCALE SYNTHETIC NERVOUS SYSTEMS

Some material in this chapter was presented in person as the tutorial "An Introduction to Design and Simulation using SNS-Toolbox and SNSTorch" at the 2024 conference on Neuro Inspired Computational Elements (NICE) on April 26, 2024.

5.1 Abstract

SNS-Toolbox is an open-source framework for designing and simulating networks of bio-plausible neurons and synapses at moderate network scale with arbitrary connectivity. Due to representing arbitrary networks as a single recurrent population however, SNS-Toolbox experiences timing and memory issues for networks with more than thousands of neurons. In this work we introduce SNSTorch, a companion package to SNS-Toolbox which supports larger networks and layer-based design. We benchmark the speed perfomance of SNSTorch against SNS-Toolbox on a population-based task, and then demonstrate the optimization of SNS networks on both regression and classification tasks.

5.2 Introduction

In the world of neural simulation software, there is a wide variety of systems which are capable of simulating different aspects of neural computation [41, 42, 91, 141]. Some focus on smaller networks with detailed models of biophysical mechanisms [59], while others focus on larger networks with simple neural dynamics but complex population interactions [15, 58]. In Chapter 3 we presented SNS-Toolbox, an open-source neural simulator in Python which can simulate synthetic nervous system (SNS) networks of hundreds to thousands of neurons in real-time

or faster [92]. While this scale of simulation is appropriate for modeling networks in invertebrate locomotion [50] or population-level modeling of vertebrate locomotion [35, 64], it is difficult to simulate networks of higher scale such as those used for visual processing [93, 110]. Additionally, SNS-Toolbox is not designed for parameter optimization and is not optimized for layer-based structures.

In this work we present SNSTorch, a companion package to SNS-Toolbox for larger networks. SNSTorch is built in PyTorch [95] to simulate the same non-spiking neural dynamics as SNS-Toolbox, and does so in a layer-based manner which is conducive to numerical optimization via automatic differentiation. As of the writing of this manuscript, no other tool exists for simulating and optimizing large networks with conductance-based synapses.

5.3 Methods

While there are a wide variety of neuronal and synaptic dynamics implemented in SNS-Toolbox, we have initially focused on implementing non-spiking neurons and their synapses in SNSTorch due to their reduced computational complexity and faster simulation speed. Additionally, non-spiking neurons are more amenable to training via gradient backpropagation than spiking neurons and synapses due to not requiring a surrogate gradient. All of the implementations described in this section are implemented using the PyTorch framework due to its popularity and ease of use, and all of SNSTorch is compatible with the built-in autodifferentiation engine in PyTorch for optimization.

5.3.1 Neural Dynamics

As in SNS-Toolbox, we base our implementation of non-spiking neurons after the following dynamics,

—— (5.1)

where is the membrane capacitance, the membrane conductance, and is the resting potential of the neuron. is an injected current of constant magnitude, is any external applied current, and is the current induced via synapses from any presynaptic neurons.

For simulation, we discretize the dynamics in 5.1 via the forward-Euler approximation as

_ _ 1 _ _ _ 1 _ _ _ (5.2)

where is the state, is the time constant, is the leak rate, is the resting state, is the constant bias, and is any input coming from outside the neuron. In comparison to SNS-Toolbox, this parameterization was done for training to be semi-independent of the specific simulation timestep.

5.3.2 Synaptic Connections

Three types of syanptic connections are currently supported in SNSTorch, all of which are based on the dynamics of a non-spiking chemical synapse:

(5.3)

where is the synaptic reversal potential and is the instantaneous synaptic conductance as a function of the presynaptic voltage . In theory any PyTorch activation function can be used to define the synaptic conductance, although the default function and the one exclusively used in this work is the piecewise sigmoid

0 ——— (5.4)

is the maximum synaptic conductance, and voltages and define the range of presynaptic voltages where the synaptic conductance depends linearly on the presynaptic neuron's voltage.

These synapses are supported in three different permutations: dense, element-wise, and convolutional. Each synapse type acts as a function which takes in the

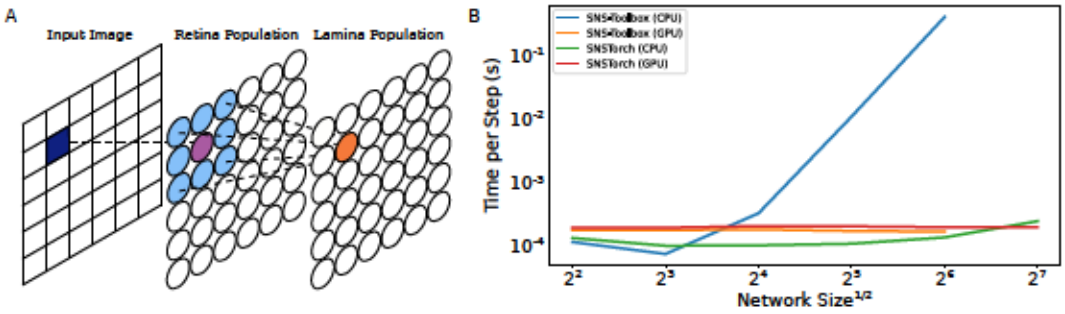


Figure 5.1: Comparison in performance between SNS-Toolbox and SNSTorch. **(A)** The network to be evaluated is the same structure as section, with two populations being connected by a convolutional synapse. **(B)** This network was compiled and then run in SNS-Toolbox and SNSTorch at increasing population size.

presynaptic and postsynaptic states and applies the dynamics in eq. 5.3, with each applying the conductance and reversal potentials based on differing connectivities. The dense connection implements a synapse from every presynaptic neuron to every postsynaptic neuron, the elementwise connection only connects neurons with the same index, and the convolutional connection reuses a local pattern that is tiled across the population.

5.4 Results

To test SNSTorch, we first compare the performance of this system with SNS-Toolbox. We then evaluate the functionality of SNSTorch on two separate optimization tasks.

5.4.1 Comparison with SNS-Toolbox

Although SNSTorch simulates some of the same dynamics as SNS-Toolbox, internally they are structured differently. SNS-Toolbox makes no assumptions about connectivity and represents the entire network as a single recurrent population, which lends it well to densely recurrent networks with nested feedback loops such as those in locomotion [68]. SNSTorch on the other hand is designed in the more

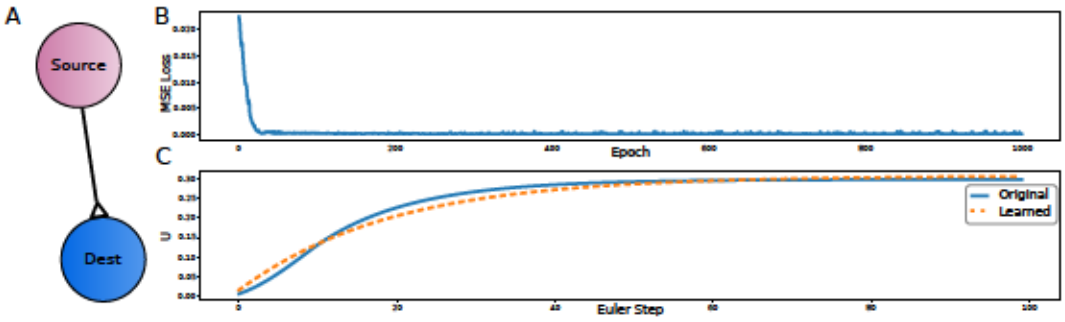


Figure 5.2: Using SNSTorch for a parameter identification task. (**A**) We are trying to match the behavior of a simple network of neurons, where one neuron receives a random stimulus and excites the other neuron via an excitatory chemical synapse. Using a ground truth model, the network learned the neural and synaptic properties to replicate this behavior. We chose to focus on minimizing the mean-squared error between the final state of the original and trained network. Shown in (**B**) is the training loss over 1000 random stimuli, and in (**C**) we plot the trajectory of the postsynaptic neuron in the original and trained networks.

conventional fashion in deep learning, where each population is represented and evaluated individually as its own layer. While this means arbitrary connectivity is more challenging, the payoff is in higher speed and reduced memory consumption for large networks. To demonstrate this, we simulated the same network structure using SNS-Toolbox and SNSTorch, running it on the CPU and GPU and varying across large differences in network size. Results can be seen in Fig. 5.1. For small networks, SNS-Toolbox runs faster than SNSTorch. As the network size increases, SNSTorch quickly becomes the fastest solution (or only solution, as memory consumption increases).

5.4.2 Parameter Tuning and Regression

As a toy example, we use SNSTorch for parameter tuning in a simple network (shown in Fig. 5.2A). Two neurons are connected via a chemical synapse, and the presynaptic neuron is stimulated with a constant input. In this task, we have two versions of this network: one which acts as the target, and the other which must be trained to match the target. This system has 12 parameters: , , , , and 0 for

both neurons, and the synaptic conductance and reversal potential. For each batch of random inputs, we evaluate the loss as the mean-squared error between the final state of the destination neuron in both networks:

$$\frac{1}{} \qquad ^2 \tag{5.5}$$

This loss is then passed through the network using backpropagation through time [136]. We used the Adam optimizer [72] for training, with a learning rate of 0 001. Training converged relatively quickly to a near solution, with the training performance shown in Fig. 5.2.

5.4.3 Sequential Classification

In order to showcase the optimization capabilities of SNSTorch, we expanded our optimization problem to one with orders of magnitude more parameters. We trained and evaluated a recurrent SNS on the row-wise sequential MNIST [36] hand-written digit dataset, where the image is split into 28 rows and fed into the model sequentially. The structure of the SNS (shown in Fig. 5.3A) is a single recurrent layer with 128 non-spiking neurons, which receives a synaptic current vector and input digit row every step. For training we used the Adam optimizer with a learning rate of 0.001, and evaluated the loss as the cross entropy between the maximally active output and the target class. We compared this performance with a traditional RNN, which had a hidden size of 177 in order to match the number of parameters in the SNS, and the results are shown in Fig. 5.3. The SNS is able to learn up to 95 percent accuracy, although it does converge more slowly than the traditional RNN.

5.5 Discussion and Future Work

In Chapter 3, we presented SNS-Toolbox as a tool for designing and simulating networks of synthetic nervous systems of neurons and synapses with biologically plausible dynamics. While SNS-Toolbox is effective in its role, it has difficulty

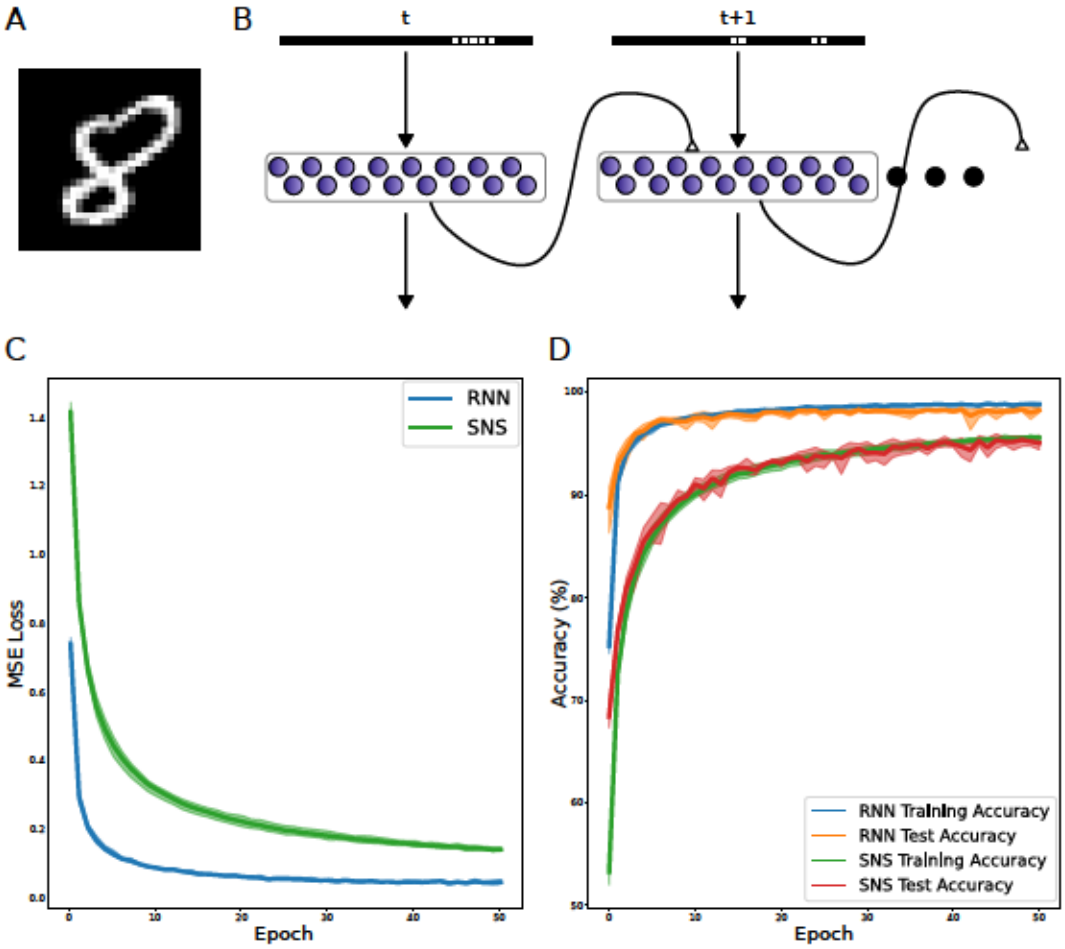


Figure 5.3: Training an SNS for sequence classification. (**A**) We train an SNS network to classify the row-wise sequential MNIST dataset [78], where each handwritten digit is divided into 28 1x28 images. (**B**) The SNS network consists of a single recurrent layer of non-spiking neurons, with the recurrence implemented using chemical synapses. Training loss (**C**) and accuracy (**D**) of the SNS network and an RNN with a similar number of parameters. Line denotes the mean across five trials, the shaded area denotes the fifth and ninety-fifth percentiles.

when simulating large networks of neurons which could be structured based on layers of populations. It is also primarily a tool for design, so it is not optimized for optimizing parameters without recompiling the network. In this chapter, we present SNSTorch as a companion software package to SNS-Toolbox. SNSTorch simulates some of the same dynamics as SNS-Toolbox, while being able to simulate larger networks and supporting optimization via automatic differentiation in PyTorch. We

demonstrated its efficiency compared to SNS-Toolbox on a two-layer convolutional network, and then evaluated its ability to train on both regression and classification tasks.

In its current state, SNSTorch only supports neurons and synapses with non-spiking dynamics. In order to match some of the versatility of SNS-Toolbox, in future work we will incorporate all of the neural and synaptic dynamics implemented in SNS-Toolbox in SNSTorch. While some of these dynamics may be difficult or ineffective to train with gradient backpropagation, there should still be a performance improvement in SNSTorch over SNS-Toolbox for simulating these dynamics at scale in the forward direction. Once all of these dynamics are supported, we will also provide a conversion tool which will map each population in SNS-Toolbox into a corresponding layer in SNSTorch.

One of the major differences between SNSTorch and SNS-Toolbox is its support for numerical optimization, particularly gradient backpropagation, which is not natively integrated into the workflow for SNS-Toolbox. Even though gradient backpropagation is a powerful technique for optimizing neural networks, it is inherently a mathematical solution which does not occur in biological brains. Due to the flexibility of being able to design new connection layers, in future work we will incorporate synapses with local and modulated learning rules such as timing-dependent plasticity. This will enable designers to simulate SNS networks which train and adapt online.

Chapter 6

FLYWHEEL: A MOBILE ROBOT FOR TESTING MODELS OF FLY MOTION CONTROL

Material in this chapter is under review for the 13th Conference on Biomimetic and Biohybrid Systems as

- Nourse, W. R. & Quinn, R. D. (2024). FlyWheel: A Robotic Platform for Modeling Fly Visual Behavior.

Edits have been made to place this material into context with the rest of this dissertation.

6.1 Abstract

An ongoing problem in robotics is the calculation of body motion given motion in the visual field, also known as ego-motion estimation. This is a problem which has been solved in the visual system of most animals, including the fruit fly *Drosophila melanogaster*. Here we present FlyWheel, an open-source robotic platform for studying models of the visual motion-processing system in insects. We showcase a dataset of rotational motion data in real-world conditions using the robot, and use a simplified model of the motion pathway in *Drosophila* as a baseline for further comparison and development.

