DEVELOPMENT OF AN ASTROCYTIC MODULE FOR SPIKING NEURAL NETWORKS ON NEUROMORPHIC HARDWARE

BY ARPIT SHAH

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Konstantinos P. Michmizos

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2019

ABSTRACT OF THE THESIS

Development of an Astrocytic Module for Spiking Neural Networks on Neuromorphic Hardware

by Arpit Shah

Thesis Director: Konstantinos P. Michmizos

Astrocytes have long been neglected in application to neuronal networks due to being electrically silent. While these glial cells have been hypothesized to serve as a support for neurons, recent research suggests that they may have a role in learning through spatial and temporal modulation of neurons. Astrocytes may form their own networks and communicate amongst themselves through calcium signaling. They have so far been absent in the Spiking neural networks (SNNs) and consequently, they have not been incorporated into neuromorphic chips such as Intel's Loihi. In this work, we discuss a new astrocytic module to extend the capabilities of Loihi to facilitate the inclusion of astrocytes in SNNs. This transformation from SNNs to Spiking Neural-Astrocytic Networks (SNANs) would enable researchers to both explore and leverage the capabilities of astrocytes in neuromorphic hardware. The module serves as a higher-level interface on top of Intel's NxSDK to allocate resources which serve as internal components of our astrocyte model to inject Slow Inward Current (SIC) and then introduce synchronous activity in the postsynaptic neurons. In addition, this work also addresses an additional project focused on the Unidimensional SLAM problem where we focus on solely the orientation of a robot placed in a variety of environments. We show that the spike-based algorithm implemented on Loihi requires approximately 100

times less power than the state-of-the-art GMapping algorithm implemented on a CPU. This work demonstrates the viability of Spiking Neural Networks running on Loihi as an alternative solution for mobile robots.

Acknowledgements

I would like to thank Professor Konstantinos Michmizos for his guidance and support throughout my thesis. I have had the rare opportunity and privilege of participating in research within the Computational Brain Lab for almost 2 years following a course with the professor. His dedication to the success of his students is clearly evident through his patience and efforts to impart wisdom to both myself and others at the lab. It is through his dedication and unwavering guidance that this work is complete today.

I am also grateful to Guangzhi Tang, with whom I collaborated with for the research related this work [1]. His focus, work ethic, and the quality of his work has served as an example to myself for the expectations I should always strive to meet in the future.

I would also like to thank Neelesh Kumar, Karan Banga, Giannis Polykretis, and Vladimir Ivanov for their guidance and support both in and outside of the lab.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the development of large-scale neuromorphic processors by companies such as Intel and IBM, there are more opportunities to explore the viability of Spiking Neural Networks (SNNs). Intel's neuromorphic chip, Loihi, serves as a power-efficient alternative to traditional hardware for developing and running SNNs. While Intel's NxSDK enables developers and researchers to create neurons, the API for the SDK does not provide an easy method for creating astrocytes on the neuromorphic chip.

The work discussed in Chapter 4 focuses on the use of SNNs and Loihi to achieve results comparable to the state-of-the-art solutions available on traditional Von Neumann architecture while utilizing power approximately 100 times less than conventional CPUs. This work can lead to higher power efficiency in intelligent robotic agents deployed for a variety of different applications such as space exploration and crisis response.

1.2 Prior Work

While there has previously been no notion of an astrocyte introduced to a large-scale neuromorphic processor, Intel's Neuromorphic Computing Lab provides NxSDK, a Python-based API for creating SNNs and a toolchain for compiling and running those SNNs on Loihi [2]. The API serves as both an inspiration and basis for the astrocyte module, *combra_loihi*.

1.3 Description of the remaining chapters

There are 4 other chapters remaining. Chapter 2 provides background knowledge and theory relevant to Chapters 3 and 4. The first half of Chapter 2 (Section 2.1 - 2.3) discuss some introductory material related to Spiking Neural Networks, the Leaky Integrate-and-Fire model, astrocytes, and Loihi. These sections prepare for Chapter 3 which focuses on the first project - the design and implementation of an astrocytic module running on top of Loihi. The latter half of Chapter 2 introduce the SLAM problem, ROS, and the Gazebo simulator - all in relevance to Chapter 4 - measuring power efficiency of SLAM with GMapping. Finally, Chapter 5 provides a conclusion to both research projects and some work which may be done in the future to further improve upon the work detailed in this work.

Chapter 2

Theory

2.1 Spiking Neural Networks

A spiking neural network (SNN) is fundamentally different from the traditional neural networks that have been gaining attention in media and the technology industry as a whole in the past decade. SNNs leverage the knowledge of how neurons behave and interact to form larger systems from neuroscience and the computational power of machines to reproduce the intelligent behavior which can be found in humans and other living organisms. One of the beliefs is that by simulating neurons as they behave in the brain, it may be possible to leverage them to control robots, discern different patterns and tackle tasks much closer to how a human may approach them.

One excellent example of such an application is the work by P. Balachandar and K. Michmizos [3]. Using various neural studies, Balachandar was able to develop a neuromorphic oculomotor controller which would process the vision input as spikes injected into biologically plausible models of neurons and their connectome to generate the spike activity representing the motion needed for a robotic head to track a target. The SNN utilized models representing the retina structure for encoding the images and multiple types of neurons such as Ipsilateral Feedback Neurons, Tonic Neurons and Motor Neurons among others to accomplish the horizontal and vertical movements of the head [3].

In the following sections, we discuss the basic components of a Spiking Neural Network such as the Leaky Integrate-and-Fire model on Intel's Loihi chip, synapses, and how a simple SNN can be constructed conceptually.

2.1.1 Neurons

Neurons are one of the core processing units in the brain. At the most fundamental level of the Spiking Neural Networks, artificial neurons are implemented using simple mathematical models representing the behavior observed in their biological counterparts. For example, the simplest model of a neuron, the Integrate-and-Fire model, accumulates the membrane potential (voltage) until that value crosses a certain threshold (integrate) and then emits a spike (fire). The most basic properties of the neuron models in a Spiking Neural Network are the input current ($I(t)$), the voltage/membrane potential ($V(t)$) at some time t. Mathematically, the Integrate-and-Fire model can be stated as:

$$C_m \frac{dV}{dt} = I(t) \tag{2.1}$$

where C_m represents the capacitance, V is the voltage at a given timestep and $I(t)$ is the input current at that timestep.

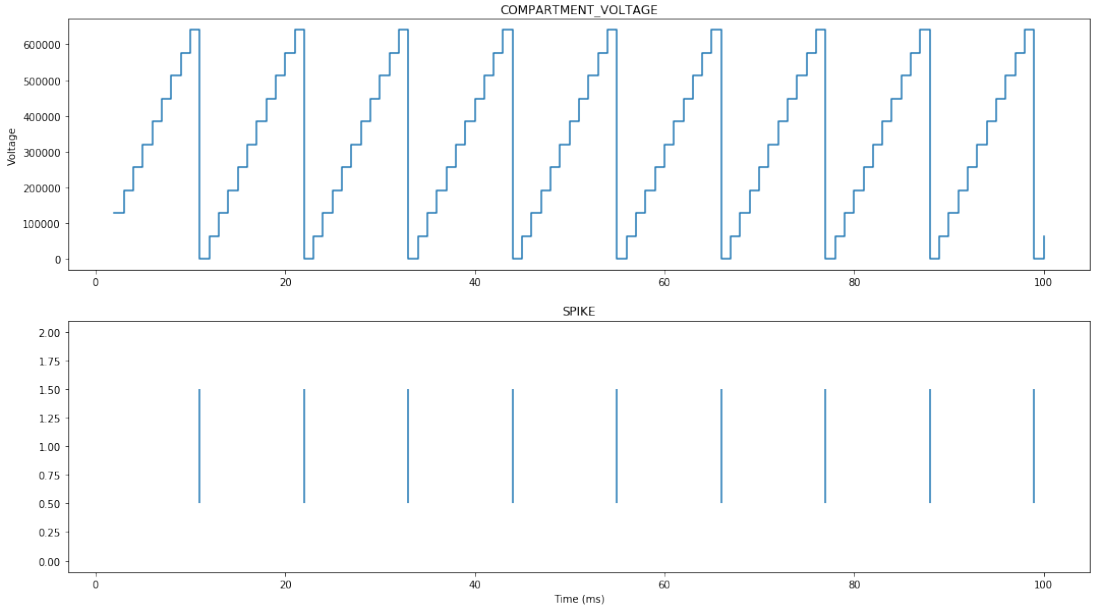

Figure 2.1: Voltage and spike activity of an Integrate-and-Fire neuron

Whenever a neuron "fires" or emits a spike, the voltage potential is reset to some value V_r. This reset value and the threshold for the neuron's voltage potential can also

influence the frequency of the neuron's spikes as having a lower threshold can enable the neuron to fire much more easily.

