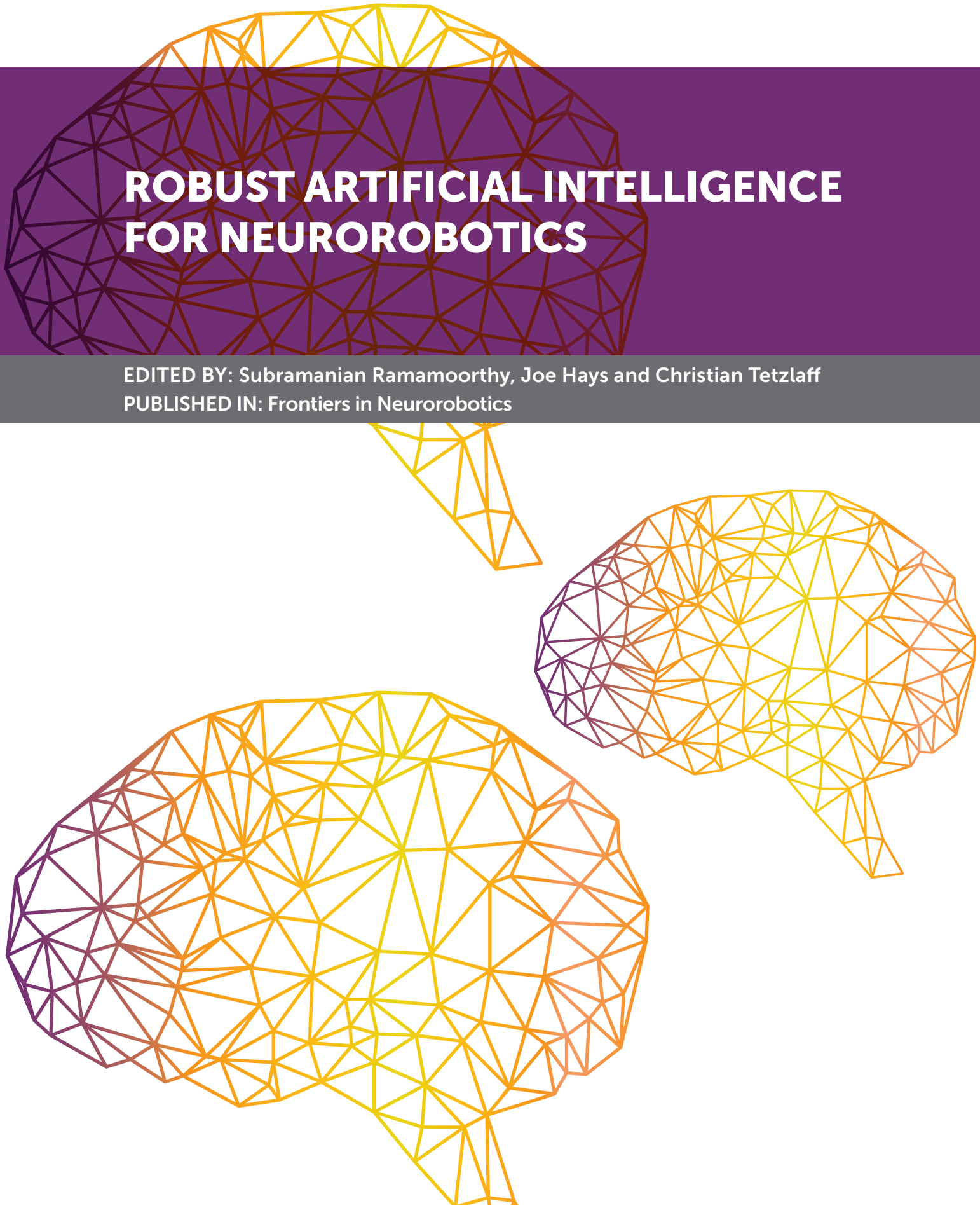
ROBUST ARTIFICIAL INTELLIGENCE FOR NEUROROBOTICS

EDITED BY: Subramanian Ramamoorthy, Joe Hays and Christian Tetzlaff
PUBLISHED IN: Frontiers in Neurorobotics



frontiers Research Topics

frontiers

ISSN 1664-8714
ISBN 978-2-88974-246-2
DOI 10.3389/978-2-88974-246-2

About Frontiers

Frontiers is more than just an open-access publisher of scholarly articles: it is a pioneering approach to the world of academia, radically improving the way scholarly research is managed. The grand vision of Frontiers is a world where all people have an equal opportunity to seek, share and generate knowledge. Frontiers provides immediate and permanent online open access to all its publications, but this alone is not enough to realize our grand goals.

Frontiers Journal Series

The Frontiers Journal Series is a multi-tier and interdisciplinary set of open-access, online journals, promising a paradigm shift from the current review, selection and dissemination processes in academic publishing. All Frontiers journals are driven by researchers for researchers; therefore, they constitute a service to the scholarly community. At the same time, the Frontiers Journal Series operates on a revolutionary invention, the tiered publishing system, initially addressing specific communities of scholars, and gradually climbing up to broader public understanding, thus serving the interests of the lay society, too.

Dedication to Quality

Each Frontiers article is a landmark of the highest quality, thanks to genuinely collaborative interactions between authors and review editors, who include some of the world's best academicians. Research must be certified by peers before entering a stream of knowledge that may eventually reach the public - and shape society; therefore, Frontiers only applies the most rigorous and unbiased reviews.
Frontiers revolutionizes research publishing by freely delivering the most outstanding research, evaluated with no bias from both the academic and social point of view. By applying the most advanced information technologies, Frontiers is catapulting scholarly publishing into a new generation.

What are Frontiers Research Topics?

Frontiers Research Topics are very popular trademarks of the Frontiers Journals Series: they are collections of at least ten articles, all centered on a particular subject. With their unique mix of varied contributions from Original Research to Review Articles, Frontiers Research Topics unify the most influential researchers, the latest key findings and historical advances in a hot research area! Find out more on how to host your own Frontiers Research Topic or contribute to one as an author by contacting the Frontiers Editorial Office: frontiersin.org/about/contact

ROBUST ARTIFICIAL INTELLIGENCE FOR NEUROROBOTICS

Topic Editors:
Subramanian Ramamoorthy, University of Edinburgh, United Kingdom
Joe Hays, United States Naval Research Laboratory, United States
Christian Tetzlaff, University of Göttingen, Germany

Citation: Ramamoorthy, S., Hays, J., Tetzlaff, C., eds. (2022).
Robust Artificial Intelligence for Neurorobotics. Lausanne: Frontiers Media SA.
doi: 10.3389/978-2-88974-246-2

Table of Contents

frontiers in Neurorobotics

EDITORIAL
published: 16 December 2021
doi: 10.3389/fnbot.2021.809903

Editorial: Robust Artificial Intelligence for Neurorobotics

Joe Hays[1], *Subramanian Ramamoorthy*[2*] *and Christian Tetzlaff*[3]

[1] *Robotics and Machine Learning Section, Dynamics and Control Systems Branch, Spacecraft Engineering Division, United States Naval Research Laboratory, Naval Center for Space Technology, Washington, DC, United States,* [2] *Institute of Perception, Action and Behaviour, School of Informatics, University of Edinburgh, Edinburgh, United Kingdom,* [3] *Bernstein Center for Computational Neuroscience, Third Institute of Physics, Georg-August-Universität Göttingen, Göttingen, Germany*

Keywords: robotics, adaptation, neuromorphic, artificial intelligence, autonomy

Editorial on the Research Topic

Robust Artificial Intelligence for Neurorobotics

INTRODUCTION

Neural computing is a powerful paradigm that has revolutionized machine learning. Building from early roots in the study of adaptive behavior and attempts to understand information processing in parallel and distributed neural architectures, modern neural networks have convincingly demonstrated successes in numerous areas—transforming the practice of computer vision, natural language processing, and even computational biology.

Applications in robotics bring stringent constraints on size, weight and power constraints (SWaP), which challenge the developers of these technologies in new ways. Indeed, these requirements take us back to the roots of the field of neural computing, forcing us to ask how it could be that the human brain achieves with as little as 12 watts of power what seems to require entire server farms with state of the art computational and numerical methods. Likewise, even lowly insects demonstrate a degree of adaptivity and resilience that still defy easy explanation or computational replication.

In this Research Topic, we have compiled the latest research addressing several aspects of these broadly defined challenge questions. As illustrated in **Figure 1**, the articles are organized into four prevailing themes: Sense, Think, Act, and Tools.

OVERVIEW

Sense

Three contributed articles focused primarily on the perception tasks of a robotic system. Specifically, Kirkland et al. investigated how to add understanding to the perception action cycle with spiking neural network (SNN) based segmentation. They demonstrated that an event based neuromorphic camera coupled with a Spiking fully Convolutional Neural Network (SpikeCNN) successfully provided semantic segmentation and understanding of their test scenes, and whose output was fed into a spiking control system providing actions.

Risi et al. presented a neuromorphic architecture for stereo vision, based on SNNs, that leverages the advantages of brain-inspired neuromorphic computing by interfacing two event-based vision sensors to an event-based mixed-signal analog/digital neuromorphic processor. Their results show a path toward the realization of low-latency, end-to-end event-based, neuromorphic architectures for stereo vision.

OPEN ACCESS

Edited and reviewed by:
Florian Röhrbein, Technische Universität Chemnitz, Germany

***Correspondence:**
Subramanian Ramamoorthy
s.ramamoorthy@ed.ac.uk

Received: *05 November 2021*
Accepted: *22 November 2021*
Published: *16 December 2021*

Citation:
Hays J, Ramamoorthy S and Tetzlaff C (2021) Editorial: Robust Artificial Intelligence for Neurorobotics. Front. Neurorobot. 15:809903. doi: 10.3389/fnbot.2021.809903

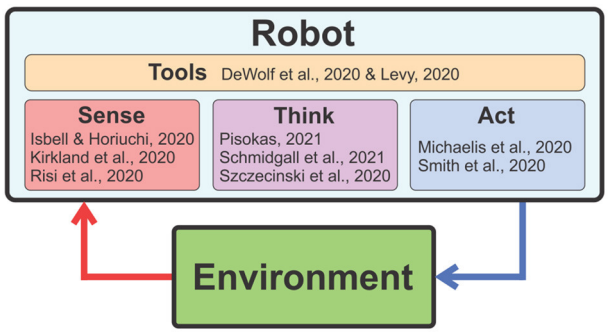


FIGURE 1 | The articles of this Research Topic are organized into the following four themes: Sense, Think, Act, and Tools.