6.2 Introduction

An important problem which must be solved for navigation in both robotic systems and animals is visual ego-motion estimation, or the ability to use motion in the visual field to calculate how the body is moving through a fixed world [71]. Calculating ego-motion based on sequences of images has been a longstanding challenge in the field

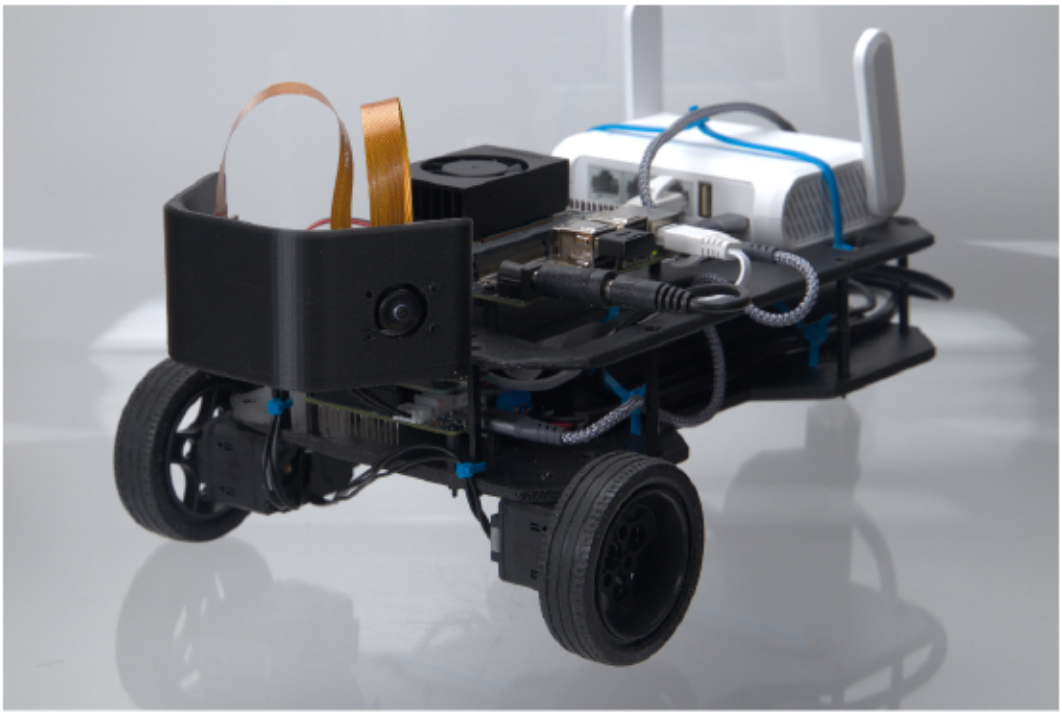


Figure 6.1: FlyWheel, a mobile robot for testing models of motion control in flies.

of computer vision, with the majority of successful solutions being built off of either measurements of optic flow or point correspondence. Initial methods estimated rotation then translation based on changing flows of point triplets [98], followed by methods which took the first derivative of image brightness in regions of interest [62]. Further methods were developed which used motion parallax between pairs of images, either to compute changes in depth [103] or image deformation [128]. In recent years convolutional neural networks have also been used, either as an end-to-end solution [30] or starting from an initial optical flow field [143].

An organism often studied for its motion-vision processing is the fruit fly *Drosophila melanogaster* [112, 115, 125], due to having a nervous system with significantly fewer neurons than vertebrates [13, 79] as well as recent efforts to create a full brain connectome [108, 137]. *Drosophila* and other insects generate an estimate of directional velocity throughout the visual field using the differences in timing between adjacent neurons in response to changing amounts of brightness,

with sections being sensitive to increases in brightness and others to decreasing brightness [13]. Initial models of this mechanism for directional selectivity consisted of two-pixel (or arm) detectors which either enhanced motion in the preferred direction [57] or suppressed motion in the opposite direction [8]. Many bio-inspired algorithms have been developed which are based off of these two-pixel motion detectors, and have been successfully used for applications including quadrotor flight control [142] and target tracking [7]. It is also possible to estimate the velocity of natural images by combining multiple of these two-pixel detectors which are tuned to different spatial frequencies [21].

As more detailed information has become available about connectivity within the Drosophila optic lobe, it has become clear that the motion detectors are implemented as three-arm detectors, not two, allowing the combination of preferred-direction enhancement and null-direction suppression within a single circuit [54, 115]. In the work of Nourse et al. [93], a simplified version of this three-arm detector was used to calculate the velocity of moving square-wave gratings using a network of neurons with bio-inspired dynamics. Currently however, no three-arm detector system has been used for any robotics application.

In this work, we present the open-source robot FlyWheel, a platform for testing models of insect motion vision. We showcase a dataset of video clips collected during rotation on the robot, and provide results of a neural three-arm motion detector as a baseline for further improvement.

6.3 Robot Design

FlyWheel is built from three subsystems: central computation, a wheeled base, and visual input. Each of these components was designed to be modular, and can be removed and replaced on the robot. The robot body is fabricated using 3d-printed PLA on a consumer-grade printer. The hardware and software for FlyWheel is available at https://github.com/wnourse05/FlyWheel.

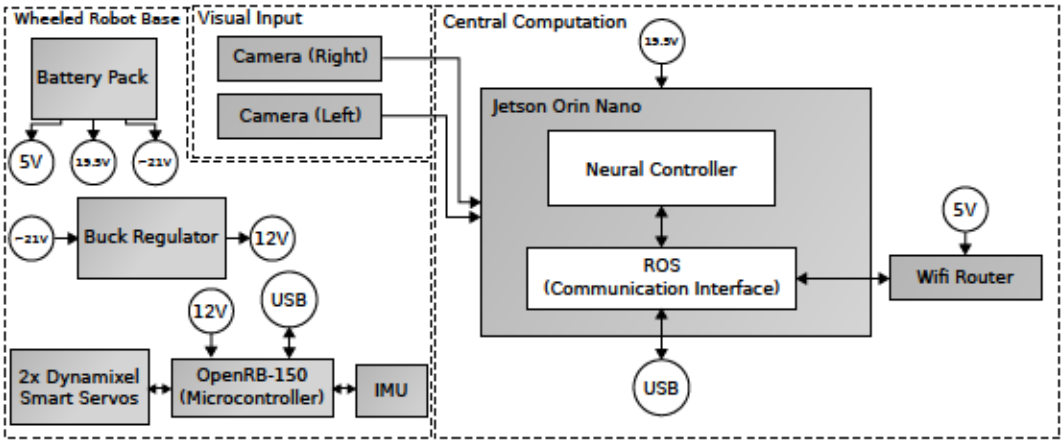


Figure 6.2: System diagram of hardware and software components for FlyWheel. There are three subsystems: visual input, central computation, and a wheeled base. Each of these components is modular, and can be removed and replaced on the robot. The visual input consists of two 160 degree FOV cameras, arranged to have a similar stereo FOV as *Drosophila melanogaster*. The wheeled base provides power to the system, and has two Dynamixel (Robotis Co. Ltd., Seoul, South Korea) smart motors to provide propulsion. The central computing platform runs a ROS framework on an NVIDIA Jetson Orin Nano (NVIDIA, Santa Clara, CA), with a small wireless router as an external access point.

6.3.1 Central Computation

The Central Computation module onboard FlyWheel is responsible for communicating with all of the sensors and actuators on the robot, as well as running any desired control algorithms. We chose to use an NVIDIA Jetson Orin Nano (NVIDIA, Santa Clara, CA) as our embedded computer due to its relative affordability and support for GPU acceleration, although the mechanical design would support the less expensive and less powerful Jetson Nano as well. For communicating with the cameras, motors, and any external computers, the Jetson Orin Nano uses a custom library built in the Robot Operating System (ROS) [99]. The combined camera feed is published as a rostopic, along with the status of a wireless game controller and commanded velocities for the motor system. Users can access the robot remotely via a miniature wireless router onboard.

6.3.2 Wheeled Base

The wheeled base has two primary roles: to move the robot, and to provide power to the embedded computer. Power comes from a lithium-ion battery bank, which provides 5 volts DC for the wireless router, 19.5 volts DC for the Jetson Orin Nano, and a 21-29 volts DC output which is regulated down to 12 volts DC for the motors. The motors are a pair of Dynamixel XL430 (Robotis Co. Ltd., Seoul, South Korea) smart motors, allowing precise control of the wheel rotational velocities. The motors receive commands from the Jetson Orin Nano using a Robotis OpenRB-150 microcontroller board. This board also sends rotational velocity information to the Jetson Orin Nano based on readings from a Bosch BNO055 inertial measurement unit (Robert Bosch GmbH, Gerlingen, Germany). The wheels are positioned in a unicycle-model configuration on either side of the robot, in order to reduce the distance between the axis of rotation and the cameras.

6.3.3 Visual Input

Since FlyWheel is designed for testing models of insect vision, it is important that the robot has a visual system which replicates that of the model insect as much as possible. Each of *Drosophila melanogaster*'s eyes has a field-of-view (FOV) of approximately 144 degrees [133], and are combined to produce an overall FOV of 270 degrees with a stereo overlap of 17 degrees [117, 119]. To replicate this, FlyWheel uses two 160 degree FOV IMX219 cameras from Yahboom (Yahboom, Shenzhen, China). These cameras are arranged to produce a net FOV of 286 degrees with a stereo overlap of 34 degrees as shown in Fig. 6.3.

Although flies are able to improve the resolution of their vision through vibrating their photoreceptors [44], they still have a fairly small visual resolution compared to modern camera sensors. Each eye consists of approximately 800 facets or ommatidia [75] arranged in a hexagonal pattern [20], each with a receptive angle of five

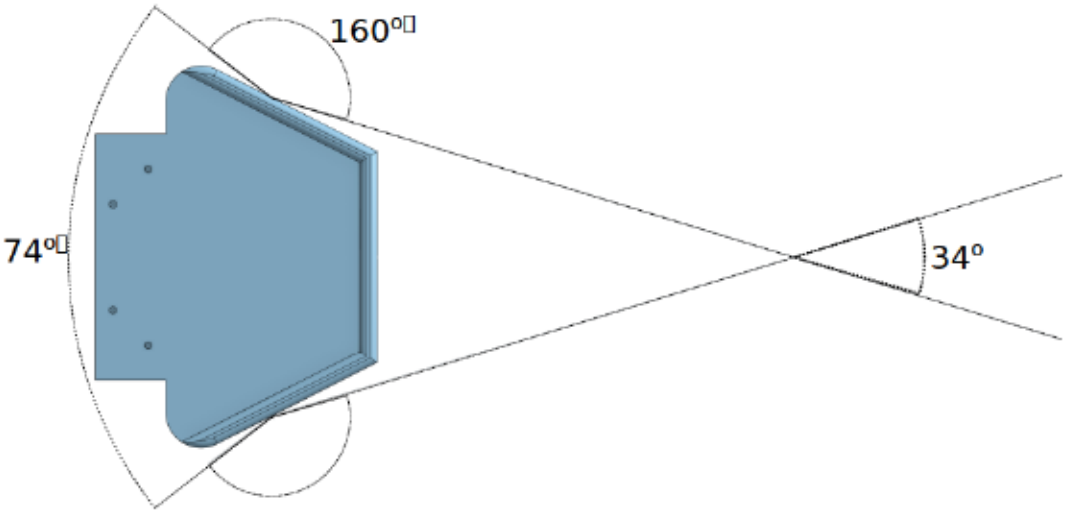


Figure 6.3: FlyWheel field of view (FOV). Each eye has a 160 degree FOV, and is arranged to produce a net FOV of 286 degrees with a stereo overlap of 34 degrees.

degrees [13]. In total, each eye can be approximated as having a visual resolution of 25x32 pixels. Additionally, most *Drosophila melanogaster* photoreceptors are sensitive to green wavelengths of light [25, 116], although others are sensitive to blue light [104, 114]. To replicate this on FlyWheel, we first strip the red and blue channels from the images captured by the cameras. We then concatenate the stereo pair of images side by side into a single image, and then downsample the image using nearest-neighbor interpolation. While not as accurate as area-based interpolation, the processing time of nearest-neighbor interpolation is significantly faster. For a comparison of latency between both interpolation methods across different final resolutions, please refer to Fig. 6.5A. The final resolution of the stereo image is 24x64 pixels, the dimensions which are closest to that of *Drosophila melanogaster* while still having satisfactory performance. The entire image processing pipeline runs at a rate of 30 frames per second (FPS), with a processing latency of 6 ms. For a visual example of the final images, please refer to Fig. 6.4.

Figure 6.4: Example stereo video frames after processing. Stereo pairs are concatenated into a single image, converted to greyscale, then downsampled from the native resolution of 1232x3280 to 24x64 pixels.

6.4 Motion Vision Dataset

A goal of FlyWheel is to test models of ego-motion estimation in *Drosophila melanogaster*, specifically for lateral steering. We collected a dataset of rotational motion data by placing the robot in multiple different interior locations with varied lighting conditions and commanded it to spin in place while varying its rotational speed between 0.1 and 0.5 radians per second. Using this paradigm, we created a dataset of 2,646 one-second long clips which are labeled with their corresponding rotational velocity, and where twenty percent of the clips are reserved for validation. Through data augmentation including spatial and temporal reversal, we have expanded this set to 21,168 labeled clips. This dataset is freely available at https://github.com/wnourse05/flywheel-rotation-dataset.

6.5 Motion Processing Network

As a base system of performance, we implemented a motion vision processing network in SNSTorch () based on the motion vision circuitry in the optic lobe of the fruit fly *Drosophila melanogaster*. We used the same techniques as Nourse et

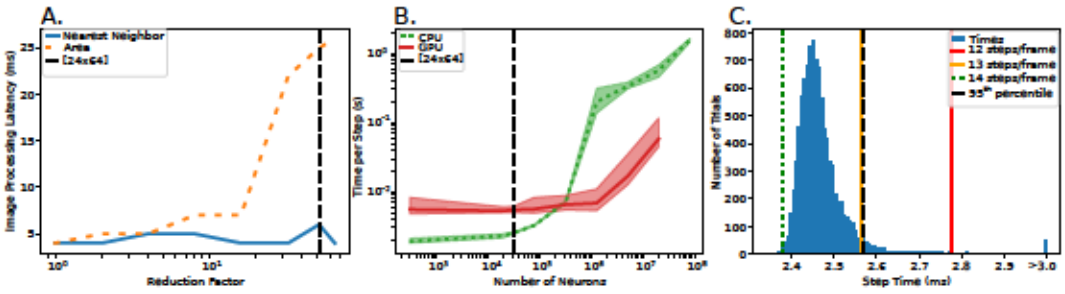


Figure 6.5: Timing performance of image formatting and processing execution on target hardware. **A:** Latency in image processing as the target image reduces in size. Two different interpolation methods are compared, with nearest-neighbor interpolation shown in solid blue and area interpolation shown in dashed orange. A vertical dashed line is present at the image resolution 24x64, the scaled dimensions used in our dataset. **B:** Time per simulation step of our visual motion processing network, in seconds, as the dimensionality of the input increases. Execution on the Jetson Orin Nano CPU are shown in dotted green, and times for the Jetson Orin Nano GPU are shown in solid red. Dark lines correspond to the average, the shaded area corresponds to the 5th and 95th percentiles over 1000 steps. We use a vertical dashed line to denote the dimensionality corresponding to an input image size of 24x64 pixels. **C:** Detailed timing of our network with an input dimensionality of 24x64 pixels. Shown is a histogram of time per simulation step in milliseconds, over a testing run of 10,000 steps. A black dashed vertical line denotes the 95th percentile of the distribution. Shown in dashed green, solid orange, and solid red would be the time per step needed for 14, 13, or 12 simulation steps per video frame. In this work we chose to use 13 steps per frame for our simulations.

al. [93], with some adjustments to account for the use of natural images instead of simulated square gratings. The full network is shown in Fig. 6.6. In this section we will begin with an overview of the neural modeling techniques employed, and then will examine the design of each individual network section, emphasizing the changes made in this work. The network and all remaining support code can be found at https://github.com/wnourse05/FlyWheelBaseline-LivingMachines2024.