2.1.2 Leaky Integrate-and-Fire Neuron model

While the simple Integrate-and-Fire model is computationally efficient, it does not exhibit the "leak" in the membrane potential that we can observe in actual neurons. The leak in potential is accomplished by further modifying the mathematical formula for the Integrate-and-Fire model to be the following:

$$C_m \frac{dV}{dt} = I(t) - \frac{V_m(t)}{R_m} \tag{2.2}$$

where C_m is the capacitance, $\frac{dV}{dt}$ is the change in the voltage potential, $I(t)$ is the present input current into the neuron, $V_m(t)$ is the voltage potential at that timestep, and R_m represents the resistance. The additional term in the differential equation represents the "leak" or decay in the voltage potential.

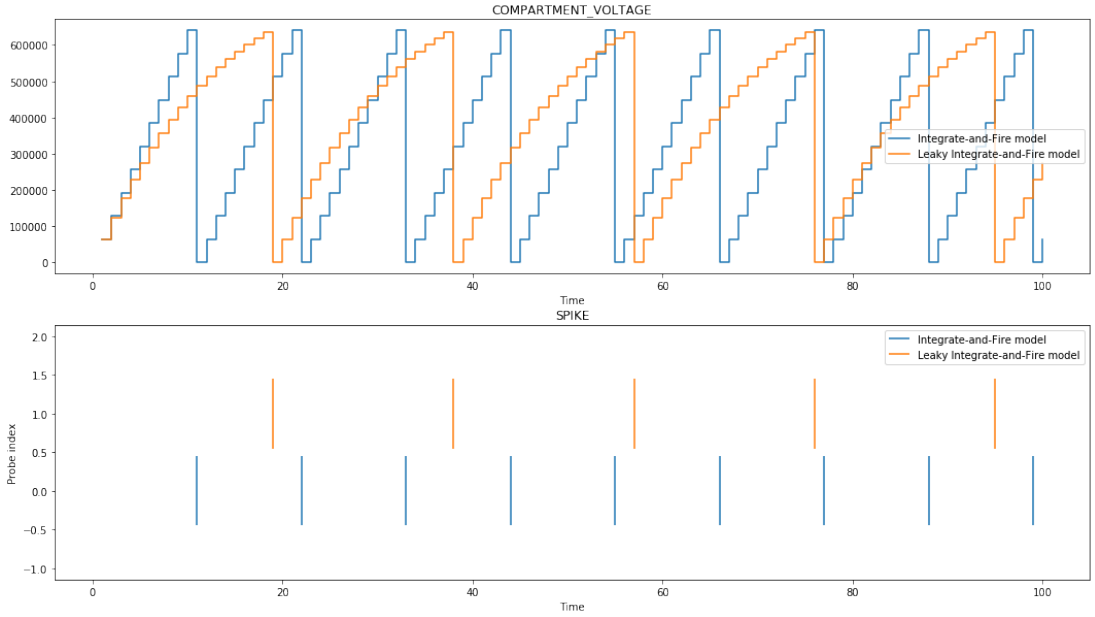

Figure 2.2: Comparison of Leaky Integrate-and-Fire neuron (orange) and normal Integrate-and-Fire neuron model (blue)

2.1.3 Synapses and Networks

With the understanding of how the artificial neurons themselves behave, we can begin to scale our understanding to the interaction between neurons through the synapses connecting them and beyond that, the overall neural network that is established as we increase the number of neurons and synapses.

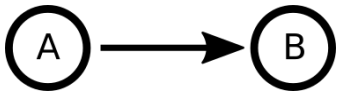

Figure 2.3: Diagram of a simple network with 2 neurons

In the figure above, we show some neuron A being connected to another neuron B through a synapse. Neuron A would be considered the *pre*synaptic neuron and neuron B would be the *post*synaptic neuron. The synapse itself may have properties of its own such as the strength/weight of the synapse.

When there is a spike emitted from the presynaptic neuron, the input current for the postsynaptic neuron is increased by an increment with regards to the weight of the synapse connecting the two neurons.

$$\frac{dIj}{dt} = w_{ij} \tag{2.3}$$

where i is the presynaptic neuron, j is the postsynaptic neuron, w_{ij} is the weight of the synapse from neuron i to neuron j and I_j is the input current at neuron j. The SNN illustrated in Figure 2.3 was simulated and its activities can be provided in Figure 2.4.

2.2 Astrocytes

Astrocytes are a type of glial cells found in the nervous system. Historically, astrocytes were once regarded as primarily just insulators and supporting cells [4] due to being electrically silent when researchers injected electrodes. Advancements in Calcium imaging have led to the revelation that astrocytes may play a larger role [5]. They can influence synaptic learning which is key for processing information [6].

Astrocytes can modulate the synaptic activity by reacting to the spikes from the

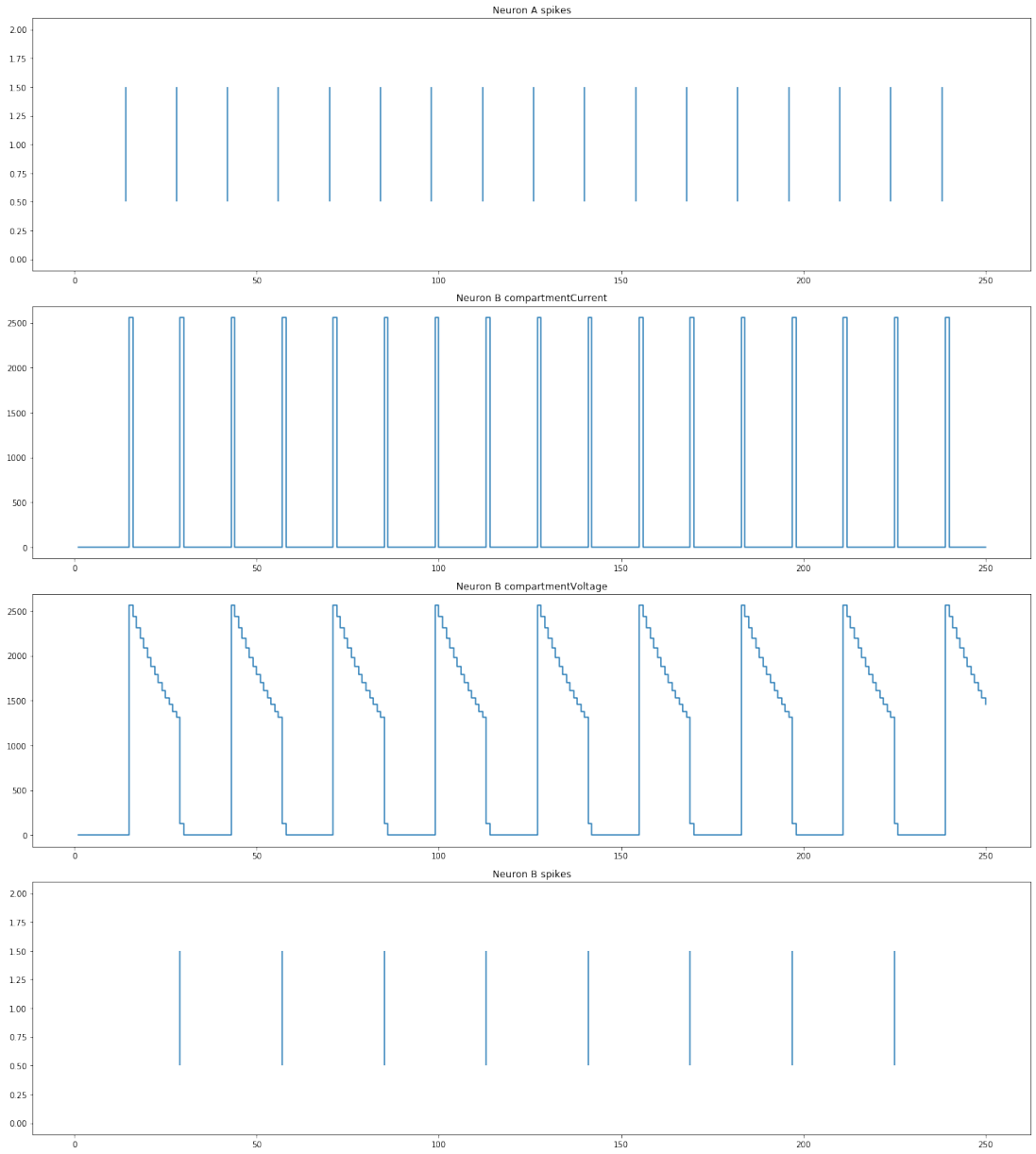

Figure 2.4: Voltage in neuron B over time as neuron A regularly fires

presynaptic neurons and releasing gliotransmitters. Neurons communicate with astrocytes at the tripartite synapses through a spillover of synaptic transmitters which bind to receptors linked to IP_3 production. Once the IP_3 concentration exceeds a certain threshold, an intracellular Ca^{2+} wave is released. From this Ca^{2+} wave, a release of gliotransmitters is triggered - resulting in a Slow Inward Current (SIC) being introduced to the tripartite synapse and more specifically, the postsynaptic neurons. Through the SIC, astrocytes are capable of synchronizing the postsynaptic neuronal activity [7]. Figure 2.5 illustrates the interaction between the astrocyte and the neurons at the site of the tripartite synapses.