Finally, Isbell and Horiuchi presented the concept of Echo View Cells which used bat-inspired sonar to mimic how bats might sense objects in the environment and recognize the views associated with different places. They successfully demonstrated spatial invariance by training feed-forward neural networks, both traditional artificial neural networks (ANNs) and SNNs, to recognize 66 distinct places in a laboratory environment over a limited range of translations and rotations.

Think

Three articles focused on general approaches to algorithm development based on Spiking Neural Networks, representing the thinking aspects of systems. Pisokas presented the perspective that the combination of reverse engineering with simulations allows the study of both the structure and function of biological neural circuits. This approach augmented understanding of both the computation performed by the neuronal circuit and the role of its components. Thus, a robotics practitioner can gain added inspiration and guidance in the development of network-based robotic algorithms.

Szczecinski et al. extend their previously-developed method for tuning ANNs, the "Functional Subnetwork Approach," to SNNs based on generalized linear integrate-and-fire neurons. This extension enabled specific functions to be realized in SNNs through a direct analytic method of assembling and tuning the networks without the use of global optimization, or other forms of machine learning. Robotic algorithm developers are now able to directly implement specific functions in a network without training.

Finally, Schmidgall et al. introduced a framework, SpikePropamine, for simultaneously learning the underlying fixed-weights, and the rules governing the dynamics of synaptic plasticity and neuro-modulated synaptic plasticity in SNNs through gradient descent. This offers practitioners an approach to leverage the strengths of online plasticity in their robotic algorithm development through methods such as reinforcement learning.

Act

Two articles addressed the action, or motor output, aspects of an embodied robotic system. Michaelis et al. presented a network architecture including the anisotropic network and a pooling layer which allows fast spike read-out from a neuromorphic processor (Intel's research chip Loihi) and performs an inherent regularization. With this, they showed that the anisotropic network reliably encoded sequential patterns of neural activity, each representing a robotic action, and that the patterns allowed the generation of multidimensional trajectories on control-relevant timescales.

Additionally, Smith et al. present latest results from their framework for self-organizing robotic sensorimotor control, DIAMOND, which employs a deep recurrent neural network, based on the principles of predictive coding. Their results provide evidence that deeper networks enable more complex exploratory behaviors.

Tools

From a development tool perspective, Levy presented a minimalist application programming interface (API) for sensors and PID controllers, which makes it relatively easy for engineers to prototype neuromorphic approaches to micro-air-vehicle sensing and navigation.

Additionally, DeWolf et al. demonstrated how the Nengo neural modeling and simulation libraries enable users to quickly develop robotic perception and action neural networks for simulation on neuromorphic hardware. Hereby, users can rely on tools they are already familiar with, such as Keras and Python.

OUTLOOK FOR THE FUTURE

A long standing open question at the intersection of many fields—Artificial Intelligence, Neural Computing, Neuromorphic Systems and other forms of Biomimesis—pertains to the specification of how artificial systems should emulate the natural phenomena around learning and adaptation, and what the construction of such artificial systems might tell us about nature itself. Papers in this volume explore exactly this interface. With rapid advances both in our understanding of models (natural and artificial) and in our ability to fabricate new devices, the gaps between these diverse methodologies are rapidly closing, potentially enabling entirely new ways of answering these long-standing questions.

AUTHOR CONTRIBUTIONS

All authors reviewed multiple contributed articles in this research task and collaboratively wrote this Editorial.

ACKNOWLEDGMENTS

We thank all authors contributing their work to this Research Topic. JH acknowledges support from the US Naval Research Laboratory's Base Program Safe Lifelong Motor Learning (WU1R36). SR acknowledges support from the US Office of Naval Research–Global (award no. N62909-19-1-2072) for

the organization of the Robust Artificial Intelligence for Neurorobotics Workshop. CT acknowledges support by the H2020 projects Plan4Act (#732266), ADOPD (#899265), and by the Intel Corporation via a financial gift without restrictions.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The Intel Corporation did not influence the current work nor had any role in it.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 16 March 2020
doi: 10.3389/fnbot.2020.00016

Robustness Through Simplicity: A Minimalist Gateway to Neurorobotic Flight

*Simon D. Levy**

Computer Science Department, Washington and Lee University, Lexington, VA, United States

In attempting to build neurorobotic systems based on flying animals, engineers have come to rely on existing firmware and simulation tools designed for miniature aerial vehicles (MAVs). Although they provide a valuable platform for the collection of data for Deep Learning and related AI approaches, such tools are deliberately designed to be general (supporting air, ground, and water vehicles) and feature-rich. The sheer amount of code required to support such broad capabilities can make it a daunting task to adapt these tools to building neurorobotic systems for flight. In this paper we present a complementary pair of simple, object-oriented software tools (multirotor flight-control firmware and simulation platform), each consisting of a core of a few thousand lines of C++ code, that we offer as a candidate solution to this challenge. By providing a minimalist application programming interface (API) for sensors and PID controllers, our software tools make it relatively painless for engineers to prototype neuromorphic approaches to MAV sensing and navigation. We conclude our discussion by presenting a simple PID controller we built using the popular Nengo neural simulator in conjunction with our flight-simulation platform.

Keywords: drones, miniature aerial vehicles, spiking neural network, PID control, flight simulator

OPEN ACCESS

Edited by:
Subramanian Ramamoorthy, University of Edinburgh, United Kingdom

Reviewed by:
Terrence C. Stewart, University of Waterloo, Canada
Sebastian Scott James, University of Sheffield, United Kingdom

***Correspondence:**
Simon D. Levy
simon.d.levy@gmail.com

Received: *26 November 2019*
Accepted: *27 February 2020*
Published: *16 March 2020*

Citation:
Levy SD (2020) Robustness Through Simplicity: A Minimalist Gateway to Neurorobotic Flight. Front. Neurorobot. 14:16. doi: 10.3389/fnbot.2020.00016

1. INTRODUCTION

Beginning with J.J. Gibson's pioneering research on visual perception (Gibson, 1979), decades of research in behavioral neuroscience have shown the importance of robust, tightly-coupled perception/action cycles in supporting successful movement (predation, obstacle avoidance) in challenging environments. This is especially true for flying animals like birds and insects, whose survival depends on overcoming of a variety of forces in three-dimensional space; most obviously, gravity (Floreano et al., 2009).

In attempting to build neurorobotic systems based on flying animals, engineers have come to rely on existing firmware and simulation tools designed for miniature aerial vehicles (MAVs). Although they provide a valuable platform for quick entrée into the world of first-person-view (FPV) racing or aerial photography (firmware), and the collection of data for Deep Learning and related AI approaches (simulation), such tools are deliberately designed to be as feature-rich and general as possible, to appeal to the widest audience. The most popular software tools support air, ground, and water vehicles and provide a hierarchy of safety mechanisms for minimizing the likelihood of injury and property damage. Unsurprisingly, the sheer amount of code required to support such broad capabilities can make it a daunting task to adapt these tools to building neurorobotic systems for flight.

In the remainder of this paper we present a pair of simple, object-oriented software tools—*Hackflight* and *MulticopterSim*—each consisting of a core of a few thousand lines of C++ code, that we offer as a candidate solution to this challenge. These software tools are built on the popular Arduino microcontroller platform and the popular video game platform Unreal Engine 4. By providing a minimalist application programming interface (API) for sensors and PID controllers, these tools make it relatively painless for engineers to prototype neuromorphic approaches to MAV sensing and navigation.

2. HACKFLIGHT

Hackflight is an open-source toolkit for building multirotor flight-control firmware and software. The project began in 2015 as an attempt by the author to build a simple open-source flight-control firmware program for MAVs using the Arduino platform (Banzi and Shiloh, 2014). At that time, as well as today, there were two major firmware projects for MAVs: ArduPilot (ArduPilot Dev Team, 2019a) and Cleanflight (Cleanflight Team, 2019). ArduPilot focuses on sophisticated mission planning with waypoint navigation and other features, and runs mainly on the Pixhawk flight controller. Cleanflight and its derivatives (Betaflight, Raceflight) are popular with FPV racing enthusiasts, and run on a broad variety of flight-control boards designed for FPV racing. (A more recent Cleanflight derivative, iNav, adds features for navigation and for fixed-wing aircraft). Although both projects can trace their origin to the Arduino platform, they have long since switched to using their own non-Arduino hardware drivers for sensing and motor control. Both projects are supported by large development teams and have a code base of several hundred thousand lines (see **Table 1**). Hackflight, by contrast, uses approximately 4,500 lines[1].

How can Hackflight get away with using to or three orders of magnitude less code than the two most popular flight-control firmware packages? As discussed in the sections below, we attribute this difference to a few important design principles: (1) limitation to multirotor vehicles, not fixed-wing or ground vehicles; (2) targeting programmers instead of general users; (3) Arduino compatibility; (4) simple object-oriented API.

2.1. Features

Unlike ArduPilot, which supports a variety of vehicle types (multirotors, fixed-wing aircraft, ground vehicles, marine vehicles), Hackflight supports only multirotors. Cleanflight and its derivatives, while supporting mainly multirotors (and perhaps fixed-wing aircraft), offer a variety of configuration features and flight modes (PID controllers), allowing everyone from beginners to professional racing pilots to use them. Hackflight, by contrast, uses only a the bare minimum of PID controllers necessary for stable flight, allowing you to create your own PID controllers with relative ease (see section 2.4 below).

TABLE 1 | Approximate size of flight-control firmware packages.