6.5.1 Neural Modeling

We choose to implement our motion-vision processing network as a Synthetic Nervous System (SNS) of non-spiking leaky integrator neurons, where the neural

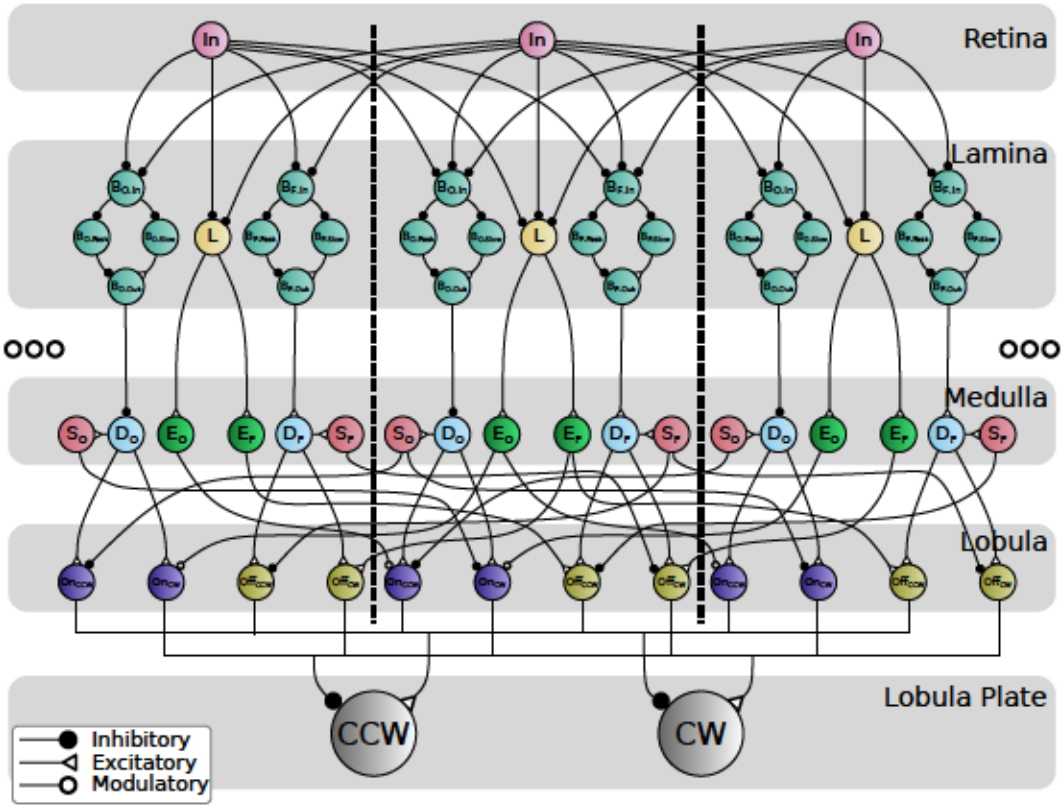


Figure 6.6: Visual motion processing network used in this work, inspired by the anatomy of *Drosophila melanogaster* and adapted from [93]. Visual stimuli are encoded into a neural representation in the retina. They are then spatiotemporally filtered in the lamina, and temporally filtered again in the medulla. The lobula combines the neural activity in the medulla into estimates of motion at each pixel, and these estimates are summed across the entire visual field to generate a global estimate of motion in the lobula plate.

state is updated as

(6.1)

where is the neural time constant, is any external input, and is a constant bias term. is the synaptic input from any presynaptic neurons in the network,

(6.2)

with denoting a presynaptic neuron, and denoting the synaptic reversal potential. Throughout the network, we design for neurons to communicate when

their state is between 0 and , with 1 in this work for numerical simplicity. All excitatory synapses in our model have 5 , 2 for inhibitory synapses, and modulatory synapses have a reversal potential of 0. is a monotonic function which describes the incoming synaptic conductance, defined as

0 ——— 1 (6.3)

where 0 and are the lower and upper threshold states of synaptic activity.

Taking the model in equation 6.1, if we set to 0 we find the steady-state response [121] as

— (6.4)
1 —

Synaptic Pathway Designs

In the SNS network, we tune many of the synapses using one of three analytic rules. The first is for signal transmission, where the target value of the postsynaptic neuron from eq. 6.4 is the steady-state voltage of the presynaptic neuron multiplied by a transmission gain . As seen in Szczecinski et al. [121], this can be solved as

——— (6.5)

where is either or depending on the role of the synapse.

An alternative formulation is to set a target state of the postsynaptic neuron, depending on the presence of external inputs. This is shown in Nourse et al. [93] as

——— (6.6)

The other analytic synapse design in this work is a modulatory one, meant to modulate the sensitivity of a postsynaptic neuron to external and synaptic inputs. This can be designed following Szczecinski et al. [121], setting the synaptic conductance as

1 (6.7)

and using the modulatory reversal potential . In this formulation, the steady state is divided by when .

For simulating these dynamics, we discretized equations 6.1-6.3 into a forward-euler formulation, and simulated them as individual layers in the PyTorch numerical simulation software [95] in order to support execution on either a CPU or a GPU.

6.5.2 General Network Properties

Our first step in adapting the network from Nourse et al. [93] to this task was to determine how fast the network could run. To do this we evaluated the network over multiple different input image dimensions, and recorded the execution time for each simulation step over 1000 steps. We compared the step simulation time with increasing image size to the image processing latency with decreasing image size, and determined an input size of 24x64 pixels to be an appropriate balance of processing speed and biorealism. From here, we simulated the network for that input dimensionality for 10,000 steps in order to find a realistic time per step to base our simulations upon. We found that 95 percent of all trials could be accounted for with a timestep of 2 56 , equivalent to an update rate of 390 Hz or 13 steps per input frame. These results can be seen in Fig. 6.5.

6.5.3 Retina

The Retina converts the image stream into a neural representation. To do this, we create a layer of neurons the same size as the input image. For the temporal properties, we set the time constant from equation 6.1 as small as possible while maintaining numeric stability. We denote this as , and empirically found this to be 15.4 ms.

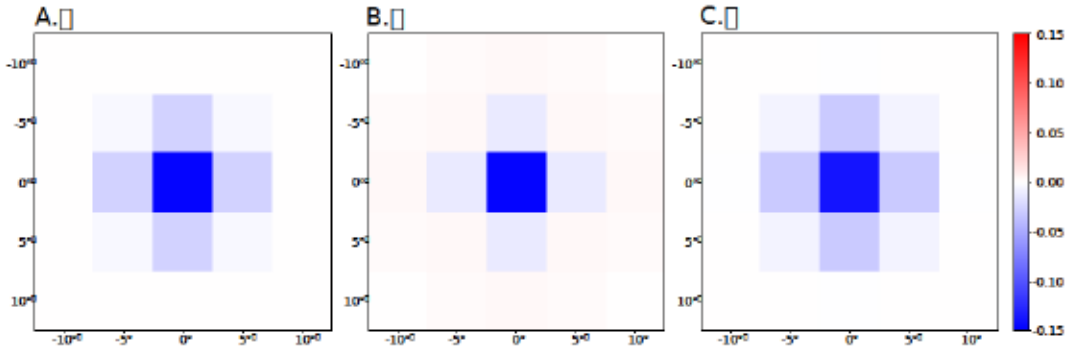


Figure 6.7: Receptive fields of the second layer in our visual motion processing network. The fields for B_O(**A**), L(**B**), and B_F(**C**) are based on gaussian parameterizations of receptive fields between the retina and lamina in *Drosophila melanogaster* [4]

.

6.5.4 Lamina

The Lamina takes the image representation from the retina and applies a bank of spatiotemporal filters. The visual processing system is also first split into the On and Off pathways, with each pathway having a bandpass network and sharing a lowpass filter.

Spatial Filtering

The most significant change in this work over that of Nourse et al. is that we implement the spatial filtering which was ommitted in that work. To perform spatial filtering of the image representation in the retina, each of the lamina input neurons integrates across a 5x5 grid of retinal neurons. Given that the retina encodes the image brightness at each pixel as a neural state between zero and R, we want to ensure that the inputs into the lamina layer remain stable. Specifically, we aim to bound $U^* \in [0, R]$. In the trivial case where all presynaptic neurons are at rest, the steady-state postsynaptic voltage from eq. 6.4 is $U^* = B$. When all presynaptic

neurons are at a maximum state of R, eq. 6.4 becomes

$$\frac{}{1} \tag{6.8}$$

With the inputs to , , and , we are interested in the inhibitory case, where 0 and .

$$0 \quad \frac{}{1} \tag{6.9}$$

Rearranging, we find that the sum of the synaptic conductances multiplied by the reversal potentials must be bounded by .

(6.10)

To do this, we choose to parameterize the sum on the left hand side of eq. 6.10 as a probability density function (PDF) scaled by -R, as all PDFs by definition have a combined sum of 1. Any PDF could be chosen, but in order to replicate experimental results in *Drosophila melanogaster* we parameterize the sum as a difference of gaussians

$$\frac{1}{\overline{2}} \quad \frac{^{2\ 2}}{^{2\ 2}} \quad \frac{}{\overline{2}} \quad \frac{^{2\ 2}}{^{2\ 2}} \tag{6.11}$$

where and are the standard deviations of the center and surround gaussians, and 0 1 is a scaling coefficient [4]. As is constrained to a finite set of choices, we divide equation 6.11 by the corresponding based on its sign to arrive at .

Temporal Filtering

Within the lamina, the On and Off pathway each have a bandpass filter network and share a neuron acting as a lowpass filter. For the lowpass filter ( ), for simplicity we set its membrane time constant to since it will be further filtered later on in the Medulla. To implement a bandpass filter, we take the difference between two non-spiking neurons of different time constants (and). We parameterize

both bandpass filters identically, with the fast pathway being set to and the slow pathway being approximately five times slower.

6.5.5 Medulla

In the medulla, the spatiotemporal filtering performed in the lamina is temporally filtered again. In both the On and Off pathways, there are three neurons responsible for direct stimulation, suppression, and enhancement. The direct and enhancement (and) neurons receive synaptic input from the bandpass and lowpass (and) neurons, while the suppression ( ) neuron receives input from the direct neuron. Each of these neurons behaves as a lowpass filter, with the temporal properties tuned following the procedure in Nourse et al. and assuming a spatial resolution of five degrees and a rotational speed of 0.1 to 0.5 radians per second.

6.5.6 Lobula

The role of the circuitry in the lobula is to convert the filtered representations in each column of the medulla into an estimate of the velocity at each pixel. The enhancement, direct, and suppression neurons from adjacent columns are combined to behave as a three-arm elementary motion detector [54]. In the On pathway, excites the motion detector whenever an increase in brightness passes over the column. This excitation is dampened by a modulatory synapse from in the precending column (eq. 6.7), and sharply inhibited by in the next column in the preferred direction. The mechanism is nearly identical in the Off pathway, with the only difference being that is excitatory instead of modulatory. As the velocity of the stimuli increases, the magnitude of the response in the motion detector decreases as the time between excitation and inhibition decreases.

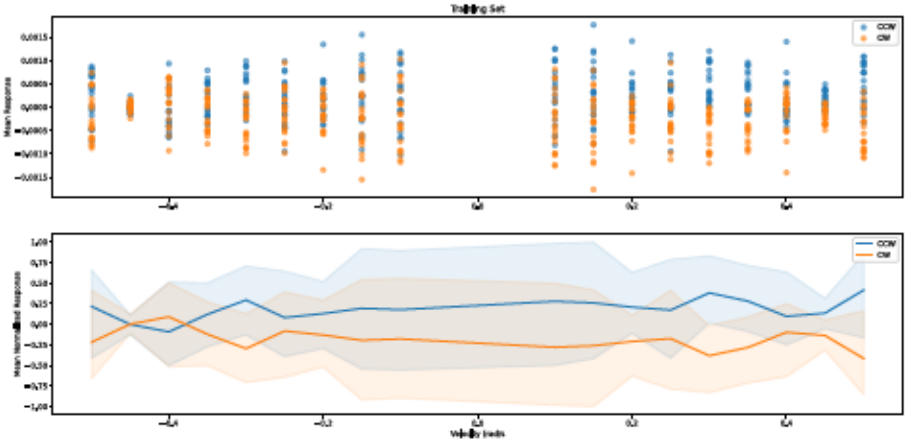


Figure 6.8: Performance of the simple motion vision processing network on the video clips in the test portion of the FlyWheel dataset. **(A)** Scatterplot of average neuron state for the clockwise and counter-clockwise neurons for each image sequence. **B.** Curves denote the mean neural response of all trials at each velocity, shaded area represents the 5th and 95th percentiles. All data is normalized to the maximum of the 95th percentile across all velocities.

6.5.7 Lobula Plate

In the final layer, we extend the network presented in Chapter 4 to include an approximation of the circuitry present in the *Drosophila melanogaster* lobula plate. We add two horizontal sensitive neurons, and , one corresponding to counter-clockwise rotation and the other to clockwise rotation. These neurons receive synaptic input from every motion detector neuron in the lobula, with the counter-clockwise sensitive detectors exciting and inhibiting , and the inverse case for clockwise sensitive detectors [13]. These neurons have a time constant of , and the synapses are tuned using eq. 6.5 such that the sum of all the synaptic gains is one.

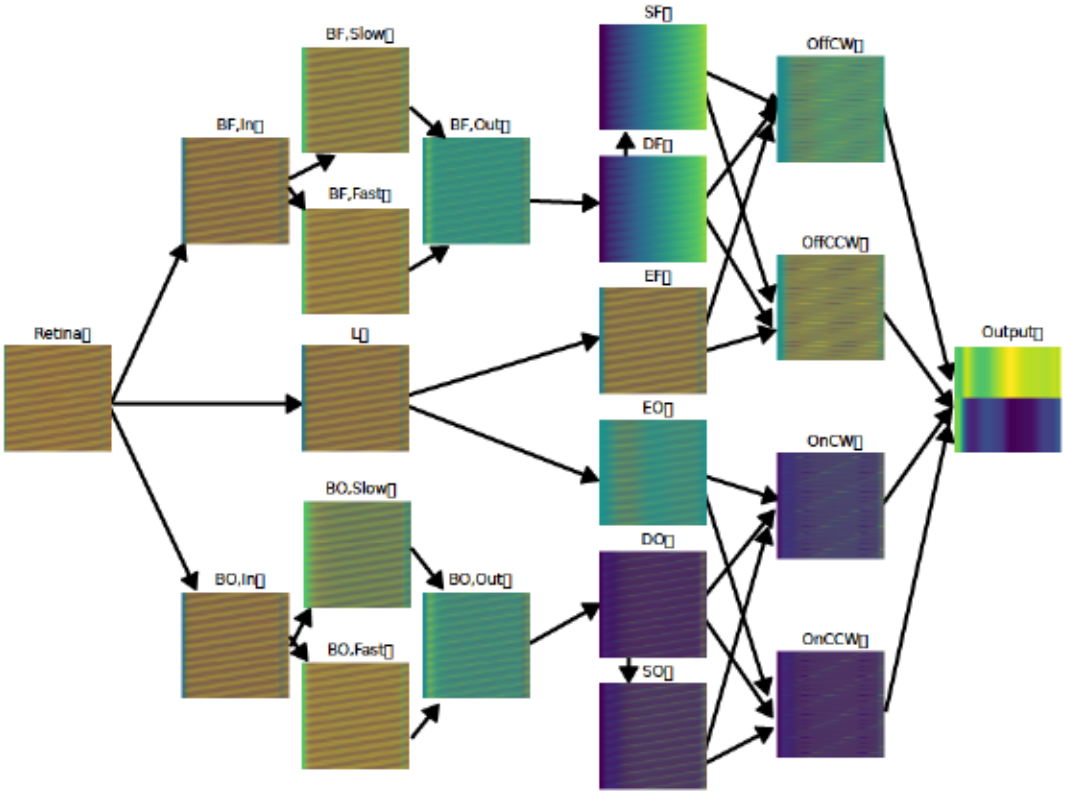


Figure 6.9: Activity within each population over the course of a horizontal grating stimulus. Within each plot, the vertical axis represents the different neurons in the population and the horizontal axis shows the progression of time. Brighter colors denote higher neural state.

6.6 Results

Using the test portion of the dataset described in Section 6.4, we evaluated the performance of the visual motion processing network described in Section 6.5. We simulated the network with 5x5 pixel receptive fields, and for each sample clip we let the network states stabilize to the first video frame and then recorded the neural state of the readout neurons CCW and CW over the remaining frames. Across the test set, the network correctly identified the direction of rotation 43% of the time, with a preferential bias towards counter-clockwise rotation. Performance for each velocity is shown in Fig. 6.8.

Using SNSTorch (Chapter 5), we attempted to improve the performance of our

motion processing network by training it using gradient backpropagation through time (BPTT) [136]. These results are omitted, as the network with this architecture did not converge to a classifier with a success rate of more than 50%. Additional neurons were also added to better approximate the architecture seen in Fig. 6.6, but did not improve performance. In order to validate the new elements in the network's lobula plate which were not implemented in Chapter 4, we trained the network on a shifting square wave grating. Using autodifferentiation, the network successfully trained the synaptic strengths between the lobula and the lobula plate as well as between the medulla and the lobula. When the moving grating is applied to the retina, the activity over time is filtered by the spatiotemporal filters in the lamina and the temporal filters in the medulla. This results in the output layer converging to one active neuron, representing translation in that direction. Example data from one trial can be seen in Fig. 6.9.

6.7 Discussion

In this work, we present the robotic platform FlyWheel. FlyWheel is a wheeled robot with a binocular camera system designed to mimic the FOV of the fruit fly *Drosophila melanogaster*. We collected a dataset of video sequences across a range of turning velocities, and then implemented a visual motion processing SNS network to discriminate between counter-clockwise and clock-wise rotation over this dataset. The hardware and software for FlyWheel are open-source along with the dataset, and we believe that FlyWheel can be a valuable platform for benchmarking and studing models of motion processing and navigation in *Drosophila melanogaster* and other insects.