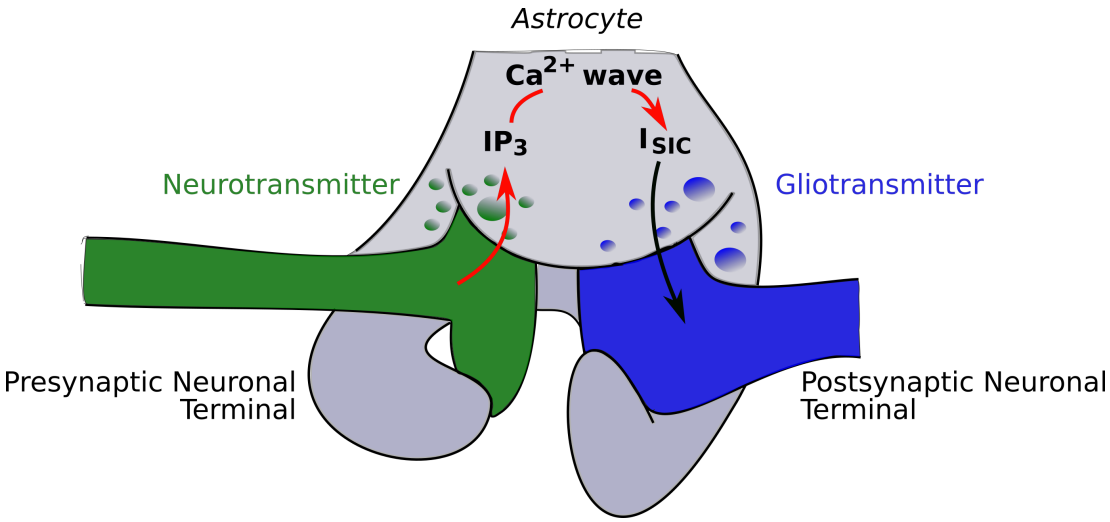

Figure 2.5: Communication at a tripartite synapse

While the communication between neurons may be on the scale of milliseconds and individual synapses, astrocytes can integrate activity from multiple synapses across a single microdomain, and ultimately multiple microdomains, with their activity spanning over anywhere between milliseconds to tens of seconds. As the synchronous activity is introduced to the postsynaptic neurons, it influences the learning in Spiking Neural Networks. During the same period of synchronous activity in the postsynaptic neurons, the synaptic weights between those neurons and presynaptic neurons that spike are affected as well. The variation in the spatial and temporal dimension through the generation of the Ca^{2+} wave and injection of synchronous activity to the postsynaptic

neurons introduces the possibility of enhanced intelligence in spiking neural networks as we integrate the astrocytic processes into SNNs.

2.3 Intel's Neuromorphic Chip, Loihi

Loihi is a crucial part of the research discussed in the later chapters. The neuromorphic chip developed at Intel's Neuromorphic Computing Lab represents some of the most recent advances in the development of a large-scale neuromorphic processor. The non-Von Neumann architecture enables researchers and software engineers to leverage asynchronous parallelism in the pursuit of developing Spiking Neural Networks and leveraging their ability with much lower power consumption and high computational efficiency.

2.3.1 Chip Architecture

The chip is essentially a mesh of 128 cores communicating over 2 physical networks through the broadcasting of spikes from one core to the other as necessary [8]. Each core contains 1024 spiking neural compartments which can serve as neurons based on the Leaky Integrate-and-Fire model similar to how it is Section described 2.1.2. Each neuron has internal state values for the electrical current and voltage potential of the neuron. When the voltage exceeds a certain threshold, the neuron emits a spike sent over the physical networks to each of the destination neurons which may be located on other cores on the same chip or a different chip entirely.

This behavior can be represented as the following equations:

$$v_{reset} = 0 \tag{2.4}$$

$$spike(t) = 1 \; if \; v \geq v_{threshold} \; else \; 0 \tag{2.5}$$

$$v(t) = 0 \; if \; spike(t-1) \; is \; 1 \tag{2.6}$$

where v_{reset} is the voltage to be set back to when the neuron spikes, $spike(t)$ is a boolean function to determine whether there is a spike emitted at some timestep t, and $v_{threshold}$ is the voltage threshold at which the voltage is reset to 0 following the timestep when the neuron emits a spike.

2.3.2 NxSDK

The NxSDK accompanies Intel's Loihi chip as a higher-level API for interacting with the chip itself. The SDK enables the developer to program in Python to allocate a group of compartments to represent neurons, synapses, define learning rules, inject spikes for input, and inspect the activity within the compartments [2]. Compartments and synapses are allocated dynamically by the underlying SDK code.

NxSDK provides the concept of an overall network and its subcomponents in an intuitive manner that should be easily relatable to the structure of SNNs themselves. Properties such as voltage thresholds for spiking, decay constants, and synaptic weights are exposed as members of the Compartment and Connection classes.

2.4 SLAM: Simultaneous Localization and Mapping

The SLAM problem poses the challenge of enabling a robot to determine the layout of its environment (mapping) and determine its location within that same environment (localization) using visual sensors and odometry readings. The difficulty of this problem comes from having each of the two tasks dependent on one another.

Solutions to the SLAM problem often involve the particle filtering approach where there are multiple particles instantiated in the map. Each particle represents a possible position and orientation (pose) of the robot and maintains its internal version of the map. As the robot moves, the odometry information is applied to each of the particles, and then each particle is evaluated for a likelihood of finding the same observation as the robot given the supposed pose of the particle.

According to Murphy [9], we can represent this probability as:

$$P(x_{0:t}, m_t | z_{0:t}, u_{0:t}) \tag{2.7}$$

where $x_{0:t}$ is the pose from the beginning until time t, m_t is the map at that time for a particular particle, $z_{0:t}$ is the sequence of observations, and $u_{0:t}$ is the sequence of odometry measurements.

Taking advantage of the factorization, we can calculate the likelihood of the pose

and the map independently as shown in [10],

$$P(x_{0:t}, m_t | z_{0:t}, u_{0:t}) = P(x_{0:t} | z_{0:t}, u_{0:t}) * P(m_t | z_{0:t}, u_{0:t}) \tag{2.8}$$

This factorization serves as a basis for the Rao-Blackwellized particle filtering technique utilized in the implementation of FastSLAM and OpenSLAM's GMapping library.

2.5 ROS: Robot Operating System

ROS serves as a framework for developing and running multiple microservices as ROS nodes. The collection of ROS nodes communicate via TCP amongst each other through a system of topics and services where a given node may *publish* and *subscribe* to send and receive information. Each node registers with a "master" node that is unique to a given ROS network which may span multiple physical machines connected to the same actual WiFi/LAN network.

ROS itself provides the command-line interface needed to debug the services with respect to the data being published on the ROS topics and determining which ROS node may be listening to those topics. This becomes very useful as we launch multiple services built on top of ROS or those which can be utilized in tandem with ROS such as the Gazebo simulator and open-source ROS libraries including the ROS wrapper for GMapping and a pose publisher library to determine the current position and orientation of the robot in the map generated by GMapping.

Launching the ROS nodes themselves can be very easy through the CLI as shown below in a command used to launch a node which would command the robot to rotate:

```
$ rosrun turtlebot_control rotate-random.py
```

Through this command, we are able to launch a Python application within the package "turtlebot_control" and register it with the ROS master node.