Package	Lines of code
Cleanflight	851,659
ArduPilot	283,316
Hackflight	4,445

2.2. Audience

Although both ArduPilot and Cleanflight are open-source, their target users are mostly non-programmers. There is therefore a heavy focus in both projects on GUI-based configurator programs. Hackflight, by contrast, is targeted toward engineers and researchers comfortable with coding in C++. Adding a feature to Hackflight therefore requires significantly less code support, enabling rapid prototyping of new sensors, PID, controllers, etc.

2.3. Arduino Compatibility

As mentioned above, Hackflight began as the author's attempt to build a simple open-source flight-control program using the Arduino software libraries. Although Hackflight now supports a subset of the STM32F3/4 flight controllers supported by Cleanflight and its derivatives, our focus has always been on Arduino compatibility. Thanks to the recent availability of small, fast, 32-bit microcontroller development boards like Teensy and the STM32L4 line from Tlera Corporation[2], Arduino compatibility is no longer tied to slower, eight-bit boards lacking floating-point support (see **Figure 1**). Arduino compatibility means that Hackflight can quickly exploit the increasing variety of new sensors available today, without the need to write a custom driver. Although the variety of neuromorphic sensors currently available cannot rival the variety of Arduino-compatible MEMS sensors (inertial measurement units, proximity sensors, and the like), we are optimistic that neuromorphic devices will follow the same trajectory; i.e., they will provide a UART or other low-level serial interface for working with Arduino and similar development platforms.

2.4. Simple Object-Oriented API

Hackflight is written entirely in C++, with the core components written in header-only style. Our focus is on object-oriented design, with most classes (altitude PID control, distance sensing) being subclasses of other, more abstract classes (PID controller, sensor). In addition to enabling extensive code re-use, this approach allows us to abstract the driver code for a component (sensor, motor) from the algorithms using that component (Madgwick quaternion filter, mixer). This clean separation allows Hackflight to be "dropped" directly into a simulation environment (through the use of C++ `#include` statements), without the need for "Hardware-In-the-Loop" (HIL), socket connections, or other indirect mechanisms (see section 3 below).

[1]To estimate the number of lines of code in each package, we cloned the package repository from github, ran the **cloc** program (https://github.com/AlDanial/cloc) in the root directory of the repository, and summed over the reported number of lines in C/C++ header files, C files, and C++ files. For reference, the respective git commits were: Hackflight: `206a6dd`; Cleanflight: `83ed5df`; Ardupilot: `87a5189`.

[2]https://www.tindie.com/stores/TleraCorp/

Although both ArduPilot and Cleanflight separate the driver code from the algorithmic code, Hackflight's consistent use of object-oriented design allows us to avoid pre-processor macros (`#ifdef ... #else ... #endif`) that are used extensively in those two packages and can make it difficult to arrive at a basic understanding of much of the code.

As well as keeping the codebase small, simple, and portable, these design principles support a more direct connection between the mathematical theory underlying flight control and its implementation in code. **Figure 2** illustrates this point by showing the main loop in Hackflight. In the figure, each box (demands, state) represents a simple datatype in the C++ code, and each oval (R/C Receiver, Sensors, PID controllers, Mixer) represents an abstract class. Mathematically, then, each abstract class is a function from one datatype to another: $\mathsf{Sensor} : \mathsf{State} \mapsto \mathsf{State}$; $\mathsf{PIDController} : (\mathsf{State} \times \mathsf{Demands}) \mapsto \mathsf{Demands}$. We believe that this design principle makes Hackflight both easy to understand and simple to adapt.

Figure 3 illustrates these principles by showing a complete Arduino firmware sketch (main program) for a quadcopter using the flight controller in **Figure 1**. As the sketch shows, Hackflight's simple API supports programs in which only the required components (microcontroller, IMU, receiver, PID controllers, mixer, motors) need to be specified (as opposed to choosing from a list of options with a control statement). This approach results in example programs that are easy for a programmer to read and to adapt for use with new sensors, vehicle designs and control paradigms.

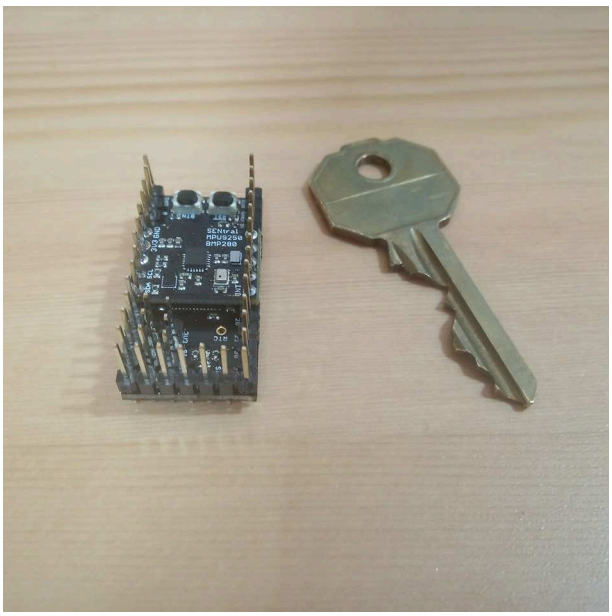


FIGURE 1 | Arduino-compatible flight controller for Hackflight (total cost ~$55 U.S.).

3. MULTICOPTERSIM

Like Hackflight, MulticopterSim is designed as a minimalist solution to a difficult engineering problem; in this case, a physically realistic multirotor simulator general enough to interface with a variety of flight-control packages. As with similar efforts by others who have attempted to use a general-purpose robotics simulator like Gazebo (Koenig and Howard, 2004), our simulator began as a plugin for a more general robotics simulation platform, V-REP (Rohmer et al., 2013). The lack of realistic simulated camera images in these packages led us to a photo-realistic game engine, UnrealEngine4 (Sanders, 2016). Because UE4 is also used by Microsoft's popular AirSim (Shah et al., 2017) drone simulator, AirSim provides a useful frame-of-reference for MulticopterSim[3].

In addition to its focus on Deep Learning, AirSim has since expanded to include support for self-driving cars, and provides Python APIs for remote operation of the vehicles. As with flight-control firmware discussed in the previous section, this rich set of features translates into significantly more code. **Table 2** shows the relative sizes of AirSim and MulticopterSim, based on the same metric used in **Table 1**. As we saw with Hackflight, the design principles used in MulticopterSim help keep the codebase small, manageable, and easily extendable.

The core of MulticopterSim is the abstract C++ *FlightManager* class. This class provides support for running the vehicle dynamics and the PID control regime (e.g., Hackflight) on its own thread, after it first disables the built-in physics in UE4. The dynamics we used are based directly on the model presented in Bouabdallah et al. (2004), written as a standalone, header-only C++ class that can be easily adapted for other simulators and applications if desired. This class also supports different frame configurations (quadcopter, hexacopter) via virtual methods. By running the FlightManager on its own thread, we are able to achieve arbitrarily fast updates of the dynamics and flight-control. We currently limit the update rate to 1kHz, based on the data output rate of current MEMS gyrometers. It would also be possible to run the dynamics and control on separate threads, though we have not yet found it advantageous to do that.

The FlightManager API contains a single virtual method, `update()`, which accepts the current time and the state of the vehicle (as computed by the dynamics), and returns the current motor values. The motor values are then passed to the dynamics object, which computes the new vehicle state. On the main thread, UE4's `Tick()` method queries the flight manager for the current vehicle pose (location, rotation) and displays the vehicle and its environment kinematically at the 60–120 Hz frame rate of the game engine. In a similar manner, the threaded *VideoManager* classes can be used to process the images collected by a simulated gimbal-mounted camera on the vehicle, using OpenCV (Bradski, 2000). An abstract C++ *Target* class supports

[3]In March of 2017 the head of Microsoft's AirSim project contacted the author about using Hackflight as the flight-control software for AirSim, citing the design principles of Hackflight is the primary reason for this interest. After a licensing incompatibility ended up making this collaboration unfeasible, the author turned to developing quadcopter flight simulator from scratch, using UE4 and the Hackflight firmware.

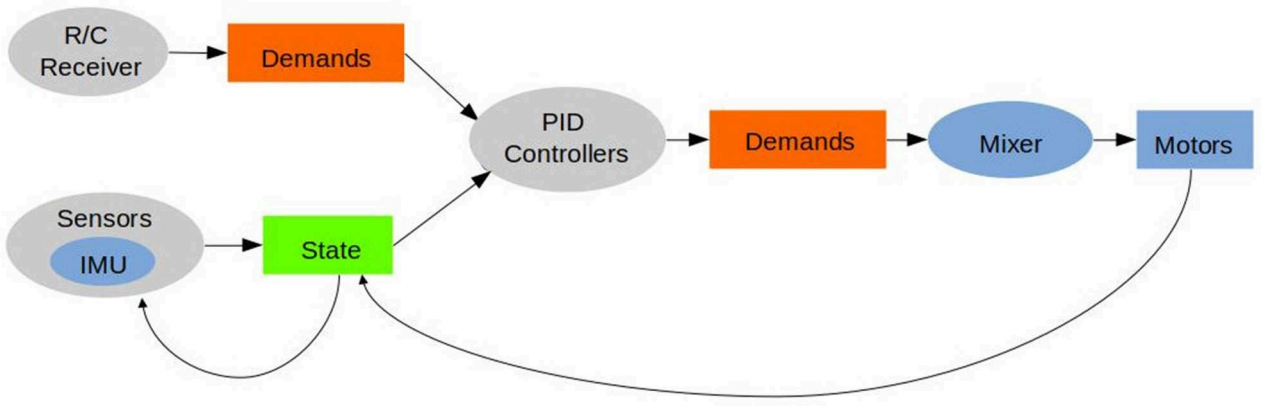


FIGURE 2 | Hackflight main loop.

```
hf::Hackflight h;
hf::USFS imu;
hf::DSMX_Receiver_Serial1 rc =
    hf::DSMX_Receiver_Serial1(CHANNEL_MAP, DEMAND_SCALE);
hf::MixerQuadXCF mixer;
hf::RatePid ratePid = hf::RatePid( 0.05f, 0.00f, 0.00f, 0.10f, 0.01f);
hf::LevelPid levelPid = hf::LevelPid(0.20f);

hf::StandardMotor motor1(5), motor2(8), motor3(9), motor4(11);
hf::Motor * motors[4] = { &motor1, &motor2, &motor3, &motor4 };

void setup(void)
{
    h.init(new hf::Butterfly(), &imu, &rc, &mixer, motors);

    h.addPidController(&levelPid);
    h.addPidController(&ratePid);
}

void loop(void)
{
    h.update();
}
```

FIGURE 3 | Sample Hackflight sketch for Arduino.

modeling interaction with other moving objects having their own dynamics; for example, in a predator/prey scenario.