As implemented, our processing network had difficulty successfully identifying the direction of global rotation in natural images, exhibiting a bias in sensitivity towards counter-clockwise rotation. Additionally, this estimate does not allow the discrimination of rotational speed. This is not surprising, given that the processing

network was adapted from one made for square-wave gratings with little additional tuning. We provide this network as a base for comparison, but a goal of future work will explore more complex architectures to better tune the network for natural image sequences. Some elements which could be added to improve accuracy include additional inputs to the motion detectors [21], recurrence and feedback within and between layers [14], or synaptic feedback loops which are found within the Off pathway in *Drosophila* [17]. This dense pattern of recurrent connectivity may be the key to implementing such complex visual behavior in a comparatively small network, with the recurrence suggested to improve contrast insensitivity [14].

Additionally, visual processing is not the only mechanism by which insects determine their rotational velocity, with feedback coming from proprioception and other sensory modalities influencing this calculation [111]. Due to the high variability of natural scenes with differing levels of depth and parallax, it is possible that the visual system is only responsible for sending a corrective signal to the motor nervous system when significant motion is presented relative to the animals current rotational velocity. An area for future exploration is the integration of these sensory systems in combination with other sources of feedback.

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Summary

In Chapter 3, I presented SNS-Toolbox as an open-source software package for designing synthetic nervous systems (SNS) and simulating them on consumer-grade hardware. SNS-Toolbox implements a wide variety of neural and synaptic dynamics based on bio-plausible dynamics, including spiking neurons, chemical synapses, and electrical synapses, and can simulate networks with hundreds to thousands of neurons in real-time. This performance is competitive with all other neural simulators which exhibit the same range of neural and synaptic dynamics as SNS-Toolbox. In addition to simulation of neural and synaptic dynamics, I also presented two examples of using SNS-Toolbox for the control of an external system: one on controlling the navigation of a mobile robot in the Robot Operating System (ROS), and the other on controlling a biomechanical system in the physics simulator MuJoco.

In Chapter 4, I used SNS-Toolbox to design and simulate an SNS network for processing visual motion. In this model, changes in visual stimuli are processed into two pathways focusing on either increases or decreases in brightness. After multiple layers of temporal filtering, these signals are combined across local pixel groups to generate a local estimate of motion. This network was based on the available connectomic information for the optic lobe in *Drosophila melanogaster*, and serves as a minimal neural representation of the computation present in that neuropil. A key feature of this network is the multiplicative effect of modulatory or shunting inhibition, a computation which is not possible using purely weight-based synapses. I evaluated this network on sequences of translating binary gratings, and

demonstrated local directional sensitivity across all cardinal directions.

Due to the computational considerations in scaling the network from Chapter 4 beyond a small group of pixels, in Chapter 5 I presented SNSTorch as a solution for simulating large-scale SNS networks. SNSTorch is a layer-based simulator for neural populations with chemical synaptic connections, which exhibits faster simulation speed than SNS-Toolbox while exhibiting a lower memory overhead. Additionally, it improves upon SNS-Toolbox through the more native support of optimization tools. In support of this, I presented two examples of SNS networks being trained: one example of regression-based parameter tuning in a tiny network, and the other showcasing the training of a larger recurrent SNS for sequential classification.

Finally, in Chapter 6 I presented ongoing efforts to adapt the circuitry in the *Drosophila melanogaster* optic lobe to estimate global motion based on sequences of natural images. First I presented FlyWheel, a mobile robot which uses two wide-angle cameras to approximate the stereo-visual properties of a fruit fly. I used this robot to generate a dataset of natural image sequences which correspond to a range of rotational velocities onboard the robot, and using data augmentation techniques extended this dataset to contain over 21,000 labeled samples. Next I presented a first attempt at using SNSTorch from Chapter 5 to simulate and optimize the network developed in Chapter 4. While the network was able to estimate global rotation direction on a simulated grating, further work is necessary to expand the network such that it can process natural image sequences.

7.2 Impact and Future Work

The primary focus of this dissertation has been the development of SNS-Toolbox and SNSTorch for designing and simulating SNS networks. SNS-Toolbox marks the first time that it has been possible to simulate these heterogeneous networks at a large enough scale to model interesting circuits found in animals for control while being able to interact with a wide variety of systems in real-time or faster. This ease

of interaction with external systems, as well as the cross-platform compatibility of the software, has facilitated the use of SNS-Toolbox across a wide variety of research projects for both research and course projects (a selected list is presented in Appendix B). The neural dynamics present in SNS-Toolbox expand what could be modeled in the SNS framework as well, with the addition of electrical synapses allowing the potential for implementing dendritic computing [23].

With the development of SNSTorch, I have developed the first software which incorporates chemical synapses into large-scale neural networks compatible with training via gradient backpropagation. This allows the capability for neural networks to learn connections with modulatory or shunting inhibition, which as evidenced in Chapter 4 is an important computation in biological nervous systems that is not present in modern neural networks. One area of future expansion is the application of this framework to other machine learning problems in order to investigate the potential benefit of these reversal potentials, particularly reinforcement learning for control problems.

As outlined in Chapter 6, further work is needed to train the insect-inspired network for rotation estimation in natural image sequences. Part of the difficulty in this task comes from the integration across multiple local motion detectors, as depth and parallax changes mean that the same local motion at a group of pixels can imply different rates of motion depending on the depth of that point in the visual scene. Focusing the training on learning the weighting of these individual motion detectors versus learning the entire network at once is a promising area of future expansion.

Additionally, while it has been shown that the insect nervous system clearly computes changes in global motion based on changing velocity stimuli, it is still unclear how much this response changes as the input stimulation pattern changes. Future work can look at how the information from this network can be combined with other sensory feedback such as leg proprioception within the insect nervous system for context-dependent decision making, as implemented in neural structures

such as the insect central complex.

Finally, SNSTorch has been primarily used for simulating large networks of non-spiking neurons and training them using gradient backpropagation. A short-term area of future work is incorporating the full suite of neural and synaptic dynamics from SNS-Toolbox into SNSTorch, although the capability of gradient backpropagation to optimize such dynamics as bidirectional electrical synapses and voltage-gated ion channels remains to be investigated. Once all of these dynamics are implemented, a full converter could be constructed which takes a network designed in SNS-Toolbox and migrates its populations to an SNSTorch representation. Additionally, SNSTorch's ease of expansion and compatibility with the rest of the PyTorch ecosystem suggests it could support local online learning methods such as short term synaptic plasticity. Integration and implementation of synaptic learning rules is a currently active field of research in neural computation, and could be an additional area of expansion for SNS-Toolbox and SNSTorch.

Appendix A

DATA AVAILABILITY

All of the work described in this document is freely open-source and available at the following domains:

- SNS-Toolbox (Software): https://github.com/wnourse05/SNS-Toolbox
- SNS-Toolbox (Documentation): https://sns-toolbox.readthedocs.io/en/latest/index.html
- Reduced model of visual processing: https://github.com/wnourse05/Motion-Vision-SNS
- SNSTorch: https://github.com/wnourse05/SNSTorch
- FlyWheel (Robot Information): https://github.com/wnourse05/FlyWheel
- Rotational Dataset: https://github.com/wnourse05/flywheel-rotation-dataset
- Initial Rotation Baseline: https://github.com/wnourse05/FlyWheelBaseline-LivingMachines2024
- SNSTorch Visual Motion Processing: https://github.com/wnourse05/SNS-for-Visual-Motion-Processing

Appendix B

PROJECTS USING SNS-TOOLBOX

While I have used SNS-Toolbox in my own work, it has also been disseminated and used by others for both research and coursework projects. A (non-exhaustive) list of these projects is given below, with citations if the work has been published.

- Control of peristaltic locomotion in a simulated compliant modular mesh robot [101, 102]
- Modeling and simulation of the capture response in a venus flytrap
- Modeling and optimization of motor microcircuits for vertebrate muscle control using Markov-Chain Monte Carlo sampling [67]
- Modeling of a sequence generation population for control of muscle coordination in quadruped locomotion
- Control of locomotion in a set of simulated feline hindlimbs based on multilayer central pattern generator networks
- Control of a simulated leg for a fruit-fly inspired robot
- Control of sideways locomotion in a simulated robotic crab

BIBLIOGRAPHY

[1] Jan M. Ache, Jason Polsky, Shada Alghailani, Ruchi Parekh, Patrick Breads, Martin Y. Peek, Davi D. Bock, Catherine R. von Reyn, and Gwyneth M. Card. "Neural Basis for Looming Size and Velocity Encoding in the *Drosophila* Giant Fiber Escape Pathway". In: *Current Biology* 29 (6 Mar. 2019), 1073–1081.e4. ISSN: 09609822. DOI: 10.1016/j.cub.2019.01.079.

[2] Denis Alevi, Marcel Stimberg, Henning Sprekeler, Klaus Obermayer, and Moritz Augustin. "Brian2CUDA: Flexible and Efficient Simulation of Spiking Neural Network Models on GPUs". In: *Frontiers in Neuroinformatics* 16 (Oct. 2022). ISSN: 1662-5196. DOI: 10.3389/fninf.2022.883700. URL: https://www.frontiersin.org/articles/10.3389/fninf.2022.883700/full.

[3] Jérémie Allard, Stéphane Cotin, François Faure, Pierre-Jean Bensoussan, François Poyer, Christian Duriez, Hervé Delingette, Laurent Grisoni, J Allard, S Cotin, F Faure, P.-J Bensoussan, F Poyer, C Duriez, H Delingette, and L Grisoni. "SOFA-an Open Source Framework for Medical Simulation". In: (2007). URL: https://hal.inria.fr/inria-00319416.

[4] Alexander Arenz, Michael S. Drews, Florian G. Richter, Georg Ammer, and Alexander Borst. "The Temporal Tuning of the Drosophila Motion Detectors Is Determined by the Dynamics of Their Input Elements". In: *Current Biology* 27 (7 Apr. 2017), pp. 929–944. ISSN: 09609822. DOI: 10.1016/j.cub.2017.01.051.

[5] Joseph Ayers and Jan Witting. "Biomimetic approaches to the control of underwater walking machines". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365 (1850 2007), pp. 273–295. ISSN: 1364503X. DOI: 10.1098/rsta.2006.1910.

[6] Joseph Ayers and Jan Witting. "Biomimetic approaches to the control of underwater walking machines". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365.1850 (2007), pp. 273–295.

[7] Zahra M. Bagheri, Steven D. Wiederman, Benjamin S. Cazzolato, Steven Grainger, and David C. O'Carroll. "Performance of an insect-inspired target tracker in natural conditions". In: *Bioinspiration and Biomimetics* 12 (2 Feb. 2017). ISSN: 17483190. DOI: 10.1088/1748-3190/aa5b48.

[8] H B Barlow and William R Levick. "The mechanism of directionally selective units in rabbit's retina." In: *The Journal of physiology* 178 (3 1965), p. 477.

[9] Chiara Bartolozzi, Giacomo Indiveri, and Elisa Donati. "Embodied neuromorphic intelligence". In: *Nature Communications* 13 (1 Dec. 2022). ISSN: 20411723. DOI: 10.1038/s41467-022-28487-2.

[10] Randall D Beer and John C Gallagher. "Evolving Dynamical Neural Networks for Adaptive Behavior". In: *Adaptive Behavior* 1 (1992), pp. 91–122.

[11] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C. Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. "Nengo: A Python tool for building large-scale functional brain models". In: *Frontiers in Neuroinformatics* 7 (JAN Jan. 2014). ISSN: 16625196.

[12] Alexander Borst. *Drosophila's View on Insect Vision*. Jan. 2009. DOI: 10.1016/j.cub.2008.11.001.

[13] Alexander Borst, Michael Drews, and Matthias Meier. *The neural network behind the eyes of a fly*. Aug. 2020. DOI: 10.1016/j.cophys.2020.05.004.

[14] Alexander Borst, Jürgen Haag, and Alex S. Mauss. *How fly neurons compute the direction of visual motion*. Mar. 2020. DOI: 10.1007/s00359-019-01375-9.

[15] James M Bower and David Beeman. *The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System*. Springer Science & Business Media, 2012.

[16] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.

[17] Amalia Braun, Alexander Borst, and Matthias Meier. "Disynaptic inhibition shapes tuning of OFF-motion detectors in Drosophila". In: *Current Biology* 33 (11 June 2023), 2260–2269.e4. ISSN: 18790445. DOI: 10.1016/j.cub.2023.05.007.

[18] Richard P Brent. *Algorithms for minimization without derivatives*. Courier Corporation, 2013.

[19] Thomas Graham Brown. "The intrinsic factors in the act of progression in the mammal". In: *Proceedings of the Royal Society of London. Series B, containing papers of a biological character* 84 (572 1911), pp. 308–319.

[20] Ross Cagan. "Chapter 5 Principles of Drosophila Eye Differentiation". In: *Current Topics in Developmental Biology* 89 (Jan. 2009), pp. 115–135. ISSN: 0070-2153. DOI: 10.1016/S0070-2153(09)89005-4.

[21] Benjamin P. Campbell, Huai Ti Lin, and Holger G. Krapp. "Weighting Elementary Movement Detectors Tuned to Different Temporal Frequencies to Estimate Image Velocity". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 14157 LNAI (2023), pp. 398–410. ISSN: 16113349. DOI: 10.1007/978-3-031-38857-6_29/FIGURES/6.

[22] Marie Claire Capolei, Emmanouil Angelidis, Egidio Falotico, Henrik Hautop Lund, and Silvia Tolu. "A biomimetic control method increases the adaptability of a humanoid robot acting in a dynamic environment". In: *Frontiers in Neurorobotics* 13 (2019). ISSN: 16625218. DOI: 10.3389/fnbot.2019.00070.

[23] Suma G Cardwell and Frances S Chance. "Dendritic computation for neuromorphic applications". In: *Proceedings of the 2023 International Conference on Neuromorphic Systems*. 2023, pp. 1–5.

[24] Hillel J. Chiel and Randall D. Beer. "The brain has a body: adaptive behavior emerges from interactions of nervous system, body and environment". In: *Trends in Neurosciences* 20 (12 Dec. 1997), pp. 553–557. ISSN: 01662236. DOI: 10.1016/S0166-2236(97)01149-1.

[25] Thomas R Clandinin, Chi-Hon Lee, Tory Herman, Roger C Lee, Annie Y Yang, Shake Ovasapyan, and S Lawrence Zipursky. *Drosophila LAR Regulates R1-R6 and R7 Target Specificity in the Visual System*. 2001.

[26] Damon A. Clark and Jonathan B. Demb. *Parallel Computations in Insect and Mammalian Visual Motion Processing*. Oct. 2016. DOI: 10.1016/j.cub.2016.08.003.

[27] David Cofer, Gennady Cymbalyuk, William J. Heitler, and Donald H. Edwards. "Control of tumbling during the locust jump". In: *Journal of Experimental Biology* 213 (19 Oct. 2010), pp. 3378–3387. ISSN: 00220949. DOI: 10.1242/jeb.046367.

[28] David Cofer, Gennady Cymbalyuk, James Reid, Ying Zhu, William J. Heitler, and Donald H. Edwards. "AnimatLab: A 3D graphics environment for neuromechanical simulations". In: *Journal of Neuroscience Methods* 187 (2 Mar. 2010), pp. 280–288. ISSN: 01650270.

[29] Gregory Cohen. “Gooaall!!!: Why we Built a Neuromorphic Robot to Play Foosball”. In: *IEEE Spectrum* 59 (3 Mar. 2022), pp. 44–50. ISSN: 0018-9235.

[30] Gabriele Costante, Michele Mancini, Paolo Valigi, and Thomas A. Ciarfuglia. “Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation”. In: *IEEE Robotics and Automation Letters* 1 (1 Jan. 2016), pp. 18–25. ISSN: 23773766. DOI: 10.1109/LRA.2015.2505717.

[31] Matthew S. Creamer, Omer Mano, and Damon A. Clark. “Visual Control of Walking Speed in *Drosophila*”. In: *Neuron* 100 (6 Dec. 2018), 1460–1473.e6. ISSN: 10974199. DOI: 10.1016/j.neuron.2018.10.028.

[32] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[33] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. “Loihi: A neuromorphic manycore processor with on-chip learning”. In: *IEEE Micro* 38 (1 2018), pp. 82–99.

[34] Scott L. Delp, Frank C. Anderson, Allison S. Arnold, Peter Loan, Ayman Habib, Chand T. John, Eran Guendelman, and Darryl G. Thelen. “OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement”. In: *IEEE Transactions on Biomedical Engineering* 54 (11 Nov. 2007), pp. 1940–1950. ISSN: 0018-9294.

[35] Kaiyu Deng, Nicholas S. Szczecinski, Dirk Arnold, Emanuel Andrada, Martin Fischer, Roger D. Quinn, and Alexander J. Hunt. “Neuromechanical model of rat hind limb walking with two layer CPGs and muscle synergies”. In: vol. 10928 LNAI. Springer Verlag, 2018, pp. 134–144. ISBN: 9783319959719.