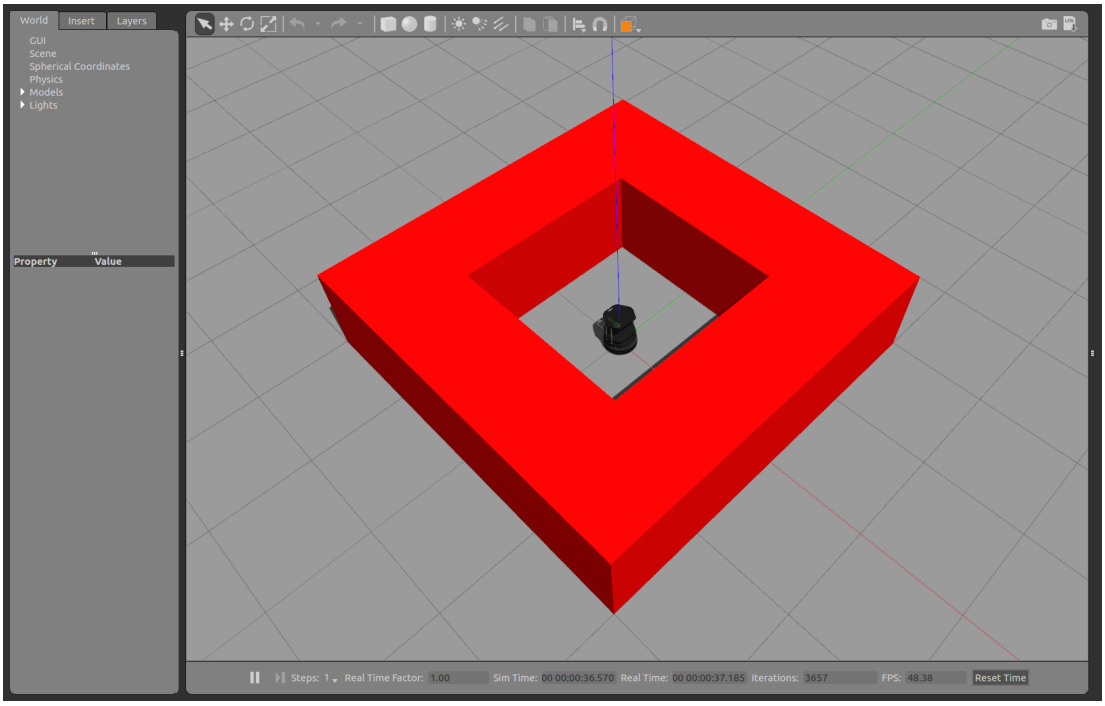

Figure 2.6: Gazebo running for a simple box environment with a Kobuki Turtlebot

2.6 Gazebo Simulator

The Gazebo Simulator is available both individually and alongside ROS as a tool in robotics for simulating robots in a virtual environment. The simulator offers out-of-the-box models for many robots and sensors in addition to the ability to add noise within the simulations running on high-performance physics engines. The software is also extendible using plugins and allows for TCP/IP communication through Protobufs as well.

In the scope of the research discussed in this work, we use the ROS launch file provided with the simulator (turtlebot_world) and custom world files to run experiments within different environments. Figure 2.6 shows the UI for Gazebo simulating a Kobuki Turtlebot with a RGB-Depth Camera sensor in a fully-enclosed box environments.

Chapter 3

Implementation of Astrocytes on Loihi

3.1 Modeling an astrocyte on Loihi

To represent an astrocyte on Loihi, we focused on the behavior of a microdomain behaving under the constraints of the neuromorphic hardware. Given that the neuron compartments of Loihi behave according to only the Leaky Integrate-and-Fire model, we reproduced the behavior of the Slow Inward Current (SIC) and the inject synchronous activity using multiple compartments.

Architecturally, we needed to accomplish the following:

1. Receive inputs from all of the tripartite synapses within the astrocyte's domain
2. Simulate the IP3 Calcium wave
3. Generate the SIC
4. Inject the current back to the tripartite synapses

Each of these tasks corresponds to a neuron compartment in the Astrocyte model. A connection from the presynaptic neuron to a *Spike Receiver (SR)* compartment for input and an additional connection for output is established from the *Spike Generator (SG)* compartment to the post-synaptic neuron to simulate the tripartite synapses. In total, the astrocyte model comprises of 4 compartments. Over time, the spike activity from the presynaptic neurons is aggregated at the Spike Receiver compartment which sends spikes to the IP3 compartment to simulate the IP3 calcium wave. When this compartment spikes, it triggers activity in the *SIC* compartment mimics the Slow Inward Current using the *Spike Generator* compartment as a bursting neuron for output to the postsynaptic neuron.

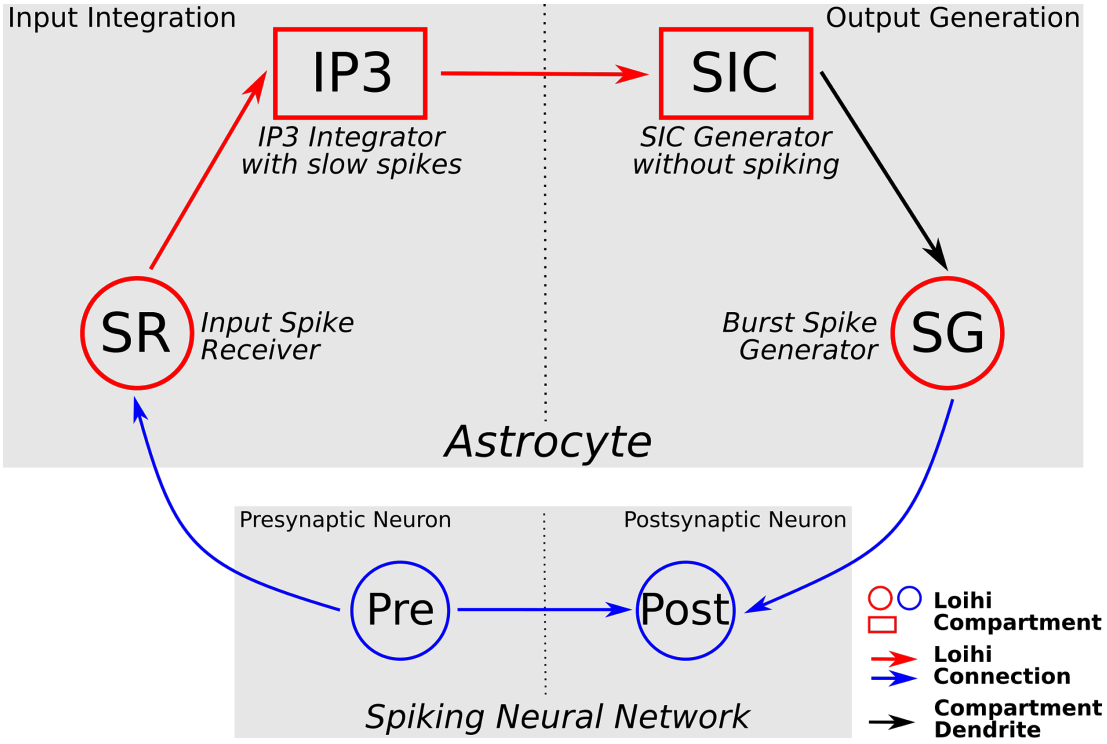

Figure 3.1: Diagram of Astrocyte Model on Loihi

One interesting thing to note is the interaction between the SIC compartment and the spike generator compartment where the voltage is integrated rather than emitting spikes. By doing this, we create the bursting Spike neuron where the activity is initially much higher and decreases over time as the voltage in the previous compartments decreases. Figure 3.2 illustrates this concept.

3.1.1 Reproducing the SIC on Loihi

As shown in Figure 3.2, we are integrating the voltage over time from the SIC compartment into the SG compartment. Due to the decay in the current and voltage within the SIC compartment, the voltage being propagated to the SG compartment diminishes over time. This diminishing behavior in the voltage leads to a much higher frequency in the spike activity from the SG compartment in the beginning and then a decreasing spike frequency until the voltage is no longer sufficient to keep the SG compartment spiking further.