This simplicity of our flight-control scheme makes it easy to connect MulticopterSim to existing flight-control software like Hackflight, or to the Software-in-the-Loop (SITL) mechanism of ArduPilot (ArduPilot Dev Team, 2019b), as modules in the MulticopterSim codebase. With the Hackflight module, for example, we treat the control device (e.g., joystick, Xbox game controller) as "virtual receiver," which provides the R/C Receiver signal shown at the top of **Figure 2**. Further, the abstraction provided by Hackflight for sensing and open-loop control allows rapid prototyping of hybrid control systems, as we describe in the next section.

TABLE 2 | Approximate sizes of two flight-simulation packages[a].

Package	Lines of code
AirSim	77,600
MulticopterSim	2,266

[a] *The line count for MulticopterSim includes the module for Hackflight (see main text for details). For reference, the respective git commits were MulticopterSim:* `aec0ae8`*; AirSim:* `ca29068`.

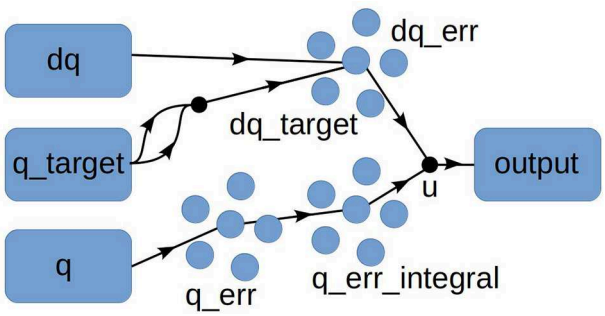


FIGURE 4 | Nengo model for simple PID control.

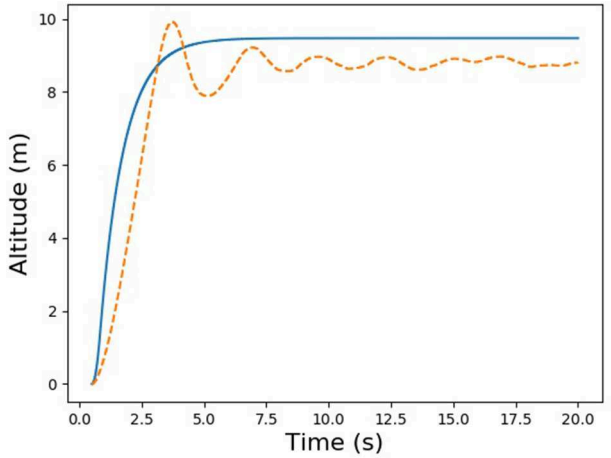


FIGURE 5 | Comparison of traditional (solid line) and neural (dashed line) PID controllers.

4. TOWARD NEUROMORPHIC FLIGHT CONTROL

As a demonstration of our approach, we used the Python-based Nengo neural simulator (Bekolay et al., 2014) to create a simple PID controller class for altitude hold. As shown in **Figure 4**, the controller consists of three populations of 200 spiking neurons: one population for computing the error between the target altitude and current altitude (*P* term); one for integrating the error (*I* term), and one for computing the error derivative *D*) term. (For this simple experiment we used only *P*.) The constants K_P, K_I, and K_d are implemented as arguments to the `transform` parameter of the `nengo.Connection` constructor; i.e., as connection weights between pools of neurons. We set the simulation time step to 0.001 s[4] and used the default values for the remaining parameters in the Nengo class constructors. We made this Python class available to MulticopterSim by adding a UDP client/server module to MulticopterSim: the PID control code runs in Python as a server, and the C++ code for the simulator acts as a client for this server, sending the vehicle state to the server and getting back motor commands to fly the vehicle.

For this trial experiment, we chose a simple PID control task common to flight-control systems like ArduPilot, namely, takeoff to a fixed altitude. We wrote two versions of the same basic Python server script. One version used the ArduPilot algorithm for altitude hold, with the error between the target and actual altitudes as a set-point for a secondary, velocity-based PID controller. The other version used the Nengo-based PID controller shown in **Figure 4**. Sample results for this experiments are provided in **Figure 5**. As the figure shows, the Nengo-based control compares favorably to the algorithm that computes the PID control signal in the traditional way, albeit with some oscillation and greater undershoot. Although this Nengo-based PID controller has been hand-tuned by us to work with our simulator, and could obviously use some improvement, it provides a simple proof of the feasibility of using an advanced neural simulator like Nengo with a real-time flight simulator, paving the way for more interesting experiments.

5. CONCLUSION AND FUTURE WORK

As the closest robotic approximation to flying insects, birds, and mammals, miniature aerial vehicles (MAVs) offer a compelling new platform for research in neuromorphic sensing, notably in the realm of vision (Mitrokhin et al., 2019). Such research faces unique challenges.

In the physical realm, the current weight and form factor of commercially-available event-based dynamic vision sensor (DVS) devices makes them impractical for deployment on micro-scale aerial vehicles. We are currently experimenting with our recently-purchased DAVIS346 sensor H (40 × 60 × 25 mm, 100 g), using a RaspberryPi to convert the sensor's USB3 signal to UART (TTL) format for consumption by an Arduino-compatible microcontroller. If that arrangement proves successful, we will look into acquiring the much smaller mini-eDVS unit (18 × 18 × 7 mm, 3 g), from the same manufacturer[5].

In simulation, the 60–120 Hz frame rate of game engines like UE4 and Unity (Menard, 2011) exceeds that of most commercially-available CMOS cameras but is inadequate for

[4] We chose this value as an order-of-magnitude approximation to the data output rate (DOR) of contemporary inertial measurements units. As one reviewer pointed out, it would also be useful to know how close to real-time such a model runs on the sort of standard CPU hardware that is available on a quadcopter (see future work section below).

[5] We thank a reviewer for suggesting the mini-eDVS, which was not available for purchase at the time of this writing.

emulating the multi-kilohertz data rates enabled by DVS (Gallego et al., 2019). Hence, one of our current research directions involves modeling the DVS datastream directly from the dynamics of the vehicle and target object.

To extend our Nengo-based PID controller in a more biologically realistic direction, we are also experimenting with a Python version of our multirotor dynamics code, to exploit Nengo's support for reinforcement learning (Bekolay and Eliasmith, 2011). This paradigm would provide an accelerated way to develop neuromorphic flight controllers in an abstract mathematical simulation, to be validated by transferring them to MulitCopterSim, and eventually to an actual vehicle.

Finally, our Python-based client/server moduel will make it significantly easier to experiment with other neural simulators offering a Python API, including Brian (Stimberg et al., 2019) and NEURON (Hines and Carnevale, 2013).

For both real and simulated flying robots, we see our minimalist, integrated approach to software and firmware design as a promising direction for robust aerial neurorobotics.

6. DOWNLOADS

The software described in this paper can be downloaded from the following repositories:

- https://github.com/simondlevy/Hackflight
- https://github.com/simondlevy/MulticopterSim
- https://github.com/simondlevy/MulticopterSim/tree/NengoModule
- https://github.com/simondlevy/gym-copter.

DATA AVAILABILITY STATEMENT

All datasets generated for this study are included in the article/supplementary material.

AUTHOR CONTRIBUTIONS

The author confirms being the sole contributor of this work and has approved it for publication.

ACKNOWLEDGMENTS

We thank Terry Stewart for help with the Nengo PID controller, Shital Shah for the header-only rewrite of Hackflight, and two reviewers for helpful suggestions. This research was supported by winter 2019 sabbatical-leave funding from Washington and Lee University.

REFERENCES

ArduPilot Dev Team (2019a). *History of Ardupilot*. Available online at: http://ardupilot.org/planner2/docs/common-history-of-ardupilot.html (accessed June 15, 2019).

ArduPilot Dev Team (2019b). *Sitl Simulator (Software in the Loop)*. Available online at: http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html (accessed June 17, 2019).

Banzi, M., and Shiloh, M. (2014). *Getting Started With Arduino*. Sebastopol, CA: Maker Media.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., etal. (2014). Nengo: a python tool for building large-scale functional brain models. *Front. Neuroinform.* 7:48. doi: 10.3389/fninf.2013.00048

Bekolay, T., and Eliasmith, C. (2011). "A general error-modulated stdp learning rule applied to reinforcement learning in the basal ganglia," in *Proceedings of the Conference on Computational Systems Neuroscience (COSYNE)* (Salt Lake City, UT), 24–27.

Bouabdallah, S., Murrieri, P., and Siegwart, R. (2004). "Design and control of an indoor micro quadrotor," in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation, ICRA 2004, April 26 - May 1, 2004* (New Orleans, LA), 4393–4398. doi: 10.1109/ROBOT.2004.1302409

Bradski, G. (2000). *The OpenCV Library*. Dr. Dobb's Journal of Software Tools.

Cleanflight Team (2019). Available online at: http://cleanflight.com (accessed June 15, 2019).

Floreano, D., Zufferey, J.-C., Srinivasan, M. V., and Ellington, C. (2009). *Flying Insects and Robots, 1st Edn*. New York, NY: Springer Publishing Company, Incorporated. doi: 10.1007/978-3-540-89393-6

Gallego, G., Delbrück, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2019). Event-based vision: a survey. *CoRR*, abs/1904.08405.

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Boston, MA: Houghton Mifflin.

Hines, M., and Carnevale, T. (2013). *NEURON Simulation Environment*. New York, NY: Springer.

Koenig, N., and Howard, A. (2004). "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems* (Sendai), 2149–2154.

Menard, M. (2011). *Game Development with Unity, 1st Edn*. Boston, MA: Course Technology Press.