[36] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[37] Sophie Denève and Christian K. Machens. *Efficient codes and balanced networks*. Feb. 2016. DOI: 10.1038/nn.4243.

[38] Travis DeWolf, Kinjal Patel, Pawel Jaworski, Roxana Leontie, Joe Hays, and Chris Eliasmith. "Neuromorphic control of a simulated 7-DOF arm using Loihi". In: *Neuromorphic Computing and Engineering* 3 (1 Mar. 2023), p. 014007. DOI: 10.1088/2634-4386/acb286.

[39] Mikael Djurfeldt, Johannes Hjorth, Jochen M. Eppler, Niraj Dudani, Moritz Helias, Tobias C. Potjans, Upinder S. Bhalla, Markus Diesmann, Jeanette Hellgren Kotaleski, and Örjan Ekeberg. "Run-time interoperability between neuronal network simulators based on the MUSIC framework". In: *Neuroinformatics* 8 (1 Mar. 2010), pp. 43–60. ISSN: 15392791. DOI: 10.1007/s12021-010-9064-z.

[40] Michael S. Drews, Aljoscha Leonhardt, Nadezhda Pirogova, Florian G. Richter, Anna Schuetzenberger, Lukas Braun, Etienne Serbe, and Alexander Borst. "Dynamic Signal Compression for Robust Motion Vision in Flies". In: *Current Biology* 30 (2 Jan. 2020), 209–221.e8. ISSN: 09609822. DOI: 10.1016/j.cub.2019.10.035.

[41] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.

[42] Jason K. Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D. Lu. "Training Spiking Neural Networks Using Lessons From Deep Learning". In: (Sept. 2021).

[43] 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 Mactinaz-Cañada, Eduardo Ros, Patrick Maier, Sandro Weber, Manuei Huber, David Plecher, Florian Röhrbein, Stefan Deser, Alina Roitberg, Patrick Van Der Smagt, Rüdiger Dillman, Paul Levi, Cecilia Laschi, Alois C. Knoll, and Marc Oliver Gewaltig. "Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform". In: *Frontiers in Neurorobotics* 11 (JAN Jan. 2017). ISSN: 16625218.

[44] Lisa M. Fenk, Sofia C. Avritzer, Jazz L. Weisman, Aditya Nair, Lucas D. Randt, Thomas L. Mohren, Igor Siwanowicz, and Gaby Maimon. "Muscles that move the retina augment compound eye vision in Drosophila". In: *Nature 2022 612:7938* 612 (7938 Oct. 2022), pp. 116–122. ISSN: 1476-4687. DOI: 10.1038/s41586-022-05317-5.

[45] Andreas K. Fidjeland, Etienne B. Roesch, Murray P. Shanahan, and Wayne Luk. "NeMo: A platform for neural modelling of spiking neurons using GPUs". In: 2009, pp. 137–144. ISBN: 9780769537320. DOI: 10.1109/ASAP.2009.24.

[46] David Fitzpatrick. "The Functional Organization of Local Circuits in Visual Cortex: Insights from the Study of Tree Shrew Striate Cortex". In: *Cerebral Cortex* 6 (3 1996), pp. 329–341. ISSN: 1047-3211. DOI: 10.1093/cercor/6.3.329.

[47] Limor Freifeld, Damon A. Clark, Mark J. Schnitzer, Mark A. Horowitz, and Thomas R. Clandinin. "GABAergic Lateral Interactions Tune the Early Stages of Visual Processing in *Drosophila*". In: *Neuron* 78 (6 June 2013), pp. 1075–1089. ISSN: 08966273. DOI: 10.1016/j.neuron.2013.04.024.

[48] Terufumi Fujiwara, Margarida Brotas, and M. Eugenia Chiappe. "Walking strides direct rapid and flexible recruitment of visual circuits for course control in *Drosophila*". In: *Neuron* 110 (13 July 2022), 2124–2138.e8. ISSN: 10974199. DOI: 10.1016/j.neuron.2022.04.008.

[49] Marc-Oliver Gewaltig and Markus Diesmann. "Nest (neural simulation tool)". In: *Scholarpedia* 2 (4 2007), p. 1430.

[50] C A Goldsmith, N S Szczecinski, and R D Quinn. "Neurodynamic modeling of the fruit fly *Drosophila melanogaster*". In: *Bioinspiration & Biomimetics* 15 (6 Sept. 2020), p. 065003. ISSN: 1748-3190.

[51] Dan Goodman and Romain Brette. "Brian: A simulator for spiking neural networks in python". In: *Frontiers in Neuroinformatics* 2 (NOV Nov. 2008). ISSN: 16625196.

[52] Lukas N. Groschner, Jonatan G. Malis, Birte Zuidinga, and Alexander Borst. "A biophysical account of multiplication by a single neuron". In: *Nature* 603 (7899 Mar. 2022), pp. 119–123. ISSN: 14764687. DOI: 10.1038/s41586-022-04428-3.

[53] Chloe K. Guie and Nicholas S. Szczecinski. "Direct Assembly and Tuning of Dynamical Neural Networks for Kinematics". In: vol. 13548 LNAI. Springer Science and Business Media Deutschland GmbH, 2022, pp. 321–331. ISBN: 9783031204692. DOI: 10.1007/978-3-031-20470-8_32.

[54] Juergen Haag, Alexander Arenz, Etienne Serbe, Fabrizio Gabbiani, and Alexander Borst. "Complementary mechanisms create direction selectivity in the fly". In: *eLife* 5 (AUGUST Aug. 2016). ISSN: 2050084X. DOI: 10.7554/eLife.17421.001.

[55] Juergen Haag, Abhishek Mishra, and Alexander Borst. "A common directional tuning mechanism of *Drosophila* motion-sensing neurons in the ON

and in the OFF pathway". In: (2017). DOI: 10.7554/eLife.29044.001. URL: https://doi.org/10.7554/eLife.29044.001.

[56] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". In: *Nature* 585 (7825 Sept. 2020), pp. 357–362. ISSN: 14764687.

[57] Bernhard Hassenstein and Werner Reichardt. "Systemtheoretische analyse der zeit-, reihenfolgen-und vorzeichenauswertung bei der bewegungsperzeption des rüsselkäfers chlorophanus". In: *Zeitschrift für Naturforschung B* 11 (9-10 1956), pp. 513–524.

[58] Hananel Hazan, Daniel J. Saunders, Hassaan Khan, Devdhar Patel, Darpan T. Sanghavi, Hava T. Siegelmann, and Robert Kozma. "BindsNET: A machine learning-oriented spiking neural networks library in python". In: *Frontiers in Neuroinformatics* 12 (Dec. 2018). ISSN: 16625196.

[59] M L Hines and N T Carnevale. "NEURON: A Tool for Neuroscientists". In: *THE NEUROSCIENTIST* 7 (2 2001). URL: http://www.neu.

[60] Roger V. Hoang, Devyani Tanna, Laurence C. Jayet Bray, Sergiu M. Dascalu, and Frederick C. Harris. "A novel CPU/GPU simulation environment for large-scale biologically realistic neural modeling". In: *Frontiers in Neuroinformatics* 7 (OCT Oct. 2013). ISSN: 16625196. DOI: 10.3389/fninf.2013.00019.

[61] A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The*

Journal of Physiology 117 (4 Aug. 1952), pp. 500–544. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1952.sp004764.

[62] Berthold K.P. Horn and E. J. Weldon. "Direct methods for recovering motion". In: *International Journal of Computer Vision* 2 (1 June 1988), pp. 51–76. ISSN: 09205691. DOI: 10.1007/BF00836281/METRICS.

[63] H Hultborn, S Lindstr6m, and H Wigstr6m. "On the Function of Recurrent Inhibition in the Spinal Cord". In: *Brain Res* 37 (1979), p. 399403.

[64] Alexander Hunt, Nicholas Szczecinski, and Roger Quinn. "Development and training of a neural controller for hind leg walking in a dog robot". In: *Frontiers in Neurorobotics* 11 (APR Apr. 2017). ISSN: 16625218.

[65] Fadi A. Issa, Joanne Drummond, Daniel Cattaert, and Donald H. Edwards. "Neural circuit reconfiguration by social status". In: *Journal of Neuroscience* 32 (16 Apr. 2012), pp. 5638–5645. ISSN: 02706474. DOI: 10.1523/JNEUROSCI.5668-11.2012.

[66] Eugene M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Burtsing*. 2007, p. 441. ISBN: 9780262090438. DOI: 10.1017/S0143385704000173.

[67] Clayton Jackson, Matthieu Chardon, Y Curtis Wang, Johann Rudi, Matthew Tresch, Charles J Heckman, and Roger D Quinn. "Multimodal Parameter Inference for a Canonical Motor Microcircuit Controlling Rat Hindlimb Motion". In: *Conference on Biomimetic and Biohybrid Systems*. Springer. 2023, pp. 38–51.

[68] Clayton Jackson, William R.P. Nourse, C. J. Heckman, Matthew Tresch, and Roger D. Quinn. "Canonical Motor Microcircuit for Control of a Rat Hindlimb". In: vol. 13548 LNAI. Springer Science and Business Media Deutschland GmbH, 2022, pp. 309–320. ISBN: 9783031204692. DOI: 10.1007/978-3-031-20470-8_31.

[69] Will L. Johnson, Devin L. Jindrich, Roland R. Roy, and V. Reggie Edgerton. "A three-dimensional model of the rat hindlimb: Musculoskeletal geometry and muscle moment arms". In: *Journal of Biomechanics* 41 (3 2008), pp. 610–619. ISSN: 00219290. DOI: 10.1016/j.jbiomech.2007.10.004.

[70] Jacques Kaiser, J Camilo Vasquez Tieck, Christian Hubschneider, Peter Wolf, Michael Weber, Michael Hoff, Alexander Friedrich, Konrad Wojtasik, Arne Roennau, Ralf Kohlhaas, et al. "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)*. IEEE. 2016, pp. 127–134.

[71] Naila Habib Khan and Awais Adnan. "Ego-motion estimation concepts, algorithms and challenges: an overview". In: *Multimedia Tools and Applications* 76 (15 Aug. 2017), pp. 16581–16603. ISSN: 15737721. DOI: 10.1007/S11042-016-3939-4/TABLES/2.

[72] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[73] Nathan Koenig and Andrew Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: vol. 3. 2004, pp. 2149–2154. ISBN: 0780384636. DOI: 10.1109/iros.2004.1389727.

[74] Jessica R Kohn, Jacob P Portes, Matthias P Christenson, LF Abbott, and Rudy Behnia. "Flexible filtering by neural inputs supports motion computation across states and stimuli". In: *Current Biology* 31.23 (2021), pp. 5249–5260.

[75] Justin P. Kumar. *Building an ommatidium one cell at a time*. Jan. 2012.

[76] Janne K Lappalainen, Fabian D Tschopp, Sridhama Prakhya, Mason McGill, Aljoscha Nern, Kazunori Shinomiya, Shin-ya Takemura, Eyal Gruntman, Jakob H Macke, and Srinivas C Turaga. "Connectome-constrained deep

mechanistic networks predict neural responses across the fly visual system at single-neuron resolution". In: *bioRxiv* (2023), pp. 2023–03.

[77] "Lava Software Framework". In: (2021). URL: https://github.com/lava-nc/lava.

[78] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86 (11 1998), pp. 2278–2323. ISSN: 00189219. DOI: 10.1109/5.726791.

[79] G Leuba and R Kraftsik. *Anatomy and Embryolo Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age*. 1994.

[80] Genevieve Leuba and Rudolf Kraftsik. "Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age". In: *Anatomy and embryology* 190 (1994), pp. 351–366.

[81] Wolfgang Maass. "Networks of Spiking Neurons: The Third Generation of Neural Network Models". In: *Neural Networks* 10 (9 1997), pp. 1659–1671.

[82] Matthew S Maisak, Juergen Haag, Georg Ammer, Etienne Serbe, Matthias Meier, Aljoscha Leonhardt, Tabea Schilling, Armin Bahl, Gerald M Rubin, Aljoscha Nern, et al. "A directional tuning map of Drosophila elementary motion detectors". In: *Nature* 500.7461 (2013), pp. 212–216.

[83] Michael Mangan, Dario Floreano, Kotaro Yasui, Barry A Trimmer, Nick Gravish, Sabine Hauert, Barbara Webb, Poramate Manoonpong, and Nicholas Stephen Szczecinski. "A virtuous cycle between invertebrate and robotics research: Perspective on a decade of Living Machines research". In: *Bioinspiration & Biomimetics* (May 2023). ISSN: 1748-3182. DOI: 10.1088/1748-3190/acc223.

[84] Elisa Massi, Lorenzo Vannucci, Ugo Albanese, Marie Claire Capolei, Alexander Vandesompele, Gabriel Urbain, Angelo Maria Sabatini, Joni Dambre, Cecilia Laschi, Silvia Tolu, and Egidio Falotico. "Combining evolutionary and adaptive control strategies for quadruped robotic locomotion". In: *Frontiers in Neurorobotics* 13 (2019), pp. 1–19. ISSN: 16625218. DOI: 10.3389/fnbot.2019.00071.

[85] Alex S. Mauss and Alexander Borst. *Motion Vision in Arthropods*. Apr. 2019. DOI: 10.1093/oxfordhb/9780190456757.013.14.

[86] Matthias Meier and Alexander Borst. "Extreme Compartmentalization in a *Drosophila* Amacrine Cell". In: *Current Biology* 29 (9 May 2019), 1545–1550.e2. ISSN: 09609822. DOI: 10.1016/j.cub.2019.03.070.

[87] R. C. MIALL. "The flicker fusion frequencies of six laboratory insects, and the response of the compound eye to mains fluorescent 'ripple'". In: *Physiological Entomology* 3 (2 June 1978), pp. 99–106. ISSN: 0307-6962. DOI: 10.1111/j.1365-3032.1978.tb00139.x.

[88] Ştefan Mihalaş and Ernst Niebur. "A generalized linear integrate-and-fire neural model produces diverse spiking behaviors". In: *Neural Computation* 21 (3 Mar. 2009), pp. 704–718. ISSN: 08997667.

[89] Milad Mozafari, Mohammad Ganjtabesh, Abbas Nowzari-Dalini, and Timothée Masquelier. "SpykeTorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron". In: *Frontiers in Neuroscience* 13 (JUL 2019). ISSN: 1662453X.

[90] Jim Mutch. "CNS: a GPU-based framework for simulating cortically-organized networks". In: *CNS: a GPU-based framework for simulating cortically-organized networks* (2010).

[91] Lars Niedermeier, Kexin Chen, Jinwei Xing, Anup Das, Jeffrey Kopsick, Eric Scott, Nate Sutton, Killian Weber, Nikil Dutt, and Jeffrey L. Krichmar. "CARLsim 6: An Open Source Library for Large-Scale, Biologically Detailed Spiking Neural Network Simulation". In: vol. 2022-July. Institute of Electrical and Electronics Engineers Inc., 2022. ISBN: 9781728186719. DOI: 10.1109/IJCNN55064.2022.9892644.

[92] William RP Nourse, Clayton Jackson, Nicholas S Szczecinski, and Roger D Quinn. "SNS-Toolbox: An Open Source Tool for Designing Synthetic Nervous Systems and Interfacing Them with Cyber–Physical Systems". In: *Biomimetics* 8.2 (2023), p. 247.

[93] William R.P. Nourse, Nicholas S. Szczecinski, and Roger D. Quinn. "A Synthetic Nervous System for on and Off Motion Detection Inspired by the Drosophila melanogaster Optic Lobe". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 14157 LNAI (2023), pp. 364–380. ISSN: 16113349. DOI: 10.1007/978-3-031-38857-6_27/TABLES/2.

[94] William R.P. Nourse, Nicholas S. Szczecinski, and Roger D. Quinn. "SNS-Toolbox: A Tool for Efficient Simulation of Synthetic Nervous Systems". In: vol. 13548 LNAI. Springer Science and Business Media Deutschland GmbH, 2022, pp. 32–43. ISBN: 9783031204692. DOI: 10.1007/978-3-031-20470-8_4.

[95] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury Google, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf Xamla, Edward Yang, Zach Devito, Martin Raison Nabla, Alykhan Tejani, Sasank Chilamkurthy, Qure Ai, Benoit Steiner, Lu Fang Facebook, Junjie Bai Facebook, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning

Library". In: ed. by H Wallach, H Larochelle, A Beygelzimer, F d'alché Buc, E Fox, and R Garnett. Curran Associates, Inc., 2019.

[96] Donald H. Perkel and Brian Mulloney. "Motor Pattern Production in Reciprocally Inhibitory Neurons Exhibiting Postinhibitory Rebound". In: *Science* 185 (4146 July 1974), pp. 181–183. ISSN: 0036-8075. DOI: 10.1126/science.185.4146.181.