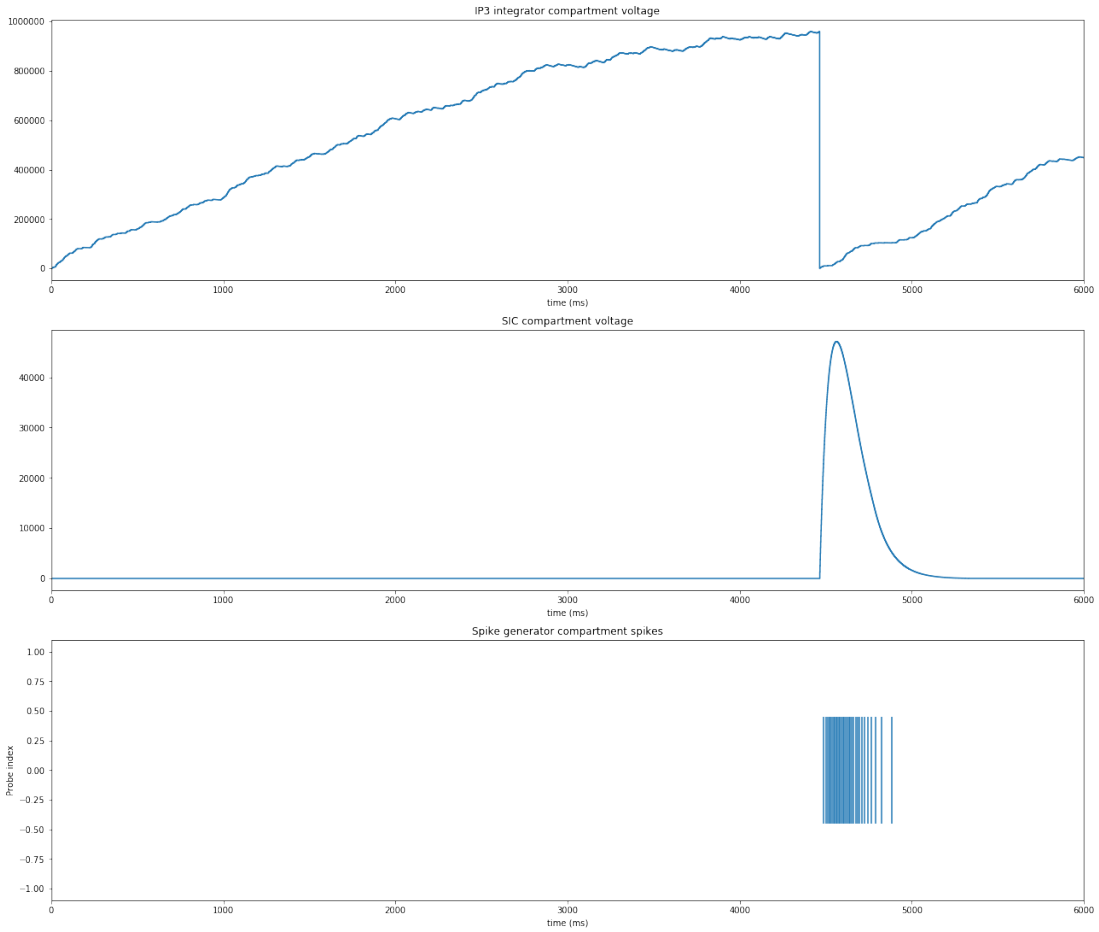

Figure 3.2: Relationship between SIC voltage and spike activity in SG compartment

3.2 API for an astrocyte

The design of the module uses core Object-Oriented programming principles. The astrocycte's interactions with the presynaptic and postsynaptic neurons are represented through the functions available for the user. When instantiating an astrocyte, the developer (user) may invoke the constructor and provide any custom parameters to tweak the model as they wish. The API exposes 19 parameters comprising of the properties of each part of the internal representation on Loihi.

Table 3.1 outlines some of the properties of the components within the Astrocyte model we've developed.

Also, the API provides access to the weights of the following synapses:

Spike Receiver (SR)	IP3	SIC	Spike Generator (SG)
srVThMant	ip3VThMant	sicCurrentDecay	sgVThMant
srCurrentDecay	ip3CurrentDecay	sicVoltageDecay	sgCurrentDecay
srVoltageDecay	ip3VoltageDecay		sgVoltageDecay
srActivityImpulse			
srMinActivity			
srMaxActivity			
srHomeostasisGain			
srEnableHomeostasis			

Table 3.1: Properties for each of the compartments in the model

1. between Spike Receiver compartment and the IP3 compartment

2. between the IP3 compartment and the SIC compartment

There are also functions for connecting the presynaptic and postsynaptic neurons to form a tripartite connection between the presynaptic, postsynaptic and the astrocyte itself. Ultimately, even the weights on these connections of the presynaptic neuron to the actual site and the extra side to the postsynaptic neuron can be adjusted if needed to better represents a tripartite synapse or for more custom behavior. Ultimately, the hyperparameters of this current model need to be adjusted to best reflect the expected behavior of the astrocyte with regards to the network as designed by users.

3.3 Easy setup feature

While there are default parameter values provided for the 19 properties that influence the Astrocyte model on Loihi, we have also tackled the challenge of making the module more user-friendly by reducing the number of parameters necessary. The Astrocyte model as represented on Loihi can be surmised into a collection of 3 key properties: the window of the SIC, the sensitivity of the IP3 integration, and the maximum firing rate of the bursting neuron - representing the maximum amplitude of the SIC. Many of the previous 19 properties can be derived from these 3 main properties such as the window being a result of how the decay and the initial values on the SIC compartment behave.

Based on the structure of the astrocyte model on Loihi, the maximum firing rate and the time window for the SIC can be influenced by the weight of the synapse from the IP3

compartment to the SIC compartment and the current decay of the SIC compartment. The current and voltage of the SIC compartment can be represented as follows:

$$u_{SIC}(t) = u_{SIC}(t-1) * (2^{12} - uDecay) * 2^{-12} + 2^6 w_{IP3 \to SIC} s_{IP3}(t) \tag{3.1}$$

where u is the current, $uDecay$ is the current decay for the compartment, w is the weight of the synapse between the two compartments.

$$v_{SIC}(t) = v_{SIC}(t-1) * (2^{12} - vDecay_{SIC}) * 2^{-12} + u_{SIC}(t) + biasMant * 2^{biasExp} \tag{3.2}$$

where v is the compartment's voltage, $vDecay$ is the voltage decay of the compartment, and the last term is the bias current overall for the compartment.

Given that the firing rate of the Spike Generator (SG) compartment is dependent on the number of spikes in a given period of time and the spiking behavior itself is dependent on the voltage fed from the SIC compartment. By adjusting the current decay of the SIC compartment, we influence the voltage of the SIC compartment and the voltage of the SG compartment.

The window of firing for the busting neuron is calculated as the difference in milliseconds from the first timestep on the Loihi chip where the SG compartment emits a spike to the last timestep where the SG compartment emits a spike in the series of spikes ultimately triggered by a single spike from the IP3 compartment. If the weight of the synapse from the IP3 compartment to the SIC compartment increases, the change in current in the SIC compartment is increased which results in a longer time needed for the current and voltage in the compartment to decay until such a time where the voltage propagated to the SG compartment is insufficient in triggering further spikes.

3.3.1 Finding the optimal configuration

The API determines the optimal configuration values for the synaptic weight and current decay using a search algorithm. The algorithm optimizes a cost function to traverse the 2-dimensional space representing the firing rates and the spike time window lengths. Given a possible configuration pair with the corresponding firing rate and window size,

the library determines the cost as:

$$cost = (firingRate_{config} - firingRate_{goal})^2 + (windowSize_{config} - windowSize_{goal})^2 \tag{3.3}$$

3.3.2 Running a simple feed-forward SNN

The open-source code for the API is also accompanied by examples such as a simple feed-forward SNN. The SNN has 10 input neurons (presynaptic neurons) and 10 postsynaptic neurons. Each neuron in the presynaptic neuron is connected to a corresponding postsynaptic neuron based on index values within the lists respectively. With a single astrocyte forming tripartite synapses over all of these connections, we can see activity such as that illustrated below in Figure 3.4. The code corresponding to the SNN from 3.3 and 3.4 can be found in Appendix A.

3.4 Results

The Python-based astrocytic module for Loihi, *combra_loihi*, can simulate the behavior of an astrocyte within a Spiking Neural-Astrocytic Network (SNAN). We were able to use an astrocyte to accomplish the following:

- form tripartite synapses
- simulate presynaptic neurotrasmitters & postsynaptic gliotransmitters
- simulate a Ca^{2+} wave being propagated from an astrocyte
- introduce synchronous activity in the postsynaptic neurons