Mitrokhin, A., Sutor, P., Fermüller, C., and Aloimonos, Y. (2019). Learning sensorimotor control with neuromorphic sensors: toward hyperdimensional active perception. *Sci. Robot.* 4:eaaw6736. doi: 10.1126/scirobotics.aaw6736

Rohmer, E., Singh, S. P. N., and Freese, M. (2013). "V-rep: a versatile and scalable robot simulation framework," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*.

Sanders, A. (2016). *An Introduction to Unreal Engine 4*. Natick, MA: A. K. Peters, Ltd.

Shah, S., Dey, D., Lovett, C., and Kapoor, A. (2017). "Airsim: high-fidelity visual and physical simulation for autonomous vehicles," in *Proceedings of the 11th Conference on Field and Service Robotics (FSR 2017)* (Zurich).

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8:e47314. doi: 10.7554/eLife.47314

Conflict of Interest: The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

BRIEF RESEARCH REPORT
published: 15 September 2020
doi: 10.3389/fnbot.2020.00062

The DIAMOND Model: Deep Recurrent Neural Networks for Self-Organizing Robot Control

Simón C. Smith[1], *Richard Dharmadi*[1], *Calum Imrie*[1], *Bailu Si*[2,3] *and J. Michael Herrmann*[1,2*]

[1] Institute of Perception, Action and Behaviour (IPAB), School of Informatics, University of Edinburgh, Edinburgh, United Kingdom, [2] State Key Laboratory of Robotics, Shenyang Institute of Automation, Institutes for Robotics and Intelligent Manufacturing, Chinese Academy of Sciences, Shenyang, China, [3] School of Systems Science, Beijing Normal University, Beijing, China

The proposed architecture applies the principle of predictive coding and deep learning in a brain-inspired approach to robotic sensorimotor control. It is composed of many layers each of which is a recurrent network. The component networks can be spontaneously active due to the homeokinetic learning rule, a principle that has been studied previously for the purpose of self-organized generation of behavior. We present robotic simulations that illustrate the function of the network and show evidence that deeper networks enable more complex exploratory behavior.

Keywords: deep neural networks, autonomous learning, homeokinesis, self-organizing control, robot control

1. INTRODUCTION

Deep neural architectures (Fukushima and Miyake, 1980; Hinton et al., 2006) have reached a level comparable to human performance in certain pattern recognition tasks (Krizhevsky et al., 2012). Also in robotic applications, deep networks gain more and more importance, from state abstraction to seamless end-to-end control in complex repetitive tasks (Levine et al., 2016). Moreover, it has been speculated whether deep feed-forward networks can account for some aspects of information processing in the mammalian visual system (Serre et al., 2007), which is not to say that the brain *is* nothing but a collection of deep neural networks. Quite to the contrary, the brain is known to have dynamical properties that are much richer than standard deep architectures:

- Biological neural systems consist of patches of interconnected neurons which also receive re-entrant connectivity via other patches.
- Spontaneous behavior can occur at any level of depth and may spread in either direction.
- Sensory inputs are not only providing information for decision about actions, but are also analyzed for effects of previous actions.
- A hierarchical organization enables lateral transferability and flexible compositionality.
- There is little use for supervised learning.

Based on these considerations, we propose here an architecture that combines the undeniable strengths of deep neural networks with *homeokinesis* (Der, 2001), an approach to meet requirements of autonomous robots (see section 2). Our work connects to (Carvalho and Nolfi, 2016) where the introduction of flexibility and plasticity in a neural controller showed a good effect in a cleaning task, however, mainly based on an evolutionary approach, whereas we aim at a more principled architecture that achieves an increased flexibility by a hierarchy of identical controllers. The autonomously generate activity of higher-lever controllers provide an

OPEN ACCESS

Edited by:
Christian Tetzlaff,
University of Göttingen, Germany

Reviewed by:
Luca Patanè,
University of Catania, Italy
Yinyan Zhang,
Jinan University, China

***Correspondence:**
J. Michael Herrmann
michael.herrmann@ed.ac.uk

Received: *31 May 2020*
Accepted: *03 August 2020*
Published: *15 September 2020*

Citation:
Smith SC, Dharmadi R, Imrie C, Si B and Herrmann JM (2020) The DIAMOND Model: Deep Recurrent Neural Networks for Self-Organizing Robot Control. Front. Neurorobot. 14:62. doi: 10.3389/fnbot.2020.00062

intrinsic motivation (Oudeyer et al., 2007) for the lower ones. In this way, we are able to propose a more brain-like architecture which implicitly realizes a predictive coding principle, compare (Adams et al., 2013) for a related approach, at least in some parameter range, as discussed below. An early interesting comparison is provided by (Rusu et al., 2003) which presents a neuro-fuzzy controller for determining the behavior of a robot in a navigation task. Their architecture had a similarly layered structure, although the behaviors had to be predefined at a time when homeokinesis (Der, 2001) was just being developed. More recently, differential Hebbian learning was used to explore possible behaviors of a robot (Pinneri and Martius, 2018), presenting a more brain-like approach at the low level, whereas we aim a model that captures characteristics of the area-level organization of the brain.

In the following, we will consider first the homeokinetically controlled sensorimotor loop (Der, 2001) as the basic element of the proposed system (section 2). In this way, we incorporate a source of spontaneous activity. The composition of these elements in the DIAMOND (Deep Integrated Architecture for sensoriMotor self-Organization aNd Deliberation) architecture (section 3) will thus be able to generate activity at all levels and work in a fully self-supervised way, although it is also possible to steer the system to desired behavior by very small guiding inputs (Martius and Herrmann, 2011). The main layout of the architecture includes a basic layer that receives information from outside world and sends actions and is expected to represent low-level features. There is a variable number of deeper layers that interact only with the neighboring layers and which represent more abstract features that are extracted from the data through the lower layers. The architecture learns by the homeokinetic learning rule (see below) which implies that consistency between neighboring layers is required. We will present a few experimental results in section 4, and discuss the realism and performance of the architecture as well as further work in section 5.

2. HOMEOKINETIC CONTROL

The basic element of our architecture is formed by a homeokinetic controller, which we will describe here only briefly, details can be found in (Der and Martius, 2012). This unsupervised active learning control algorithm shapes the interaction between a robot and its environment by updating the parameters of a controller and of an internal model. The learning goal can be characterized as a balance of predictability and sensitivity with respect to future inputs. The resulting behavior is random yet coherent both temporally and across multiple degrees of freedom. The controller is a parametric function

$$\mathbf{y}_t = C(\mathbf{x}_t; \mathbf{C}) \tag{1}$$

of the vector $\mathbf{x}_t$ of current sensory states of the robot. It generates a vector of motor commands $\mathbf{y}_t$ in dependence on the current values of the parameter matrix $\mathbf{C}$. The update of the parameters is based on the sensitivity of the distance between inputs and their predictions by means of an internal model. This model

$$\hat{\mathbf{x}}_{t+1} = M(\mathbf{x}_t, \mathbf{y}_t; \mathbf{M}), \tag{2}$$

produces a prediction of future states $\hat{\mathbf{x}}_{t+1}$ based on the current input $\mathbf{x}_t$ or action $\mathbf{y}_t$ or both, and a parameter matrix $\mathbf{M}$. The difference between actual and estimated state defines the prediction error

$$\boldsymbol{\xi}_{t+1} = \mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1}, \tag{3}$$

which gives rise to one of the two complementary objective functions that are relevant here, firstly the prediction error

$$\mathcal{E}_{t+1} = \|\mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1}\|^2, \tag{4}$$

which is used to adapt the parameters $\mathbf{M}$ of the internal model (2), and secondly the *time loop error*

$$\mathbf{E}_t = \|\mathbf{x}_t - \check{\mathbf{x}}_t\|^2, \tag{5}$$

which is based on a post-diction $\check{\mathbf{x}}_t$ of previous input $\mathbf{x}_t$ obtained via the inverse of Equation (2) given the new input $\mathbf{x}_{t+1}$, i.e., $\mathbf{E}_t$ is calculated only at time step $t+1$, and is related to the prediction error (4) by

$$\mathbf{E}_t = \|\boldsymbol{\eta}_t\|^2 = \boldsymbol{\eta}_t^\top \boldsymbol{\eta}_t = \boldsymbol{\xi}_{t+1}^\top (J_t J_t^\top)^{-1} \boldsymbol{\xi}_{t+1}. \tag{6}$$

where J is the linearization of the maps from current input to next input dependent on the current controller. As only the projection $\boldsymbol{\eta}$ of J^{-1} on $\boldsymbol{\xi}$ is relevant, the time loop error can be efficiently estimated. The homeokinetic learning rule updates the parameter matrix $\mathbf{C}$ of the controller (1) by gradient descent

$$\Delta C_{ij} = -\varepsilon_{\mathbf{C}} \frac{\partial \mathbf{E}_t}{\partial C_{ij}}, \tag{7}$$

where C_{ij} is an element of $\mathbf{C}$ and $\varepsilon_{\mathbf{C}}$ is a learning rate.