[97] Claire Plunkett and Frances Chance. "Modeling Coordinate Transformations in the Dragonfly Nervous System". In: *Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference*. 2023, pp. 6–10.

[98] K. Prazdny. "Egomotion and relative depth map from optical flow". In: *Biological Cybernetics* 36 (2 Feb. 1980), pp. 87–102. ISSN: 03401200. DOI: 10.1007/BF00361077/METRICS.

[99] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. "ROS: an open-source Robot Operating System". In: (2009). URL: http://stair.stanford.edu.

[100] Magnus J.E. Richardson. "Effects of synaptic conductance on the voltage distribution and firing rate of spiking neurons". In: *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* 69 (5 2004), p. 8. ISSN: 1063651X. DOI: 10.1103/PhysRevE.69.051918.

[101] Shane Riddle, Clayton Jackson, Kathryn A Daltorio, and Roger D Quinn. "A Dynamic Simulation of a Compliant Worm Robot Amenable to Neural Control". In: *Conference on Biomimetic and Biohybrid Systems*. Springer. 2023, pp. 338–352.

[102] Shane Riddle, William RP Nourse, Zhuojun Yu, Peter J Thomas, and Roger D Quinn. "A synthetic nervous system with coupled oscillators controls peristaltic locomotion". In: *Conference on Biomimetic and Biohybrid Systems*. Springer. 2022, pp. 249–261.

[103] J. H. Rieger and D. T. Lawton. "Processing differential image motion". In: *JOSA A, Vol. 2, Issue 2, pp. 354-359* 2 (2 Feb. 1985), pp. 354–359. ISSN: 1520-8532. DOI: 10.1364/JOSAA.2.000354.

[104] Jens Rister, Claude Desplan, and Daniel Vasiliauskas. "Establishing and maintaining gene expression patterns: insights from sensory receptor patterning". In: *Development* 140 (3 Feb. 2013), pp. 493–503. ISSN: 0950-1991. DOI: 10.1242/DEV.079095.

[105] Guido Van Rossum et al. "Python Programming Language." In: vol. 41. 2007, pp. 1–36.

[106] Ilya A. Rybak, Natalia A. Shevtsova, Myriam Lafreniere-Roula, and David A. McCrea. "Modelling spinal circuitry involved in locomotor pattern generation: Insights from deletions during fictive locomotion". In: *Journal of Physiology* 577 (2 Dec. 2006), pp. 617–639. ISSN: 00223751. DOI: 10.1113/jphysiol.2006.118703.

[107] Louis K Scheffer, C Shan Xu, Michal Januszewski, Zhiyuan Lu, Shin-ya Takemura, Kenneth J Hayworth, Gary B Huang, Kazunori Shinomiya, Jeremy Maitlin-Shepard, Stuart Berg, et al. "A connectome and analysis of the adult Drosophila central brain". In: *Elife* 9 (2020), e57443.

[108] Louis K. Scheffer et al. "A connectome and analysis of the adult drosophila central brain". In: *eLife* 9 (Sept. 2020), pp. 1–74. ISSN: 2050084X. DOI: 10.7554/ELIFE.57443.

[109] Malte Schilling and Holk Cruse. "neuroWalknet, a controller for hexapod walking allowing for context dependent behavior". In: *PLoS Computational Biology* 19 (1 Jan. 2023). ISSN: 15537358. DOI: 10.1371/journal.pcbi.1010136.

[110] Anna Sedlackova, Nicholas S Szczecinski, and Roger D Quinn. "A synthetic nervous system model of the insect optomotor response". In: *Biomimetic and Biohybrid Systems: 9th International Conference, Living Machines 2020, Freiburg, Germany, July 28–30, 2020, Proceedings 9*. Springer. 2020, pp. 312–324.

[111] Johannes D Seelig and Vivek Jayaraman. "Neural dynamics for landmark orientation and angular path integration". In: *Nature* 521.7551 (2015), pp. 186–191.

[112] Etienne Serbe, Matthias Meier, Aljoscha Leonhardt, and Alexander Borst. "Comprehensive Characterization of the Major Presynaptic Elements to the Drosophila OFF Motion Detector". In: *Neuron* 89 (4 Feb. 2016), pp. 829–841. ISSN: 10974199. DOI: 10.1016/j.neuron.2016.01.006.

[113] H. Sebastian Seung, Daniel D. Lee, Ben Y. Reis, and David W. Tank. "The Autapse: A Simple Illustration of Short-Term Analog Memory Storage by Tuned Synaptic Feedback". In: *Journal of Computational Neuroscience* 9 (2 2000), pp. 171–185. ISSN: 09295313. DOI: 10.1023/A:1008971908649.

[114] Camilla R. Sharkey, Jorge Blanco, Maya M. Leibowitz, Daniel Pinto-Benito, and Trevor J. Wardill. "The spectral sensitivity of Drosophila photoreceptors". In: *Scientific Reports 2020 10:1* 10 (1 Oct. 2020), pp. 1–13. ISSN: 2045-2322. DOI: 10.1038/s41598-020-74742-1.

[115] Kazunori Shinomiya, Gary Huang, Zhiyuan Lu, Toufiq Parag, C. Shan Xu, Roxanne Aniceto, Namra Ansari, Natasha Cheatham, Shirley Lauchie, Erika Neace, Omotara Ogundeyi, Christopher Ordish, David Peel, Aya Shinomiya, Claire Smith, Satoko Takemura, Iris Talebi, Patricia K. Rivlin, Aljoscha Nern, Louis K. Scheffer, Stephen M. Plaza, and Ian A. Meinertzhagen. "Comparisons between the ON-and OFF-edge motion pathways

in the Drosophila brain". In: *eLife* 8 (2019). ISSN: 2050084X. DOI: 10.7554/ELIFE.40025.

[116] D G Stavenga. *Insect Retinal Pigments: Spectral Characteristics and Physiological Functions*. 1995.

[117] Tom Hindmarsh Sten, Rufei Li, Adriane Otopalik, and Vanessa Ruta. "Sexual arousal gates visual processing during Drosophila courtship". In: *Nature 2021 595:7868* 595 (7868 July 2021), pp. 549–553. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03714-w.

[118] Beck Strohmer, Poramate Manoonpong, and Leon Bonde Larsen. "Flexible Spiking CPGs for Online Manipulation During Hexapod Walking". In: *Frontiers in Neurorobotics* 14 (June 2020). ISSN: 16625218.

[119] James A. Strother, Shiuan Tze Wu, Edward M. Rogers, Jessica L.M. Eliason, Allan M. Wong, Aljoscha Nern, and Michael B. Reiser. "Behavioral state modulates the on visual motion pathway of drosophila". In: *Proceedings of the National Academy of Sciences of the United States of America* 115 (1 Jan. 2018), E102–E111. ISSN: 10916490. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1703090115.

[120] Nicholas S. Szczecinski, Amy E. Brown, John A. Bender, Roger D. Quinn, and Roy E. Ritzmann. "A neuromechanical simulation of insect walking and transition to turning of the cockroach Blaberus discoidalis". In: *Biological Cybernetics* 108 (1 Feb. 2014), pp. 1–21. ISSN: 03401200.

[121] Nicholas S Szczecinski, Alexander J Hunt, and Roger D Quinn. "A functional subnetwork approach to designing synthetic nervous systems that control legged robot locomotion". In: *Frontiers in neurorobotics* 11 (2017), p. 37.

[122] Nicholas S. Szczecinski, Alexander J. Hunt, and Roger D. Quinn. "Design process and tools for dynamic neuromechanical models and robot controllers". In: *Biological Cybernetics* 111 (1 Feb. 2017), pp. 105–127. ISSN: 0340-1200. URL: http://link.springer.com/10.1007/s00422-017-0711-4.

[123] Nicholas S. Szczecinski, Roger D. Quinn, and Alexander J. Hunt. "Extending the Functional Subnetwork Approach to a Generalized Linear Integrate-and-Fire Neuron Model". In: *Frontiers in Neurorobotics* 14 (Nov. 2020). ISSN: 16625218. DOI: 10.3389/fnbot.2020.577804.

[124] Nicholas Stephen Szczecinski, Clarissa Goldsmith, William Nourse, and Roger D Quinn. "A perspective on the neuromorphic control of legged locomotion in past, present, and future insect-like robots". In: *Neuromorphic Computing and Engineering* (Mar. 2023). DOI: 10.1088/2634-4386/acc04f.

[125] Shin-Ya Takemura, Aljoscha Nern, Dmitri B Chklovskii, Louis K Scheffer, Gerald M Rubin, and Ian A Meinertzhagen. "The comprehensive connectome of a neural substrate for 'ON' motion detection in Drosophila". In: (2017). DOI: 10.7554/eLife.24394.001.

[126] Corinne Teeter, Ramakrishnan Iyer, Vilas Menon, Nathan Gouwens, David Feng, Jim Berg, Aaron Szafer, Nicholas Cain, Hongkui Zeng, Michael Hawrylycz, Christof Koch, and Stefan Mihalas. "Generalized leaky integrate-and-fire models classify multiple neuron types". In: *Nature Communications* 9 (1 Dec. 2018). ISSN: 20411723. DOI: 10.1038/s41467-017-02717-4.

[127] Emanuel Todorov, Tom Erez, and Yuval Tassa. "MuJoCo: A physics engine for model-based control". In: 2012, pp. 5026–5033. ISBN: 9781467317375. DOI: 10.1109/IROS.2012.6386109.

[128] Carlo Tomasi and Jianbo Shi. "Direction of heading from image deformations". In: *IEEE Computer Vision and Pattern Recognition* (1993), pp. 422–427. DOI: 10.1109/CVPR.1993.341096.

[129] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[130] Julien Vitay, Helge Dinkelbach, and Fred H. Hamker. "ANNarchy: A code generation approach to neural simulations on parallel hardware". In: *Frontiers in Neuroinformatics* 9 (JULY July 2015). ISSN: 16625196. DOI: 10.3389/fninf.2015.00019.

[131] Thomas Voegtlin. "CLONES : a closed-loop simulation framework for body, muscles and neurons". In: *BMC Neuroscience* 12 (S1 Dec. 2011). DOI: 10.1186/1471-2202-12-s1-p363.

[132] Cong Wang, Zaizheng Yang, Shuang Wang, Pengfei Wang, Chen-Yu Wang, Chen Pan, Bin Cheng, Shi-Jun Liang, and Feng Miao. "A Braitenberg vehicle based on memristive neuromorphic circuits". In: *Advanced Intelligent Systems* 2.1 (2020), p. 1900103.

[133] Sibo Wang-Chen, Victor Alfred Stimpfling, Pembe Gizem¨ozdil, Gizem¨ Gizem¨ozdil, Louise Genoud, Femke Hurtak, and Pavan Ramdya. "NeuroMechFly 2.0, a framework for simulating embodied sensorimotor control in adult Drosophila". In: (). DOI: 10.1101/2023.09.18.556649.

[134] Barbara Webb. "Robots in invertebrate neuroscience". In: *Nature* 417 (6886 May 2002), pp. 359–363. ISSN: 0028-0836. DOI: 10.1038/417359a.

[135] Philipp Weidel, Mikael Djurfeldt, Renato C Duarte, and Abigail Morrison. "Closed loop interactions between spiking neural network and robotic simulators based on MUSIC and ROS". In: *Frontiers in neuroinformatics* 10 (2016), p. 31.

[136] Paul J. Werbos. "Bacpropagation Through Time: WHat It Does and How to Do It". In: *Proceedings of the IEEE* 78 (10 1990).

[137] Michael Winding, Benjamin D. Pedigo, Christopher L. Barnes, Heather G. Patsolic, Youngser Park, Tom Kazimiers, Akira Fushiki, Ingrid V. Andrade, Avinash Khandelwal, Javier Valdes-Aleman, Feng Li, Nadine Randel, Elizabeth Barsotti, Ana Correia, Richard D. Fetter, Volker Hartenstein, Carey E. Priebe, Joshua T. Vogelstein, Albert Cardona, and Marta Zlatic. "The connectome of an insect brain". In: *Science (New York, N.Y.)* 379 (6636 Mar. 2023), eadd9330. ISSN: 10959203. DOI: 10.1126/science.add9330.

[138] Esin Yavuz, James Turner, and Thomas Nowotny. "GeNN: A code generation framework for accelerated brain simulations". In: *Scientific Reports* 6 (Jan. 2016). ISSN: 20452322. DOI: 10.1038/srep18854.

[139] Fletcher Young. "Design and Analysis of a Biomechanical Model of the Rat Hindlimb with a Complete Musculature". In: (2022). URL: http://rave.ohiolink.edu/etdc/view?acc_num=case1648154057237043.

[140] Wenhao Yu, Deepali Jain, Alejandro Escontrela, Atil Iscen, Peng Xu, Erwin Coumans, Sehoon Ha, Jie Tan, and Tingnan Zhang. *Visual-Locomotion: Learning to Walk on Complex Terrains with Vision*.

[141] Yury V. Zaytsev and Abigail Morrison. "CyNEST: A maintainable cython-based interface for the NEST simulator". In: *Frontiers in Neuroinformatics* 8 (MAR Mar. 2014). ISSN: 16625196.

[142] Lei Zhang, Tianguang Zhang, Haiyan Wu, Alexander Borst, and Kolja Khnlenz. "Visual flight control of a quadrotor using bioinspired motion detector". In: *International Journal of Navigation and Observation* (2012). ISSN: 16875990. DOI: 10.1155/2012/627079.

[143] Baigan Zhao, Yingping Huang, Hongjian Wei, and Xing Hu. "Ego-Motion Estimation Using Recurrent Convolutional Neural Networks through Optical Flow Learning". In: *Electronics 2021, Vol. 10, Page 222* 10 (3 Jan. 2021), p. 222. ISSN: 2079-9292. DOI: 10.3390/ELECTRONICS10030222.

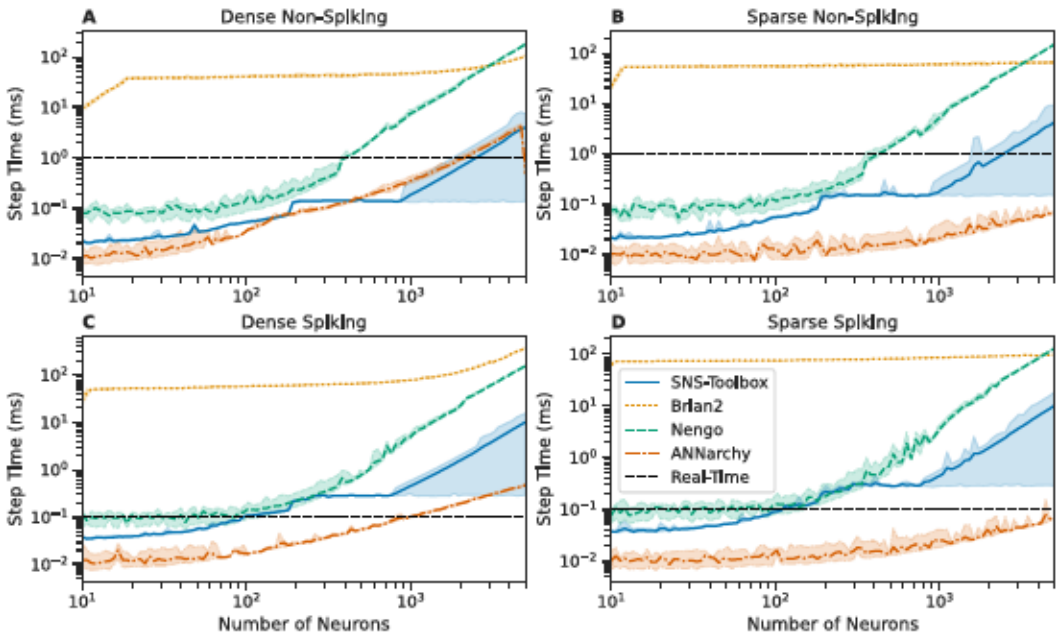


Figure 3.3: Comparison of wall-clock times for SNS-Toolbox to simulate a network for one simulation time-step over varying network sizes, using SNS-Toolbox and three other neural simulators (Brian2 [51], Nengo [11], and ANNarchy [130]). For the following simulators, the time data presented are chosen as the best-performing backend variant, Brian2, standard Brian2, and the GPU-accelerated Brian2CUDA; SNS-Toolbox, all available variants; and ANNarchy, CPU-based compilation, and GPU-based compilation. (**A**,**B**): Networks of non-spiking neurons, (**C**,**D**): networks of spiking neurons. **Left**: Fully-connected networks, **Right**: Sparsely connected networks, following the structure described in Section 3.4.2. Lines denote the mean over 1000 steps, shaded region denotes the area between the fifth and ninety-fifth percentiles. The real-time limit is denoted with a horizontal dashed black line.