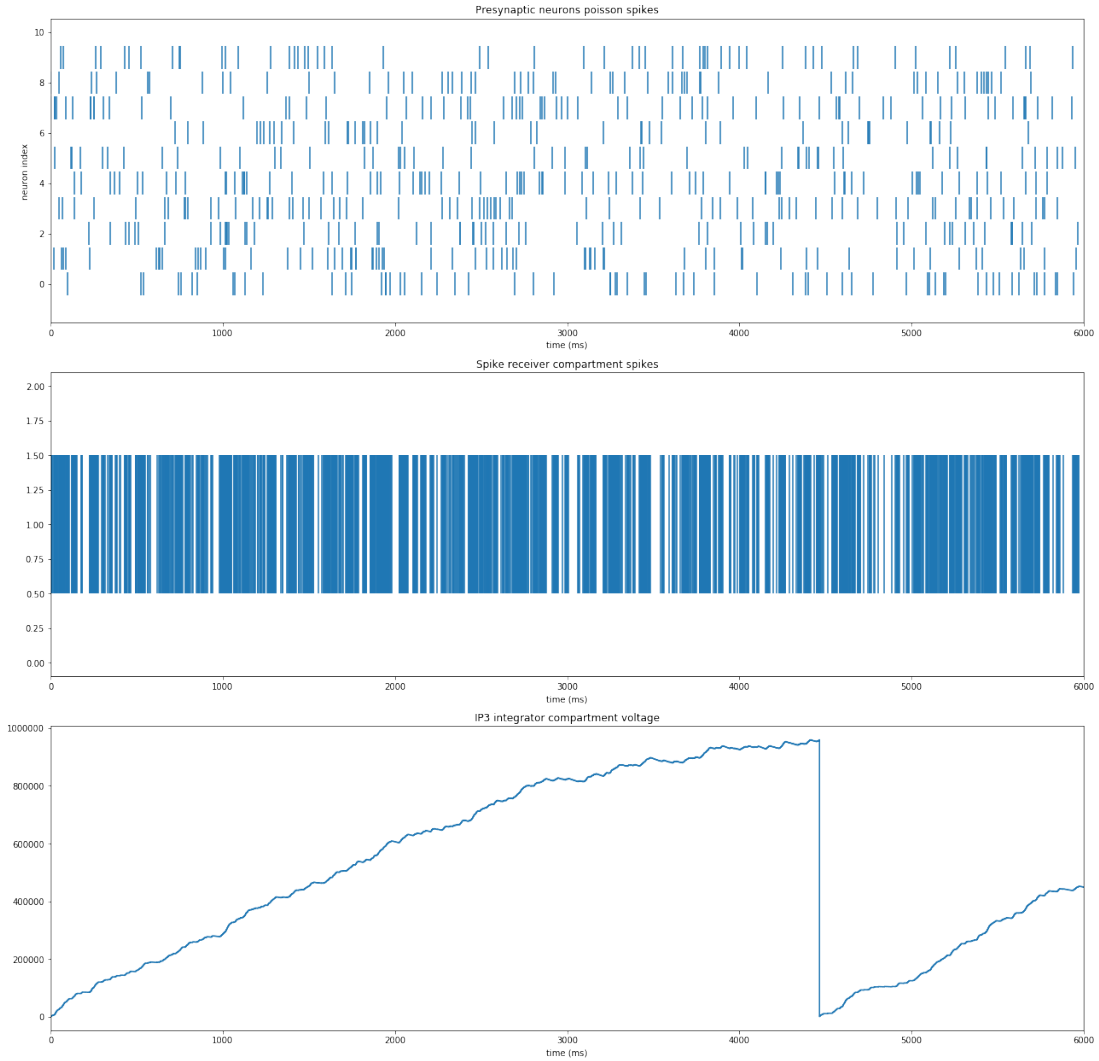

Figure 3.3: Input activity for a SNAN of 10 presynaptic and 10 postsynaptic neurons within the domain of an astrocyte

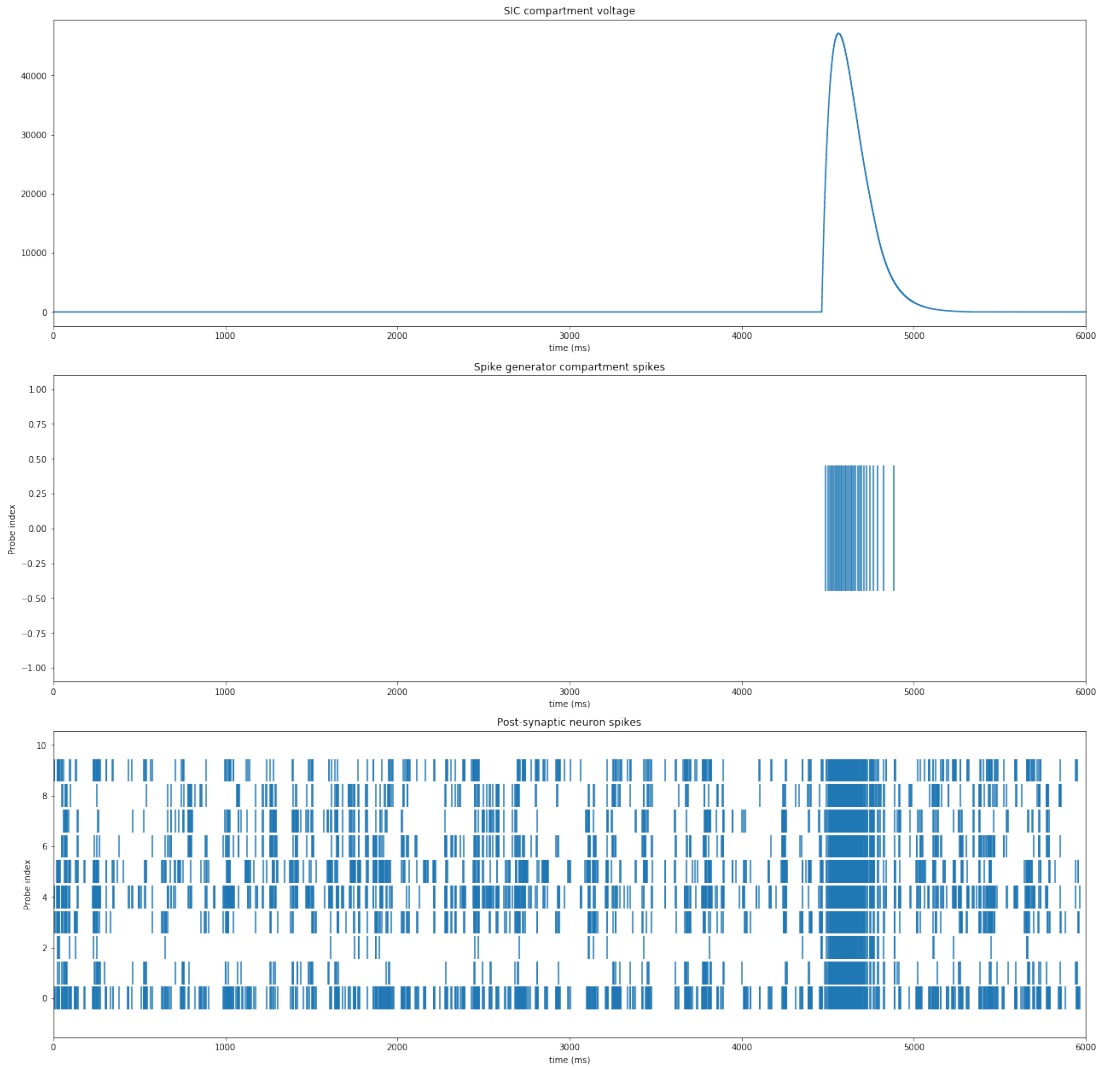

Figure 3.4: Activity in a simple SNAN of 10 presynaptic and 10 postsynaptic neurons within the domain of an astrocyte

Chapter 4

Unidimensional SLAM with GMapping on a CPU

Computational efficiency and power efficiency are critical for operating robots in multiple applications. To address this challenge, we compared the power consumption and accuracy we measured with a Spiking Neural Network on Loihi to the corresponding measurements for a CPU-based state-of-the-art method - namely, OpenSLAM's GMapping library.

To conduct the experiments using GMapping, we utilized multiple components:

- ROS [11]
- Gazebo simulator
- GMapping
- Pose Publishing library [12]
- Rotation script
- Pose error calculation script

4.1 Environments in Gazebo Simulator

We used the Gazebo simulator to simulate each of the virtual environments. The simulator allowed us to simulate what the robot's sensors would detect and provide that same information to the GMapping node.

There are 3 virtual environments to subject the robot to increasingly difficult circumstances and 1 real-world environment to test the robot with true noise. The first environment (a) shown in Figure 4.1 is a box enclosing the robot. As such, the robot has to discern between 4 identical corners based on the likelihood of facing a particular

direction based on the knowledge of the rotations thus far at any given time. Environment 2 (b) is a virtual representation of Environment 1 for use in Gazebo. There are gaps in Environment 3 (c) between the two objects, and the objects themselves vary in distance from the robot. In the last environment, the objects vary in shape and size as well in addition to the distance from the robot.