If the representational power is of less importance than the flexibility (Smith and Herrmann, 2019), then a simple quasi-linear system can be considered as sufficient. Below, when we will consider a multi-layered system, the representational power is meant to be achieved by the interaction between the layers each of which will consist of one instance of the current controller-predictor unit. A pseudo-linear controller, i.e., a quasi-linear function of the inputs with coefficients that are adaptive on the behavioral time scale,

$$\boldsymbol{y}_t = C(\mathbf{x}_t) = g(\mathbf{C}\mathbf{x}_t + \mathbf{c}) \tag{8}$$

and a linear model

$$\hat{\mathbf{x}}_{t+1} = M(\mathbf{y}_t) = \mathbf{M}\mathbf{y}_t + \mathbf{m}, \tag{9}$$

does thus not limit the complexity of achievable control. The parameters of the controller and the model are now the matrices $\mathbf{C}$ and $\mathbf{M}$ resp., which are complemented by the matching bias vectors $\mathbf{c}$ and $\mathbf{m}$. In order to incorporate limitations of actions

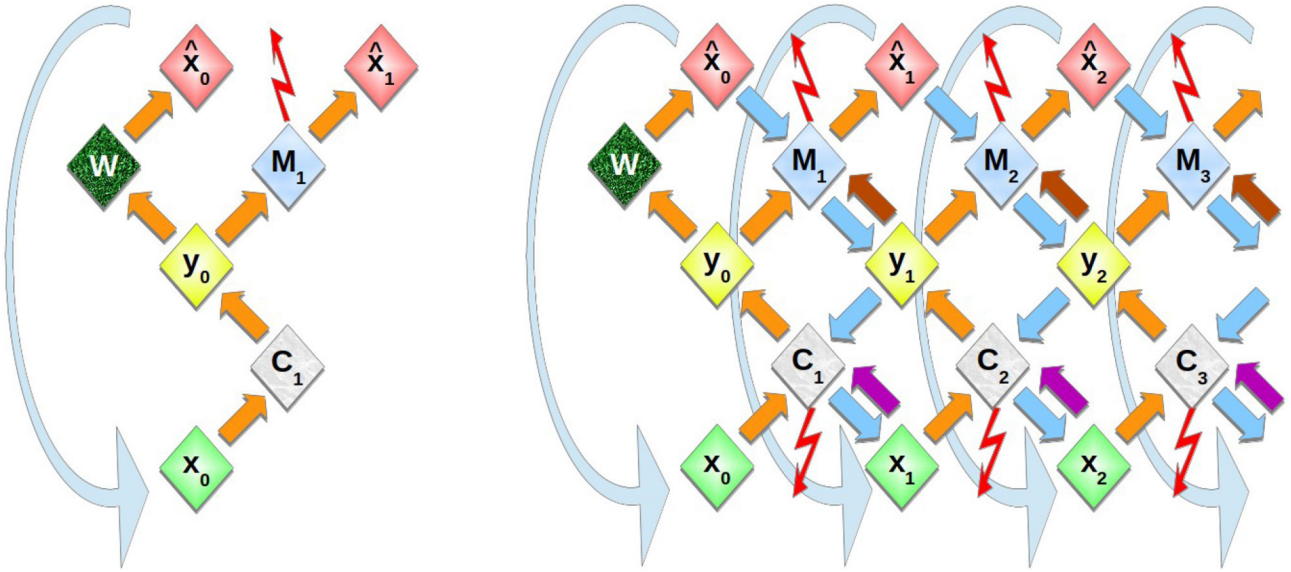


FIGURE 1 | Schematic representation of multi-layer homeokinetic learning. **Left**: In the elementary sensorimotor loop, a control action $\mathbf{y}_0$ is calculated by the controller C_1 and executed in the environment W which then produces the new input $\hat{\mathbf{x}}_0$. The prediction error is obtained as the difference of new sensory input $\hat{\mathbf{x}}_0$ and its prediction $\hat{\mathbf{x}}_1$ that was obtained from the previous input $\hat{\mathbf{x}}_0$. It is used in the update of the model M, see Equations (13), (14). **Right**: In homeokinetic learning, the time-loop error, i.e., the difference of previous input $\mathbf{x}_0$ and re-estimated previous input $\mathbf{x}_1$ (which is obtained via the downwards arrows and corresponds to $\hat{\mathbf{x}}_t$ in Equation 5), is used to update the controller parameters, see Equations (11), (12). The curved downward arrows indicate the time step: The "new" input that was previously predicted or obtained from the environment, is now used by the controller to produce the next action (rather than the re-estimated input). The inner layers follow exactly the same dynamics based on predictions from the respective outer layers rather than based on the environment. Top-down effects are included by additional connections This includes virtual actions (arrows from $\mathbf{y}_l$ to M_l) analogous to the initiation of actions in the environment, and virtual states taken into account by the controller (arrows from $\mathbf{x}_l$ to C_l). The activities are propagated alternatingly through the upwards (orange, violet, and brown) arrows and through the respective transposed matrices via downwards arrows (cyan), both of which correspond to a set of parallel fibers, whereas the adaptive interconnections are maintained in the controller (**C** nodes) or the model (**M** nodes).

of the robot, the controller is quasi-linear due to the element-wise sigmoidal function g. Because of the simple structure of Equation (8), we can omit here the state dependency (2) and define the model M only in motor space. The parameter update (7) becomes

$$\Delta C_{ij} = \varepsilon_{\mathbf{C}}\, \boldsymbol{\eta}^{\top} J \frac{\partial J}{\partial C_{ij}} \boldsymbol{\eta}, \tag{10}$$

and analogously for the bias term $\mathbf{c}$. With $\boldsymbol{\mu} = \mathbf{G}'\mathbf{M}^{\top}\left(J^{\top}\right)^{-1}\boldsymbol{\eta}$ and $\boldsymbol{\zeta} = \mathbf{C}\boldsymbol{\eta}$ the learning rules for a linear controller with a linear model are

$$\Delta C_{ij} = \varepsilon_{\mathbf{C}}\mu_i\eta_j - 2\varepsilon_{\mathbf{C}}\mu_i\zeta_i y_i x_j \tag{11}$$

$$\Delta c_i = -2\varepsilon_{\mathbf{C}}\mu_i\zeta_i y_i. \tag{12}$$

Simultaneously, but possibly with a different learning rate, the parameters $\mathbf{M}$ of the linear model (9) are updated via gradient descent on the standard prediction error (Equation 4, rather than Equation 6).

$$\Delta M_{ij} = -\varepsilon_{\mathbf{M}}\frac{\partial \mathcal{E}}{\partial M_{ij}} = \varepsilon_{\mathbf{M}}\xi_i y_j \tag{13}$$

$$\Delta m_i = -\varepsilon_{\mathbf{M}}\frac{\partial \mathcal{E}}{\partial b_j} = \varepsilon_{\mathbf{M}}\xi_i \tag{14}$$

where $\varepsilon_{\mathbf{M}}$ is the learning rate for the adaptation of the internal model. The ratio of the two learning rates $\varepsilon_{\mathbf{C}}$ and $\varepsilon_{\mathbf{M}}$ is known to be critical for the behavior of controlled robot (Smith and Herrmann, 2019). For the architecture presented next, an optimized ratio is to be used, see also **Figure 2**.

3. THE DIAMOND MODEL

3.1. Deep Homeokinesis

The DIAMOND model is a generalization of the homeokinetic controller described in section 2. As shown in **Figure 1**, the comparison of a state variable $\mathbf{x}(t)$ and its estimate $\hat{\mathbf{x}}(t)$ is now repeated also for estimates of estimates etc., $\mathbf{x}_0(t) = \mathbf{x}(t)$, $\mathbf{x}_1(t) = \hat{\mathbf{x}}(t)$, $\mathbf{x}_2(t), \ldots$, where each pair of neighboring layers corresponds to a homeokinetic controller that acts onto the lower layer as its environment and receives biases from the higher layer. In the inner layers (larger ℓ) the external information becomes less and less dominant.

In order to use homeokinetic learning in a multilayer architecture, several instances of the homeokinetic sensorimotor loop are stacked. The internal model of any lower layer serves as the "world" for the next higher layer. Likewise, estimates for input obtained at by a lower layer are the inputs for the higher layers, so each layer reproduces the elementary loop shown in **Figure 1**.

3.2. Simple Variant

The architecture consists of controllers for each layer $\ell < L$ (no controller for $\ell = L$)

$$\mathbf{y}_\ell\,(t) = C_{\ell+1}\,(\mathbf{x}_\ell\,(t)) = g\,(\mathbf{C}_{\ell+1}\mathbf{x}_\ell\,(t) + \mathbf{c}_{\ell+1}) \tag{15}$$

and linear models that are given by

$$\begin{aligned}\hat{\mathbf{x}}_\ell\,(t+1) &= M_\ell(\mathbf{y}_{\ell-1}\,(t)\,,\mathbf{y}_\ell\,(t)) \\ &= \mathbf{M}_\ell\mathbf{y}_{\ell-1}\,(t) + \tilde{\mathbf{M}}_\ell\tilde{\mathbf{y}}_\ell\,(t) + \mathbf{m}_\ell.\end{aligned} \tag{16}$$

which simplifies for the top layer $\ell = L$ where $\tilde{\mathbf{y}}_L\,(t) \equiv 0$, i.e., no higher effects are present.

In Equation (16) also the effect of virtual actions $\tilde{\mathbf{y}}_\ell\,(t)$, $\ell \geq 1$ is included as follows: First, the previous prediction of a layer $\hat{\mathbf{x}}_\ell\,(t-1)$ is copied into the input unit $\mathbf{x}_\ell\,(t)$ at the beginning of the new time step, see **Figure 1**. The back-propagated input $\check{\mathbf{x}}_\ell\,(t-1)$ that was used in Equations (5) and (6) is no longer needed. From $\mathbf{x}_\ell\,(t)$ a virtual action $\mathbf{y}_\ell\,(t)$ is computed that then contributes additively to the prediction (16). The controller update is here the same as for the one-layer model, and the $\tilde{\mathbf{M}}$ matrix (not shown in the figures) is updated in the same way as the $\mathbf{M}$ matrix.

3.3. Main Variant

The variant with extra connections (**Figure 1**) has for the controller

$$\begin{aligned}\boldsymbol{y}_\ell\,(t) &= C_{\ell+1}\,(\mathbf{x}_\ell\,(t)) \\ &= g\big(\mathbf{C}_{\ell+1}\mathbf{x}_\ell\,(t) + \tilde{\mathbf{C}}_{\ell+1}\hat{\mathbf{x}}_{\ell+1}(t-1) + \mathbf{c}_{\ell+1}\big)\end{aligned} \tag{17}$$

i.e., in the same way as new input $\hat{\mathbf{x}}_0\,(t+1)$ that is used to calculate the prediction error is also used in the next time step as input $\mathbf{x}_0\,(t)$, we are also for $\ell > 0$ using previous predictions as new virtual input. For the deepest layer $\ell = L$, Equation (17) is not applied, and for the penultimate layer we have simply

$$\boldsymbol{y}_\ell\,(t) = C_{\ell+1}\,(\mathbf{x}_\ell\,(t)) = g\,(\mathbf{C}_{\ell+1}\mathbf{x}_\ell\,(t) + \mathbf{c}_{\ell+1})\,. \tag{18}$$

For the model, Equation (16) is used as above.