Performance on Embedded Hardware

The testing procedure presented in Section 3.4.2 is again repeated, testing the performance of SNS-Toolbox on various embedded computing platforms. These included a Raspberry Pi Model 3B (trademark Raspberry Pi Limited, Cambridge, UK), Jetson Nano 4GB (trademark NVIDIA Corporation, Santa Clara, CA, USA), and an Intel NUC SWNUC11PHKi7c00 (trademark Intel Corporation, Santa Clara, CA, USA) with 32 GB of RAM. Due to the reduced available memory available on the Raspberry Pi and Jetson, network size is varied logarithmically from 10 to 1000

neurons, instead of the 10–5000 neurons in Sections 3.4.2 and 3.4.2. Results are shown in Figure 3.4; for clarity, all backends are condensed for each device such that the best performing solution at each network size is presented. The Raspberry Pi performs comparably with a Jetson Nano, with the Jetson exhibiting slightly better performance across all network sizes. The amount of memory available on the Raspberry Pi is the smallest of the three devices, so it is unable to simulate densely-connected networks over approximately 900 neurons in size. The Intel NUC is a significantly more powerful computing platform than the Raspberry Pi or the Jetson Nano, and accordingly behaves more closely to desktop-level performance.

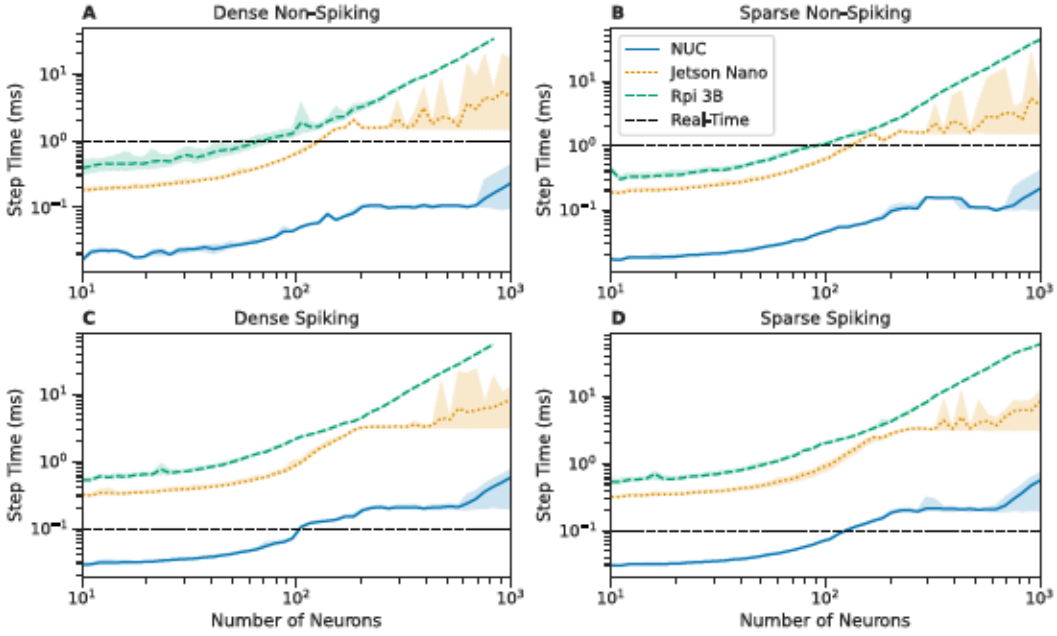
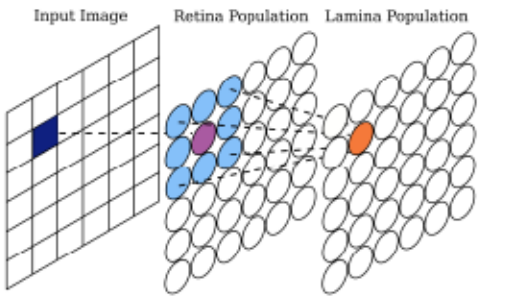


Figure 3.4: Comparison of wall-clock times to simulate a network for one simulation time-step over varying network sizes, using SNS-Toolbox on three different embedded computing platforms (Intel NUC, Raspberry Pi version 3b, and an NVIDIA Jetson Nano). The time data presented are chosen as the best-performing backend variant at each network size, with GPU-based backends excluded on the Raspberry Pi. (**A**,**B**): Networks of non-spiking neurons, (**C**,**D**): networks of spiking neurons. **Left**: Fully-connected networks, **Right**: Sparsely connected networks, following the structure described in Section 3.4.2. Lines denote the mean over 1000 steps, shaded region denotes the area between the fifth and ninety-fifth percentiles. The real-time limit is denoted with a horizontal dashed black line.

A.

```python
# General network
R = 20.0    # range of network activity (mV)
neuron_type = NonSpikingNeuron()    # generic neuron type
net = Network(name='Visual Network')    # create an empty network

# Retina
# add a 2d population the same size as the scaled image
net.add_population(neuron_type,shape,name='Retina')
# add a vector input for the flattened image
net.add_input('Retina', size=flat_size,name='Image')
# add a vector output from the retina
net.add_output('Retina',name='Retina Output')

# Lamina
net.add_population(neuron_type,shape,name='Lamina')

e_ex = 100.0    # excitatory reversal potential
e_in = -60.0    # inhibitory reversal potential
k_ex = 1.0  # excitatory gain
k_in = -1.0/9.0 # inhibitory gain
# calculate excitatory conductance
g_max_ex = (k_ex*R)/(del_e_ex-k_ex*R)
# calculate inhibitory conductance
g_max_in = (k_in*R)/(del_e_in-k_in*R)
# kernel matrix of synaptic conductances
g_max_kernel = np.array([[g_max_in, g_max_in, g_max_in],
                         [g_max_in, g_max_ex, g_max_in],
                         [g_max_in, g_max_in, g_max_in]])
# kernel matrix of synaptic reversal potentials
e_kernel = np.array([[e_in, e_in, e_in],
                     [e_in, e_ex, e_in],
                     [e_in, e_in, e_in]])
# pattern connection (acts as high pass filter)
connection_hpf = NonSpikingPatternConnection(g_max_kernel,e_kernel)
# connect the retina to the lamina
net.add_connection(connection_hpf,'Retina','Lamina',name='HPF')
# add a vector output from the lamina
net.add_output('Lamina',name='Lamina Output')

net.render_graph(view=True) # view the network diagram
```

B.



C.

Figure 3.5: Using SNS-Toolbox to design a two-layer visual processing system. **A.** Python code to generate the desired network. Image preprocessing and output plotting are omitted. **B.** Network visual representation. An input image is converted to stimulus current for a population of neurons, representing the insect retina. From the retina, a 33 kernel of inhibitory (light blue) and excitatory (purple) synapses is applied to create a high-pass filtering effect in the next layer, representing the L1 insect lamina neurons. **C.** Output of retina and lamina neurons, respectively. Voltages are mapped to grayscale intensities.

network as 5 ms.

We assume that input images are grayscale, because the retina neurons used within the motion vision pathway only respond to a single light color (green)[25]. Since a single *Drosophila* eye consists of around 800 ommatidia arranged in 32-34 columns[75], we will also design for input images which are 32x32 pixels. As an input transformation to the retina layer, pixel values are linearly scaled from their original 0-255 to 0-R nA.

After creating two 32x32 populations of neurons and attaching an input source, the last step is to define the connection between the retina and the lamina. Our lamina model only consists of L1 neurons, so each neuron has center-on surround-off receptive field. Since each L1 cell receives input from the directly adjacent

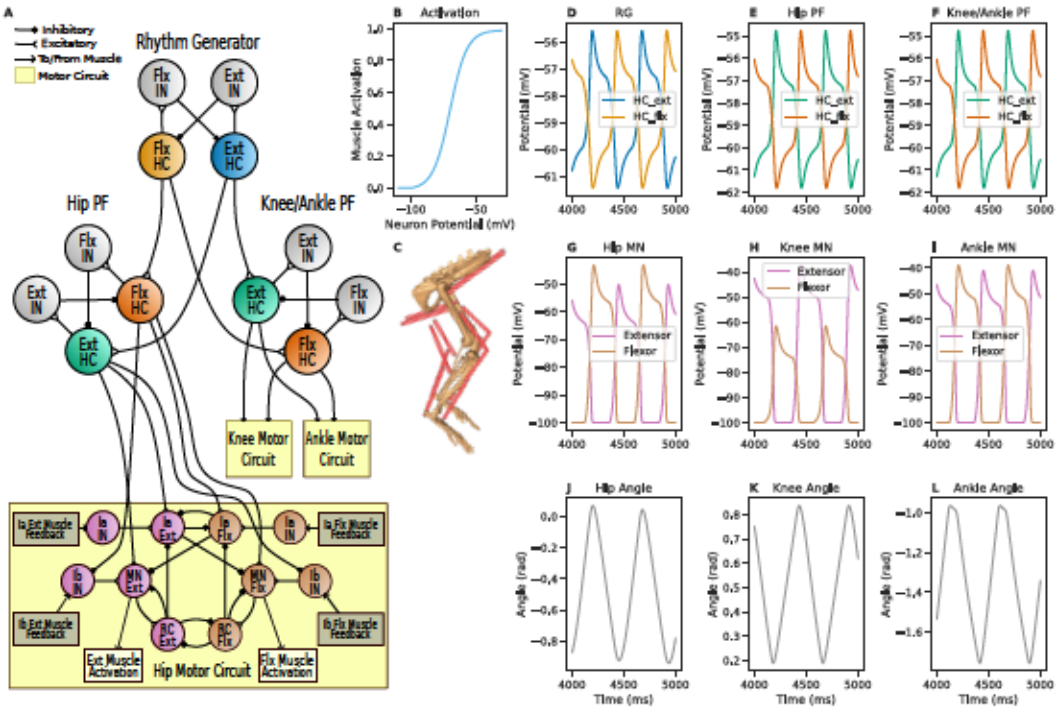


Figure 3.7: SNS-Toolbox controls a musculoskeletal model of a rat hindlimb. **(A)**: Diagram of the neural control network. **(B)**: Relationship between motor neuron voltage and muscle activation. **(C)**: The musculoskeletal model used in Mujoco [127]. **(D)**: Neural activity from the half-center neurons in the central rhythm generator. **(E,F)**: Neural activity from the hip and knee/ankle pattern formation circuits. **(G–I)**: Motor neuron activity in the motor circuits for the hip, knee, and ankle. **(J–L)**: Joint angles of the hip, knee, and ankle. All recordings are shown for a period of 1000 ms, after the model has finished initialization. Pictured are recordings from the elements within the left leg.

calculated in the same manner as [139], with the activation sigmoid defined as

$$\frac{1}{1} \tag{3.38}$$

where is the steepness of the sigmoid and is the motoneuron potential. The exact curve is shown in Figure 3.7B.

Simulation Results

The network and mechanical model are simulated for 5000 ms, with data shown in Figure 3.7. On each step, muscle tensions from Mujoco are first formatted as Ia and

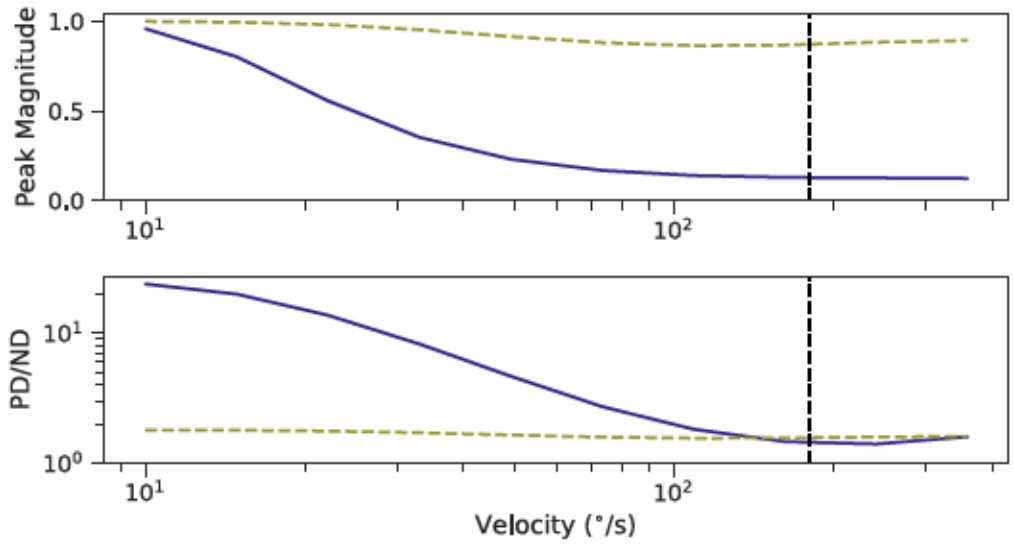


Figure 4.6: Output behavior of the On (solid indigo) and Off (dashed olive) motion detectors when subjected to a square wave, translating from 10 to 360 per second. Target maximum velocity (180) shown with a vertical dashed line. *Top:* Peak magnitude of the motion detector in the preferred direction; *Bottom:* Ratio between the motion detector in the preferred direction and the null direction.

input velocity increases [82]. However, in *Drosophila* this decrease occurs as input velocity is both increased and decreased from a peak velocity.

4.5.4 Directional Selectivity

Stimuli of a consistent wavelength and velocity are applied to the same network described in Section 4.5.3 while the direction of travel is varied from 0 360 in 45 increments, with results shown in Fig. 4.7. The EMD for each cardinal direction exhibits enhanced sensitivity to stimuli in the preferred direction, and reduced sensitivity to the other directions. The On pathway is able to generate a finer level of directional sensitivity than the Off pathway, due to its multiplicative window of reduced inhibition. Further work is necessary to find a similar multiplication mechanism for the Off pathway.

As the networks for each cardinal direction are mirrored versions of each other,

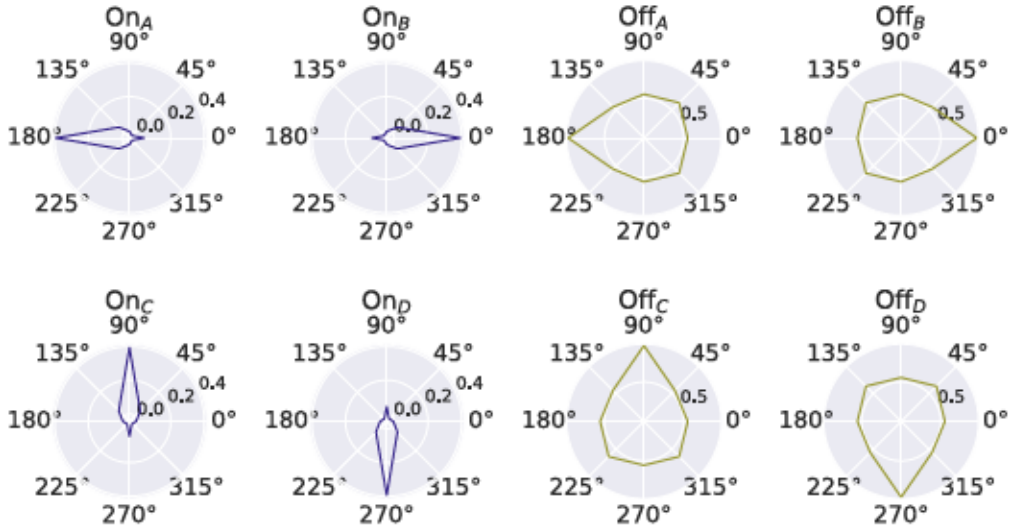


Figure 4.7: Peak response of each motion detector in the On (*Left*) and Off (*Right*) pathways to a square wave grating with 30 and 30 . Preferred direction of each sub-type: *A*: right to left; *B*: left to right *C*: bottom to top; *D*: top to bottom.

the resulting responses are identical except for their orientation. This is different than the tuning found in *Drosophila*, where the sensitivity of each cardinal direction is slightly different [82]. The general shape of our *On* and *Off* responses most closely matches the behavior of the T4b and T5b neurons in the animal, consisting of a sharp triangular point in the preferred direction and a slight bump in the null direction, however T5b is much more similar to T4b than our *Off* neurons are to the *On* neurons.