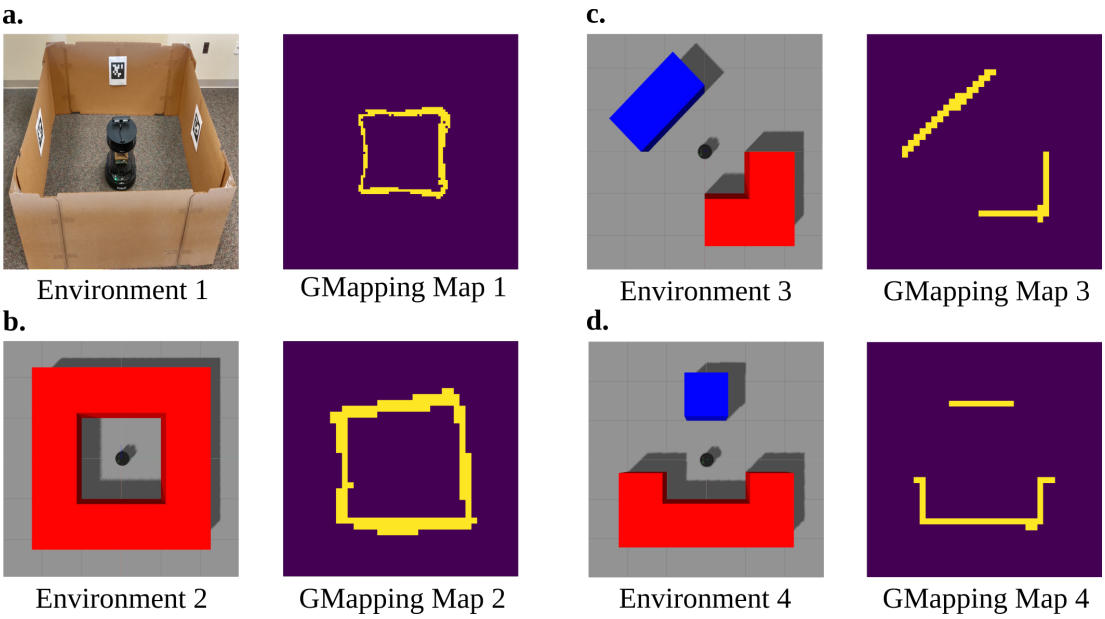

Figure 4.1: Environments

4.2 Measuring the Power Consumption

To measure the power consumption of using GMapping, we first measured the *idle* consumption with everything except the GMapping library being active and then the *running* power consumption where we have GMapping running during the experiment. Measuring the idle and running power multiple times better guaranteed the accuracy of the power measurements.

On Ubuntu 16.04, measuring the CPU power consumption can be done by utilizing the *powerstat* tool through the following command:

```
$ sudo powerstat [-R]
```

Time	User	Nice	Sys	Idle	IO	...	Watts
2:16:49	3.3	0	1.5	94.9	0.3	...	11.84
2:16:50	2.1	0	1.5	96.4	0	...	11.71
2:16:51	2	0	1.5	96.4	0	...	11.64
2:16:52	9	0	3	87.7	0.3	...	13.15
2:16:53	2.3	0	2	95.7	0	...	11.68
2:16:54	2.8	0	2	95.2	0	...	11.72
2:16:55	2.3	0	1.8	95.9	0	...	11.78
2:16:56	1.8	0	1.3	96.9	0	...	11.68
2:16:57	2.1	0	1.3	96.6	0	...	11.72
2:16:58	2.6	0	1.3	96.2	0	...	11.67
2:16:59	2.5	0	2	95.4	0	...	11.7
2:17:00	3.6	0	1.8	94.7	0	...	11.79
2:17:01	3.3	0	1.5	95.2	0	...	11.7
2:17:02	3.1	0	2.3	94.4	0.3	...	11.72
2:17:03	1.5	0	1.5	96.9	0	...	11.65
2:17:04	3.5	0	1.5	94.9	0	...	11.71
2:17:05	2.8	0	1.3	95.9	0	...	11.65
2:17:06	4.8	0	2.3	92.9	0	...	12.3
2:17:07	2.8	0	2	94.9	0.3	...	11.77
2:17:08	2.8	0	1.8	95.1	0.3	...	11.74
2:17:09	3.6	0	1.8	94.4	0.3	...	11.85
2:17:10	2.3	0	1	96.7	0	...	11.68

Table 4.1: Example power measurement output from powerstat on a CPU

4.3 Measuring the accuracy of GMapping

When running GMapping, we utilized a custom launch file with adjusted parameters to suit the problem and the scale of the environment as typically done in similar robotics problems. In this case, we adjusted the number of particles and the minimum score required for a scan so that the GMapping library considers it in its calculations. Additionally, we have also utilized a separate ROS node running in parallel to inject noise into the scans to introduce noise to GMapping and ensure the robustness of the model when using Gazebo.

The error metric is the difference between the predicted and actual orientation of the robot with respect to the pose at the beginning of the experiment.

$$\theta_0 = 0 \tag{4.1}$$

$$\theta_{error} = |\theta_{prediction} - \theta_{truth}| \tag{4.2}$$

where θ_0 is the starting orientation of the robot, $\theta_{prediction}$ is the orientation component of the predicted robot pose in GMapping, and θ_{truth} is the orientation component of the true robot pose retrieved from the Gazebo simulator.

4.4 Results

Error for the Unidimensional SLAM with GMapping was measured in each environment 5 times, and the averaged errors have been plotted below in Figure 4.2.

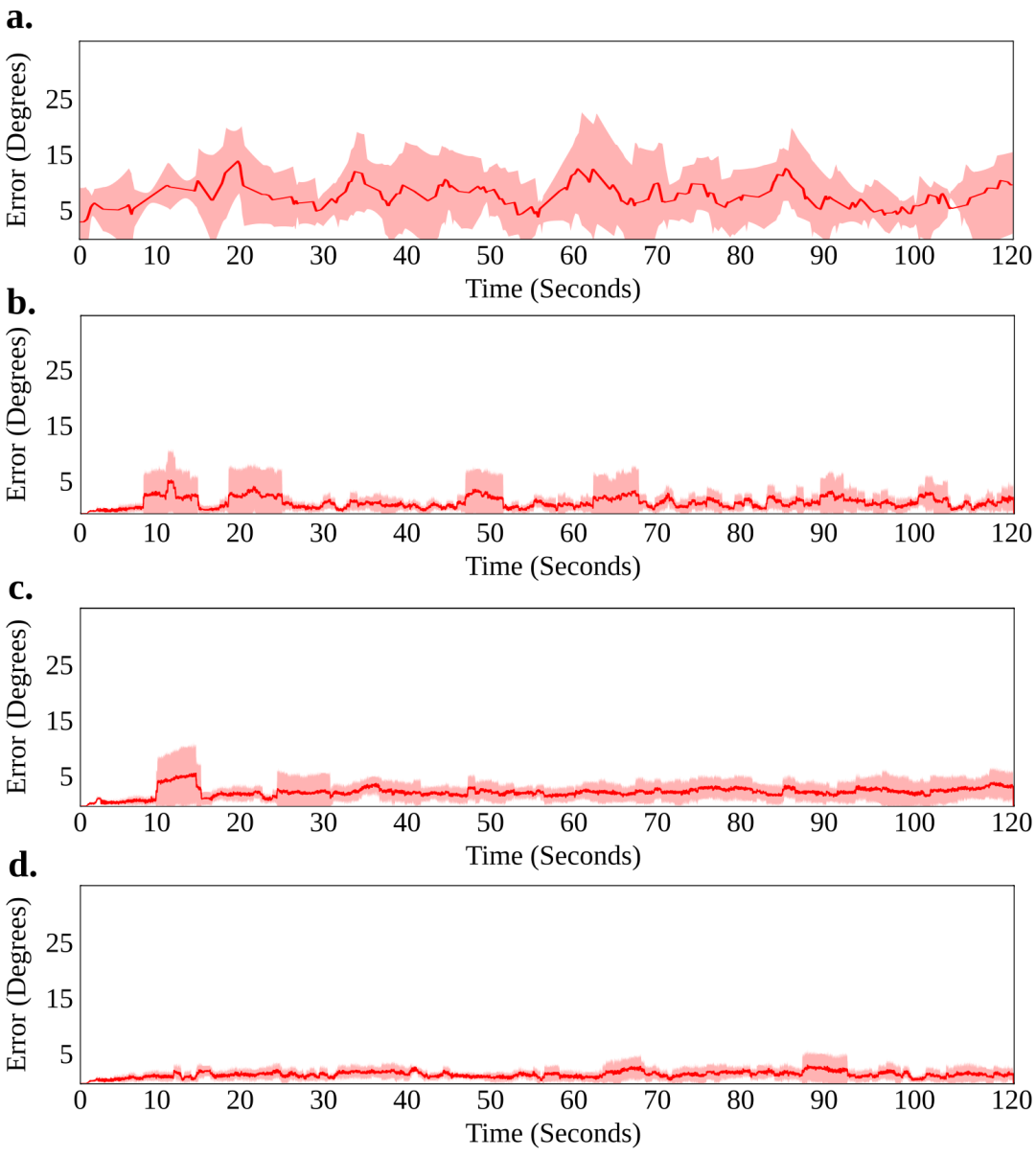

Figure 4.2: Errors in each environment

One thing to note is the level of error within each of the environments. Typically within the virtual environments (b - d), the average error is less than approximately 5 degrees. This can be attributed to the resolution of the cells within the SNN used to perform the SLAM problem on Loihi. The structure of the SNN is outlined within the paper by G. Tang, et al. [1]. In the description of the SNN, Tang describes how the SNN uses 75 neurons for one of the modules within the SNN. Each neuron was designed to span overlapping ranges of 5 degrees within the overall 360-degree space. If we were to add additional neurons and decrease the degrees each neuron spans over, we may see an additional increase in accuracy for localization.