While the first $\mathbf{C}$ matrix in Equation (17) is adapted learned in the standard way (see Equations 11 and 12), the matrix $\tilde{\mathbf{C}}$ is updated by gradient descent with respect to the prediction error for the action

$$\mathcal{E} = \left(\mathbf{y}_\ell\,(t) - \tilde{\mathbf{y}}_\ell\,(t)\right)^2,$$

where

$$\tilde{\mathbf{y}}_\ell\,(t) = g\left(\tilde{\mathbf{C}}_{\ell+1}\hat{\mathbf{x}}_{\ell+1}\,(t-1) + \tilde{\mathbf{c}}_{\ell+1}\right),$$

i.e., the input $\hat{\mathbf{x}}_{\ell+1}\,(t-1)$ from the more inner level is used to predict the motor output $\mathbf{y}_\ell\,(t)$. The update equations for $\tilde{\mathbf{C}}$ are similar to Equations (13) and (14), but also contains a derivative of g. Note that no loops are present in the network of **Figure 1**, which may not be a problem as the loops have no function (yet), and may be included later. However, it is not clear what "deliberation" could mean without these loops.

We assume that the inner (deeper) layers are updated first. The deepest layer $\ell = L$ has no variables, just the controller and the model. According to Equation (18), no higher-level input variables are needed in order to update the variables at $\ell = L-1$. In this way, virtual actions and virtual inputs are available to be used in Equations (17) and (16) to update the next layer toward the outer side, i.e., with lower ℓ. For the update of the matrices $\mathbf{M}$, $\tilde{\mathbf{M}}$, $\mathbf{C}$ and $\tilde{\mathbf{C}}$ the time order is not essential, if the variables are calculated as described above.

3.4. Main Variant With Deep Associations

As a further variant, which is, however, not implemented in the present simulations, a standard deep neural network can be employed to connecting the inputs x_ℓ directly between neighboring levels. In this case a separate set of connections $\mathbf{P}_\ell$ would be learned for map from $x_{\ell-1}$ to x_ℓ. The weights $\mathbf{P}$ are learned by the activations $\mathbf{x}_\ell$ that arise due to the activations of the network. In addition it is possible to add a further set of connections $\mathbf{R}$ that play the same role as $\mathbf{P}$, but for the predicted sensor values.

The network can sustain persistent activity that represents an action perception cycle. Activity in the subnetworks that are completed by recurrent connections arises by self-amplification of noise or spurious activity following the homeokinetic learning of the respective controller. It may be possible to use also the cycles more explicitly for learning, but we want to restrict ourselves here to one-step learning rule, i.e., gradients are calculated only over one The full model also includes perceptual pathways consisting of bridges between input-related units. In this way the network activity becomes shaped by standard deep feed-forward networks.

4. EXPERIMENTAL RESULTS

4.1. Active Response by the Recurrent Network

As a first test, we have considered the simple variant of the architecture (see section 3.2) when it is driven with a sinusoidal input and the "world" reproduces simply a noisy version of the motor action as next input to the robot. Typical results are shown in **Figure 2** for a two combinations of the learning rates $\varepsilon_\mathbf{C}$ (11, 12) and $\varepsilon_\mathbf{M}$ (13, 14), which lead either to an abstracted reproduction of the input in the deeper layers or to a self-organization of activity that, however remains without effect in this simple variant. At lower learning rates (left column), even deeper layers respond to the original input. In this case, the internal layers are square versions of the original input. For larger learning rates (right column), the internal layers have a different response. The fifth row shows a combination of homeokinetic adaptation (the red line between 310 and 320 s) and noisy output while still following the input from the first layer. Deeper layers (lower rows), tend have a decay in the generation of motor action attributed to the squashing function.

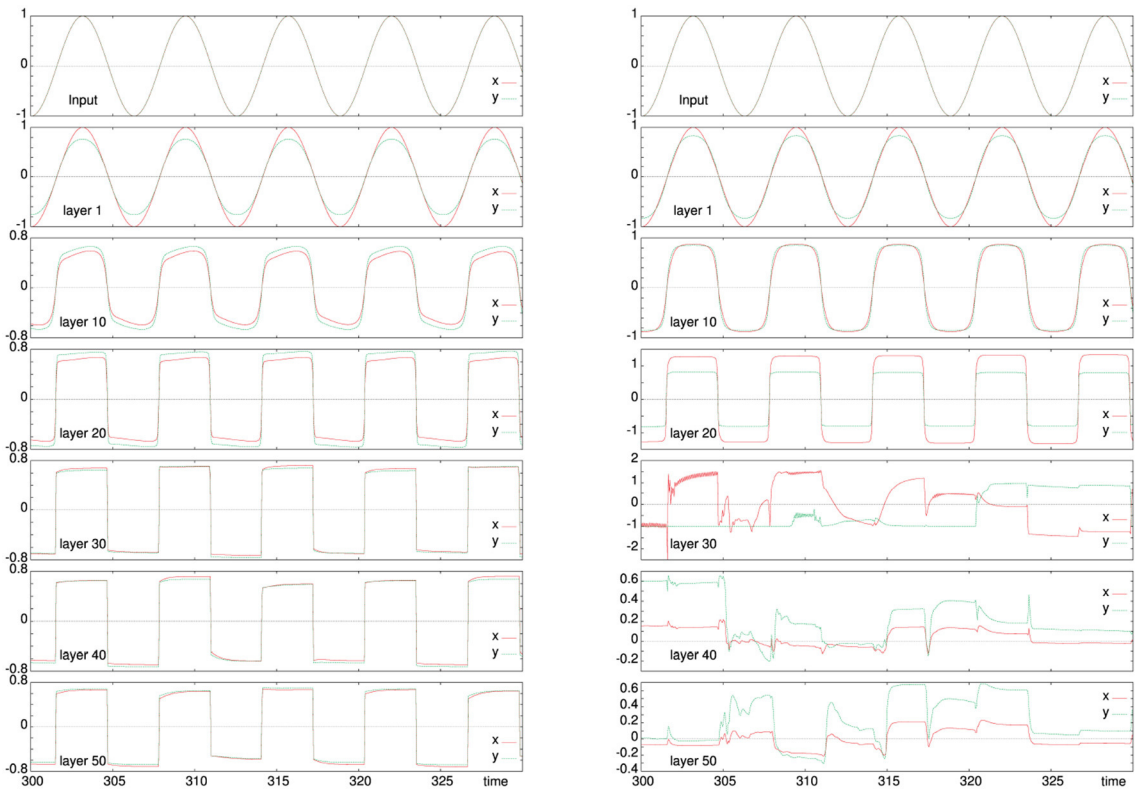


FIGURE 2 | Activity evolution in a perceptually connected network structure according to the model in section 3.2. The sensory trajectory is shown by the solid line (red) and the intermediate motor action by the dashed line (green). The top row gives the input activity, the second row the activity of the first layer and the following rows show every 10th layer of the architecture to a total depth of 50. The left panel is for learning rates $\varepsilon_{\mathbf{M}} = 0.01$, $\varepsilon_{\mathbf{C}} = 0.05$, and the right one for $\varepsilon_{\mathbf{M}} = 0.1$, $\varepsilon_{\mathbf{C}} = 0.2$. While at low leaning rates, the input is similar across all layers, for larger ratios $\varepsilon_{\mathbf{M}}/\varepsilon_{\mathbf{C}}$ the model is more flexible and the deeper activity becomes largely independent on the input, which allows for self-organized activity in the deeper layers that is not immediately affecting the outside world.

4.2. A Wheeled Robot in the Hills

The main variant (section 3.3) is used in an exploration task, where a four-wheeled robot is expected to cover a large portion of an unknown territory (Smith and Herrmann, 2019). The hilly landscape shown left in **Figure 3** can be scaled in vertical direction such that different levels of difficulty can be achieved ranging from a flat ground (level 0) to slopes that require maximal motor power (level 1) and that can cause instabilities and thus large prediction errors (4). The activity decays in a five-layer DIAMOND model for a flat arena, as the inner layers are not needed, whereas for a hilly landscape (difficulty level > 0) the inner layers did not show much attenuation. The behavior of the robot is evaluated based on a 10×10 grid overlaid to the square-shaped arena. The number of visited grid cells is averaged over five runs for each difficulty and each controller depth and represented as a coverage rate. The total coverage was in all cases below 50% such that the increase of the coverage with time was nearly linear.

Whereas a single layer can achieve a similar performance across all terrain difficulties, for increasing difficulty of the task the higher layer are more and more engaged and take advantage of the increased errors in the terrain that provide thus a potential for a more comprehensive coverage of the arena per time unit.

4.3. A Spherical Robot in a Polygonal Arena

Finally, we studied a simulated spherical robot which is controlled by three masses that a movable along internal axes, see **Figure 4**, left. The robot is exploring freely in an polygonal environment which was chosen to discourage circular movement along the wall. The controller picks up quickly a suitable rhythm of the internal weights that is effecting in moving the robot in any direction. Collisions with wall usually stop the robot until the emergence of a different mode of the movements of the internal weights moves the robot in a different direction. Although a more systematic study is yet to be performed, it is already obvious that adding a small number of additional layers increases the behavioral repertoire of the robot and reduces the duration of any wall collisions and re-emergence of behavior in the robot. The example is also meant to demonstrate, that the applications of the learning rule and architecture are beyond exploration of a planar arena and can be used in order to generate and to organize elementary robotic behaviors.