4.6 Discussion and Future Work

In this work, we implement an SNS network which is a reduced model of the *Drosophila* motion vision system. The network performs optic flow measurement at each point in the visual field, and can be tuned for different ranges of input stimuli in a parametric manner. While some parameters are found via numerical optimization, most are chosen by hand via analytic rules. With further optimization,

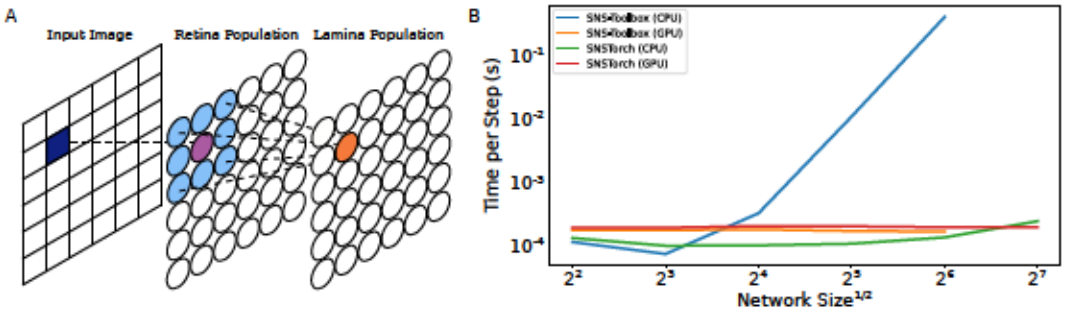


Figure 5.1: Comparison in performance between SNS-Toolbox and SNSTorch. **(A)** The network to be evaluated is the same structure as section, with two populations being connected by a convolutional synapse. **(B)** This network was compiled and then run in SNS-Toolbox and SNSTorch at increasing population size.

presynaptic and postsynaptic states and applies the dynamics in eq. 5.3, with each applying the conductance and reversal potentials based on differing connectivities. The dense connection implements a synapse from every presynaptic neuron to every postsynaptic neuron, the elementwise connection only connects neurons with the same index, and the convolutional connection reuses a local pattern that is tiled across the population.

5.4 Results

To test SNSTorch, we first compare the performance of this system with SNS-Toolbox. We then evaluate the functionality of SNSTorch on two separate optimization tasks.

5.4.1 Comparison with SNS-Toolbox

Although SNSTorch simulates some of the same dynamics as SNS-Toolbox, internally they are structured differently. SNS-Toolbox makes no assumptions about connectivity and represents the entire network as a single recurrent population, which lends it well to densely recurrent networks with nested feedback loops such as those in locomotion [68]. SNSTorch on the other hand is designed in the more

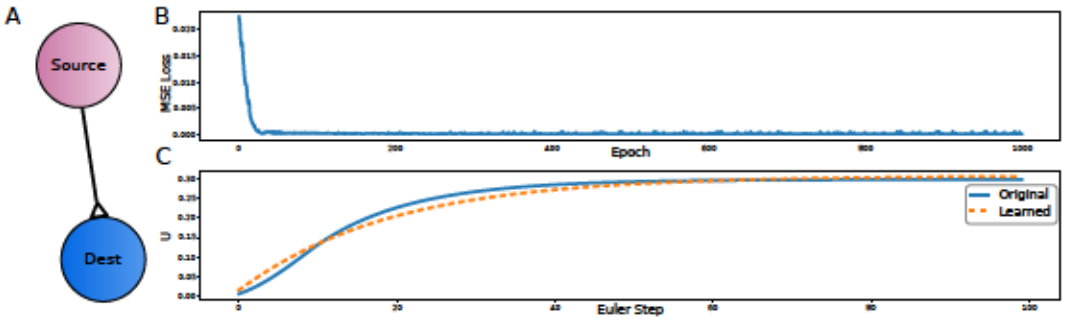


Figure 5.2: Using SNSTorch for a parameter identification task. (**A**) We are trying to match the behavior of a simple network of neurons, where one neuron receives a random stimulus and excites the other neuron via an excitatory chemical synapse. Using a ground truth model, the network learned the neural and synaptic properties to replicate this behavior. We chose to focus on minimizing the mean-squared error between the final state of the original and trained network. Shown in (**B**) is the training loss over 1000 random stimuli, and in (**C**) we plot the trajectory of the postsynaptic neuron in the original and trained networks.

conventional fashion in deep learning, where each population is represented and evaluated individually as its own layer. While this means arbitrary connectivity is more challenging, the payoff is in higher speed and reduced memory consumption for large networks. To demonstrate this, we simulated the same network structure using SNS-Toolbox and SNSTorch, running it on the CPU and GPU and varying across large differences in network size. Results can be seen in Fig. 5.1. For small networks, SNS-Toolbox runs faster than SNSTorch. As the network size increases, SNSTorch quickly becomes the fastest solution (or only solution, as memory consumption increases).

5.4.2 Parameter Tuning and Regression

As a toy example, we use SNSTorch for parameter tuning in a simple network (shown in Fig. 5.2A). Two neurons are connected via a chemical synapse, and the presynaptic neuron is stimulated with a constant input. In this task, we have two versions of this network: one which acts as the target, and the other which must be trained to match the target. This system has 12 parameters: , , , , and 0 for

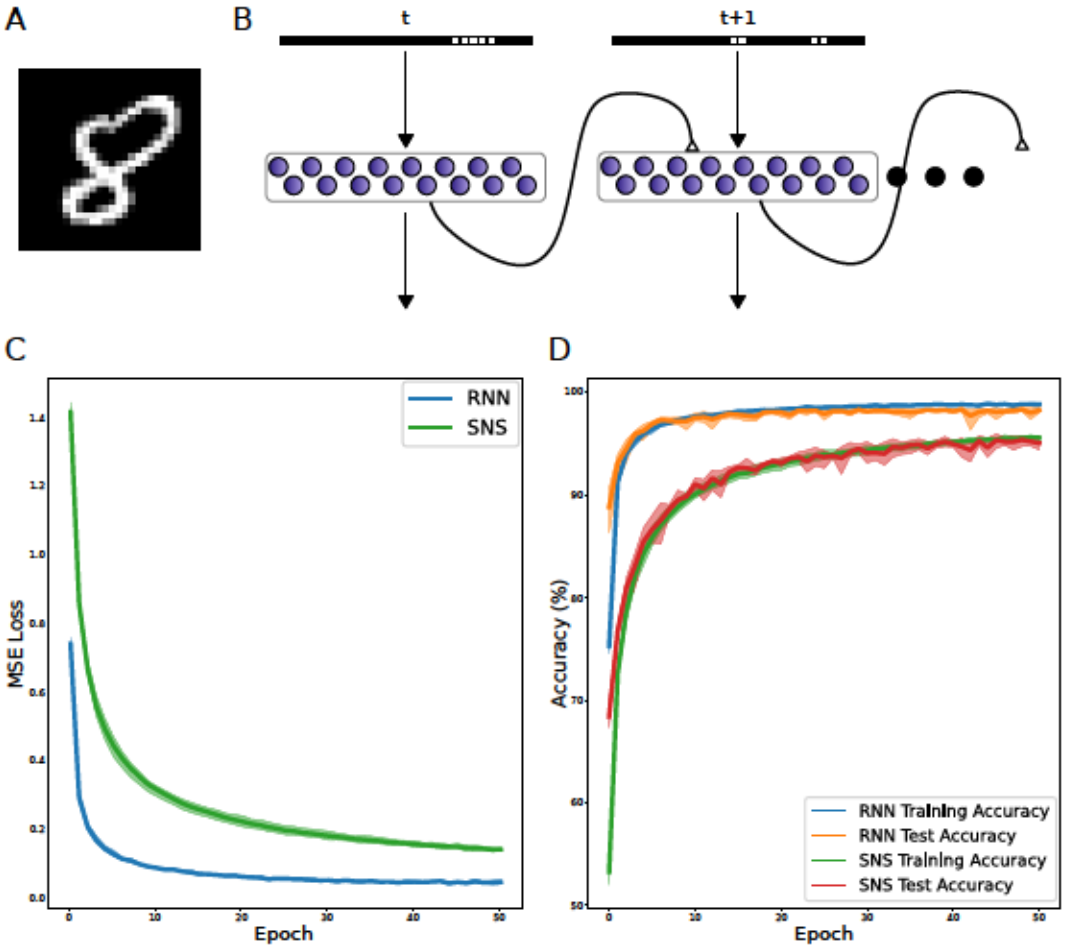


Figure 5.3: Training an SNS for sequence classification. (**A**) We train an SNS network to classify the row-wise sequential MNIST dataset [78], where each handwritten digit is divided into 28 1x28 images. (**B**) The SNS network consists of a single recurrent layer of non-spiking neurons, with the recurrence implemented using chemical synapses. Training loss (**C**) and accuracy (**D**) of the SNS network and an RNN with a similar number of parameters. Line denotes the mean across five trials, the shaded area denotes the fifth and ninety-fifth percentiles.

when simulating large networks of neurons which could be structured based on layers of populations. It is also primarily a tool for design, so it is not optimized for optimizing parameters without recompiling the network. In this chapter, we present SNSTorch as a companion software package to SNS-Toolbox. SNSTorch simulates some of the same dynamics as SNS-Toolbox, while being able to simulate larger networks and supporting optimization via automatic differentiation in PyTorch. We

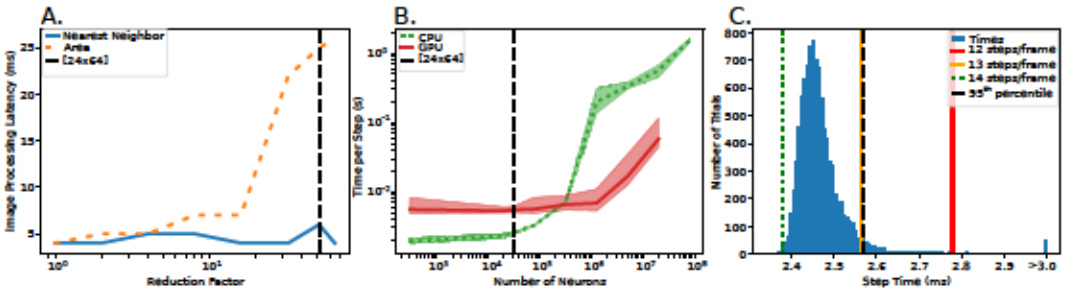


Figure 6.5: Timing performance of image formatting and processing execution on target hardware. **A:** Latency in image processing as the target image reduces in size. Two different interpolation methods are compared, with nearest-neighbor interpolation shown in solid blue and area interpolation shown in dashed orange. A vertical dashed line is present at the image resolution 24x64, the scaled dimensions used in our dataset. **B:** Time per simulation step of our visual motion processing network, in seconds, as the dimensionality of the input increases. Execution on the Jetson Orin Nano CPU are shown in dotted green, and times for the Jetson Orin Nano GPU are shown in solid red. Dark lines correspond to the average, the shaded area corresponds to the 5th and 95th percentiles over 1000 steps. We use a vertical dashed line to denote the dimensionality corresponding to an input image size of 24x64 pixels. **C:** Detailed timing of our network with an input dimensionality of 24x64 pixels. Shown is a histogram of time per simulation step in milliseconds, over a testing run of 10,000 steps. A black dashed vertical line denotes the 95th percentile of the distribution. Shown in dashed green, solid orange, and solid red would be the time per step needed for 14, 13, or 12 simulation steps per video frame. In this work we chose to use 13 steps per frame for our simulations.

al. [93], with some adjustments to account for the use of natural images instead of simulated square gratings. The full network is shown in Fig. 6.6. In this section we will begin with an overview of the neural modeling techniques employed, and then will examine the design of each individual network section, emphasizing the changes made in this work. The network and all remaining support code can be found at https://github.com/wnourse05/FlyWheelBaseline-LivingMachines2024.

6.5.1 Neural Modeling

We choose to implement our motion-vision processing network as a Synthetic Nervous System (SNS) of non-spiking leaky integrator neurons, where the neural

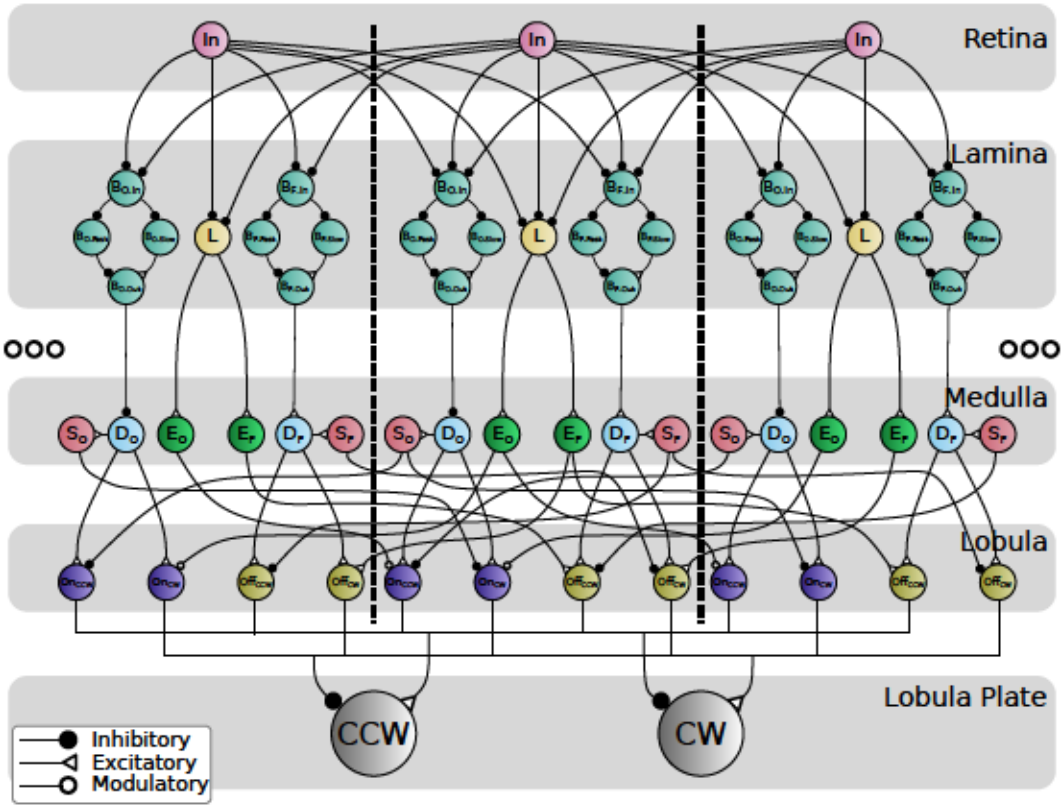


Figure 6.6: Visual motion processing network used in this work, inspired by the anatomy of *Drosophila melanogaster* and adapted from [93]. Visual stimuli are encoded into a neural representation in the retina. They are then spatiotemporally filtered in the lamina, and temporally filtered again in the medulla. The lobula combines the neural activity in the medulla into estimates of motion at each pixel, and these estimates are summed across the entire visual field to generate a global estimate of motion in the lobula plate.

state is updated as

(6.1)

where is the neural time constant, is any external input, and is a constant bias term. is the synaptic input from any presynaptic neurons in the network,

(6.2)

with denoting a presynaptic neuron, and denoting the synaptic reversal potential. Throughout the network, we design for neurons to communicate when

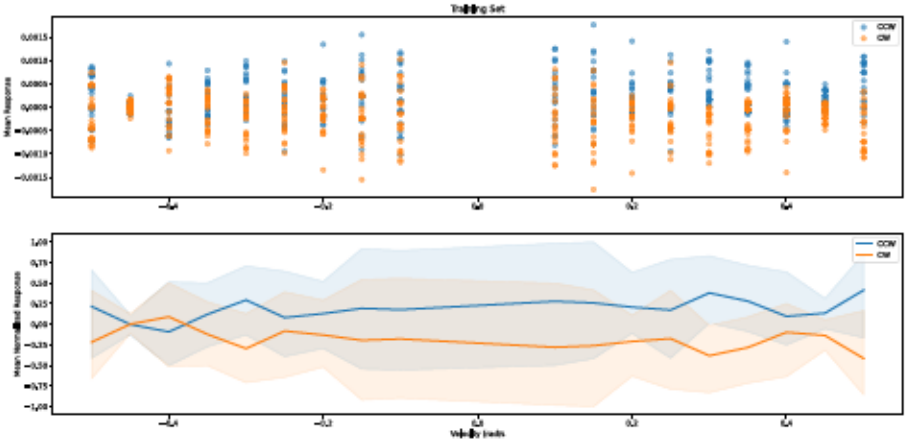


Figure 6.8: Performance of the simple motion vision processing network on the video clips in the test portion of the FlyWheel dataset. **(A)** Scatterplot of average neuron state for the clockwise and counter-clockwise neurons for each image sequence. **B.** Curves denote the mean neural response of all trials at each velocity, shaded area represents the 5th and 95th percentiles. All data is normalized to the maximum of the 95th percentile across all velocities.

6.5.7 Lobula Plate

In the final layer, we extend the network presented in Chapter 4 to include an approximation of the circuitry present in the *Drosophila melanogaster* lobula plate. We add two horizontal sensitive neurons, and , one corresponding to counter-clockwise rotation and the other to clockwise rotation. These neurons receive synaptic input from every motion detector neuron in the lobula, with the counter-clockwise sensitive detectors exciting and inhibiting , and the inverse case for clockwise sensitive detectors [13]. These neurons have a time constant of , and the synapses are tuned using eq. 6.5 such that the sum of all the synaptic gains is one.