The average power consumption measurement over multiple runs of the same experiments yielded the results in Figure 4.2. The power consumption was measured on an Intel i7-4850HQ CPU. The idle power consumption was measured by operating the machine with normal usage for multiple 10-minute intervals and using the mean of the measurements.

	Power Consumption
Idle power	4.47 W
While running GMapping	5.44 W
Dynamic power	0.97 W

Table 4.2: Average power consumption of GMapping on a CPU

Chapter 5

Discussion and Conclusion

Our research and development resulted in a Python library module extending the capabilities of Intel's NxSDK to enable incorporating astrocytes into SNNs. The use of astrocytes on Loihi introduces a method for spatial and temporal modulation in neural networks. Given the supposed role of astrocytes in learning and synaptic plasticity, their inclusion in SNNs on Loihi is very promising.

Although this was a significant step in the correct direction, we hope to improve the module further to make it even easier to use the library and make the model even more robust. Towards this end, we need to facilitate the ability to create many astrocytes with the same parameters and make it even easier to establish the tripartite synapses on Loihi.

With regards to the Unidimensional SLAM project, the measurements were critical to showing the viability of SNNs and Loihi in the paper by G. Tang, et al [1]. The overall results show the SNN achieved performance comparable to GMapping, a state-of-the-art solution for the problem while still consuming orders of magnitude less power in comparison. The next step in this line of research for us might be to explore how this can be applied to 2D SLAM rather than just the unidimensional SLAM problem we have focused on thus far using the head direction of the robot and restricting the motion to rotation.

Appendix A

Code

The code for creating a simple SNAN to observe synchronous activity from the astrocyte can be found below.

```
import os
import matplotlib as mpl
import sys
sys.path.append('../')
import matplotlib.pyplot as plt
import nxsdk.api.n2a as nx
from nxsdk.utils.plotutils import plotRaster
import combra_loihi.api as combra

import numpy as np

def gen_rand_spikes(num_neurons: int, sim_time: int,
                    firing_rate: float):
    """ Generate a random array of shape
    `(num_neurons, sim_time)` to specify spikes for input.

    :param num_neurons: The number of input neurons
    :param sim_time: Number of millisecond timesteps
    :param firing_rate: General firing rate in Hz
                        (i.e. 10 --> 10 Hz)
```

```
    :return: 2D array of binary spike values
    """
    random_spikes = np.random.rand(num_neurons, sim_time) <
                    (firing_rate / 1000.)
    random_spikes = [
        np.where(random_spikes[num, :])[0].tolist()
        for num in range(num_neurons)
    ]
    return random_spikes

np.random.seed(0)

net = nx.NxNet()
sim_time = 6000

pre_neuron_cnt = 10
post_neuron_cnt = 10

# Create pre-synaptic neuron (spike generator)
pre_synaptic_neurons = net.createSpikeGenProcess(
                            pre_neuron_cnt)
input_spike_times = gen_rand_spikes(pre_neuron_cnt,
                                    sim_time, 10)
pre_synaptic_neurons.addSpikes(
    spikeInputPortNodeIds=[
        num for num in range(pre_neuron_cnt)],
    spikeTimes=input_spike_times)

# Create post-synaptic neuron
post_neuron_proto = nx.CompartmentPrototype(
```

```
    vThMant=10,
    compartmentCurrentDecay=int(1/10*2**12),
    compartmentVoltageDecay=int(1/4*2**12),
    functionalState=nx.COMPARTMENT_FUNCTIONAL_STATE.IDLE)
post_neurons = net.createCompartmentGroup(
    size=post_neuron_cnt, prototype=post_neuron_proto)

# Create a connection from the pre to post-synaptic neuron
conn_proto = nx.ConnectionPrototype()
conn_mask = np.ones((10, 10))
weight = np.random.rand(10, 10) * 5
weight = weight * conn_mask
conn = pre_synaptic_neurons.connect(post_neurons,
                                    prototype=conn_proto,
                                    connectionMask=conn_mask,
                                    weight=weight)

# Create Astrocyte and establish connections
astrocyte = combra.Astrocyte(net)
astrocyte.connectInputNeurons(pre_synaptic_neurons,
                              pre_neuron_cnt,
                              weight=45)
astrocyte.connectOutputNeurons(post_neurons,
                               post_neuron_cnt,
                               weight=5)

# Create probes for plots
probes = dict()
probes['post_spikes'] = post_neurons.probe(
    [nx.ProbeParameter.SPIKE])[0]
```

```
probes['astro_sr_spikes'] = astrocyte.probe(
    combra.ASTRO_SPIKE_RECEIVER_PROBE.SPIKE)
probes['astro_ip3_voltage'] = astrocyte.probe(
    combra.ASTRO_IP3_INTEGRATOR_PROBE.COMPARTMENT_VOLTAGE)
probes['astro_sic_voltage'] = astrocyte.probe(
    combra.ASTRO_SIC_GENERATOR_PROBE.COMPARTMENT_VOLTAGE)
probes['astro_sg_spikes'] = astrocyte.probe(
    combra.ASTRO_SPIKE_GENERATOR_PROBE.SPIKE)

net.run(sim_time)
net.disconnect()

# Plots

fig = plt.figure(1, figsize=(18, 35))
ax0 = plt.subplot(7, 1, 1)
ax0.set_xlim(0, sim_time)
plotRaster(input_spike_times)
plt.ylabel('neuron index')
plt.xlabel('time (ms)')
plt.title('Presynaptic neurons poisson spikes')

ax1 = plt.subplot(7, 1, 2)
ax1.set_xlim(0, sim_time)
probes['astro_sr_spikes'].plot()
plt.xlabel('time (ms)')
plt.title('Astrocyte compartment 1: Spike receiver spikes')

ax2 = plt.subplot(7, 1, 3)
ax2.set_xlim(0, sim_time)
```

```
probes['astro_ip3_voltage'].plot()
plt.xlabel('time (ms)')
plt.title('IP3 integrator compartment voltage')

ax3 = plt.subplot(7, 1, 4)
ax3.set_xlim(0, sim_time)
probes['astro_sic_voltage'].plot()
plt.xlabel('time (ms)')
plt.title('SIC compartment voltage')

ax4 = plt.subplot(7, 1, 5)
ax4.set_xlim(0, sim_time)
probes['astro_sg_spikes'].plot()
plt.xlabel('time (ms)')
plt.title('Spike generator compartment spikes')

ax5 = plt.subplot(7, 1, 6)
ax5.set_xlim(0, sim_time)
probes['post_spikes'].plot()
plt.xlabel('time (ms)')
plt.title('Post-synaptic neuron spikes')

plt.tight_layout()
plt.show()
```

Bibliography

[1] Guangzhi Tang, Arpit Shah, and Konstantinos Michmizos. Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. 03 2019.

[2] Chit-Kwan Lin, Andreas Wild, Gautham N. Chinya, Yongqiang Cao, Mike Davies, Daniel M. Lavery, and Hong Wang. Programming spiking neural networks on intels loihi. *Computer*, 51(3):5261, 2018.

[3] Praveen Balachandar and Konstantinos P Michmizos. Neurobotics: A spiking neural network model of the oculomotor system for controlling a biomimetic robotic head. *Cognitive Computational Neuroscience*, Sep 2017.

[4] Cendra Agulhon, Jeremy Petravicz, Allison B. Mcmullen, Elizabeth J. Sweger, Suzanne K. Minton, Sarah R. Taves, Kristen B. Casper, Todd A. Fiacco, and Ken D. Mccarthy. What is the role of astrocyte calcium in neurophysiology? *Neuron*, 59(6):932946, 2008.

[5] Andrea Volterra, Nicolas Liaudet, and Iaroslav Savtchouk. Astrocyte ca2 signalling: an unexpected complexity, Apr 2014.

[6] Gertrudis Perea and Alfonso Araque. Glia modulates synaptic transmission. *Brain research reviews*, 63:93–102, 11 2009.

[7] Ioannis Polykretis, Vladimir Ivanov, and Konstantinos P. Michmizos. A neural-astrocytic network architecture: Astrocytic calcium waves modulate synchronous neuronal activity. In *Proceedings of the International Conference on Neuromorphic Systems*, ICONS '18, pages 6:1–6:8, New York, NY, USA, 2018. ACM.

[8] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta

Jain, and et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):8299, 2018.

[9] Kevin P. Murphy. Bayesian map learning in dynamic environments. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1015–1021, Cambridge, MA, USA, 1999. MIT Press.

[10] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Trans. Rob.*, 23(1):34–46, February 2007.

[11] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[12] Zhi Yan. Pose publisher. https://github.com/yzrobot/pose_publisher, 2017.