5. DISCUSSION

The numerical results seem to imply that a few layers are sufficient, i.e., a larger number of layers does not lead

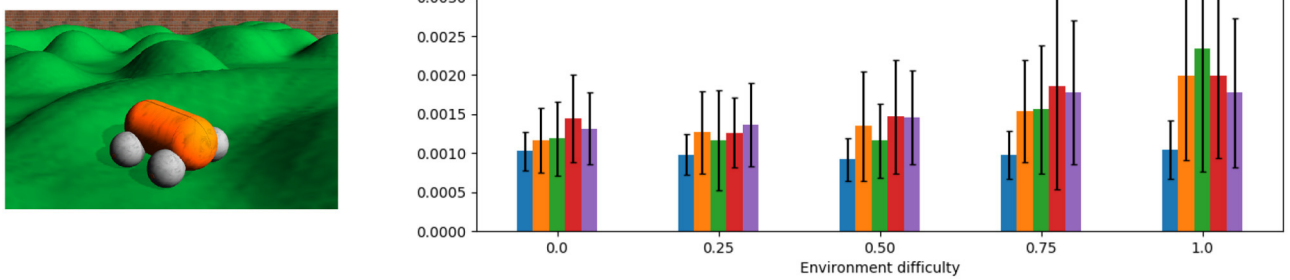


FIGURE 3 | A four-wheeled robot exploring a hilly landscape. (See text and Smith and Herrmann, 2019 for details on the task). The panel on the right shows results for five levels of difficulty (linear scaling of the slopes, with the simples level being a flat ground) and five depths of the network ($\ell = 1, 2, 3, 4, 5$) are considered, showing an increased exploration capability. The code for simulator (Der and Martius, 2012) and the DIAMOND controller architecture described here is available at https://github.com/artificialsimon/diamond.

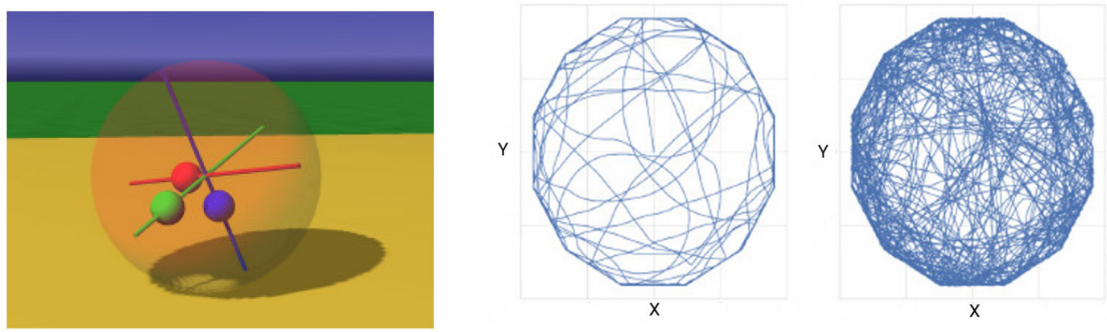


FIGURE 4 | Spherical robot **(left)** in a dodecagonal arena. For a one-layer architecture, the robot mostly follows the wall **(middle)**, while for a 3-layer network, the robot shows a highly exploratory behavior **(right)**.

to further improvements or may require a much longer learning time than attempted here. It should, however, be considered that the tasks and environments are all very simple, such that it is not possible to generalize this observation to more complex situations. It can nevertheless be expected that the spontaneous internal activations that were observed for suitable learning rate ratios, lead to a learning time that is approximately linearly increasing with the number of layers, and not much worse. This is suggested by earlier results with homeokinetic learning rule (Martius et al., 2007).

The present model is a representation of the idea (see e.g., Anderson et al., 2012) that it is difficult to define a clear boundary between brain and body or even between body and world. At all layers the system follow the same principles in its adaptation of the actions onto lower layers and in the learning of a model that affects higher layers. The reduction of complexity of the internal dynamics toward higher layers is counterbalanced by the autonomous activity such that the main eigenvalue at each layer will hover near unity (Saxe et al., 2014).

Although the activity is updated here in parallel in all layers, the stacked structure is clearly similar to the subsumption architecture (Brooks, 1986) as it allows for shorter or longer processing loops. It remains to be studied whether more general architectures are beneficial, especially when more complex tasks are considered.

In **Figure 1**, it is understood that the dynamical variables (x, y, and $\hat{x}$) exist each in two instances, one updated by the controlling and predictive pathways, the other by the feedback within the re-estimation system. The need to disambiguate these units points to an interesting parallel to the roles of the layers of the mammalian cortex.

Finally, it should be remarked the principle of predictive coding is inherent in the architecture from the homeokinetic principle. Activity can only travel to the deeper layers if it is not already predicted by the internal model of the

current layer. In some cases this can lead to a complete decay of the activity in the deeper layers (see **Figure 3**), although more complex robots and more challenging environments need to be studied in order to precisely identify parallels to the predictive coding principle in natural neural systems.

DATA AVAILABILITY STATEMENT

All datasets generated for this study are included in the article/supplementary material.

AUTHOR CONTRIBUTIONS

JH, BS, and SS: conception. JH and SS: model design. RD, SS, and CI: experiments. JH, BS, and SS: writing. JH: project organization.

FUNDING

SS was supported by the ORCA Hub EPSRC project (EP/R026173/1, 2017-2021). CI was supported by grant EP/L016834/1 for the University of Edinburgh, School of Informatics, Centre for Doctoral Training in Robotics and Autonomous Systems (http://www.edinburgh-robotics.org) from the UK Engineering and Physical Sciences Research Council (EPSRC). BS received funding from the National Key R&D Program of China (2019YFA0709503).

ACKNOWLEDGMENTS

We thank Georg Martius (Tübingen), Klaus Pawelzik (Bremen), Alessandro Treves (Trieste), and Hemang Kanwal (Edinburgh) for stimulating discussions. JH is grateful to CAS SIA for their kind hospitality.

REFERENCES

Adams, R. A., Shipp, S., and Friston, K. J. (2013). Predictions not commands: active inference in the motor system. *Brain Struct. Funct.* 218, 611–643. doi: 10.1007/s00429-012-0475-5

Anderson, M. L., Richardson, M. J., and Chemero, A. (2012). Eroding the boundaries of cognition: implications of embodiment. *Topics Cogn. Sci.* 4, 717–730. doi: 10.1111/j.1756-8765.2012.01211.x

Brooks, R. A. (1986). A robust layer control system for a mobile robot. *J. Robot. Automat.* RA-2, 14–23. doi: 10.1109/JRA.1986.1087032

Carvalho, J. T., and Nolfi, S. (2016). Behavioural plasticity in evolving robots. *Theory Biosci.* 135, 201–216. doi: 10.1007/s12064-016-0233-y

Der, R. (2001). Self-organized acquisition of situated behaviors. *Theory Biosci.* 120, 179–187. doi: 10.1007/s12064-001-0017-9

Der, R., and Martius, G. (2012). *The Playful Machine: Theoretical Foundation and Practical Realization of Self-Organizing Robots, Vol. 15.* Springer Science & Business Media.

Fukushima, K., and Miyake, S. (1980). "Neocognitron: self-organizing network capable of position-invariant recognition of patterns," in *Proc. 5th Int. Conf. Patt. Recogn.* (Berlin), 459–461.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.* 18, 1527–1554. doi: 10.1162/neco.2006.18.7.1527

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, eds F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Lake Tahoe, NV: Curran Associates Inc.), 1097–1105.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *J. Mach. Learn.* 17, 1334–1373. doi: 10.7746/jkros.2019.14.1.040

Martius, G., and Herrmann, J. M. (2011). Variants of guided self-organization for robot control. *Theory Biosci.* 131, 129–137. doi: 10.1007/s12064-011-0141-0

Martius, G., Herrmann, J. M., and Der, R. (2007). "Guided self-organisation for autonomous robot development," in *Advances in Artificial Life. ECAL 2007. Lecture Notes in Computer Science*, eds F. Almeida e Costa, L. M. Rocha, E. Costa, I. Harvey, and A. Coutinho (Berlin, Heidelberg: Springer), 766–775. doi: 10.1007/978-3-540-74913-4_77

Oudeyer, P., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Trans. Evol. Comput.* 11, 265–286. doi: 10.1109/TEVC.2006.890271

Pinneri, C., and Martius, G. (2018). "Systematic self-exploration of behaviors for robots in a dynamical systems framework," in *Artificial Life Conference Proceedings* (MIT Press), 319–326. doi: 10.1162/isal_a_00062

Rusu, P., Petriu, E. M., Whalen, T. E., Cornell, A., and Spoelder, H. J. W. (2003). Behavior-based neuro-fuzzy controller for mobile robot navigation. *IEEE Trans. Instrument. Meas.* 52, 1335–1340. doi: 10.1109/TIM.2003.816846

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2014). "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *International Conference on Learning Representations*.

Serre, T., Oliva, A., and Poggio, T. (2007). A feedforward architecture accounts for rapid categorization. *Proc. Natl. Acad. Sci. U.S.A.* 104, 6424–6429. doi: 10.1073/pnas.0700622104

Smith, S. C., and Herrmann, J. M. (2019). Evaluation of internal models in autonomous learning. *IEEE Trans. Cogn. Dev. Syst.* 11, 463–472. doi: 10.1109/TCDS.2018.2865999

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 09 October 2020
doi: 10.3389/fnbot.2020.568359

Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics

Travis DeWolf[1]*, *Pawel Jaworski*[1] and *Chris Eliasmith*[1,2]

[1] Applied Brain Research, Waterloo, ON, Canada, [2] Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON, Canada

In this paper we demonstrate how the Nengo neural modeling and simulation libraries enable users to quickly develop robotic perception and action neural networks for simulation on neuromorphic hardware using tools they are already familiar with, such as Keras and Python. We identify four primary challenges in building robust, embedded neurorobotic systems, including: (1) developing infrastructure for interfacing with the environment and sensors; (2) processing task specific sensory signals; (3) generating robust, explainable control signals; and (4) compiling neural networks to run on target hardware. Nengo helps to address these challenges by: (1) providing the NengoInterfaces library, which defines a simple but powerful API for users to interact with simulations and hardware; (2) providing the NengoDL library, which lets users use the Keras and TensorFlow API to develop Nengo models; (3) implementing the Neural Engineering Framework, which provides white-box methods for implementing known functions and circuits; and (4) providing multiple backend libraries, such as NengoLoihi, that enable users to compile the same model to different hardware. We present two examples using Nengo to develop neural networks that run on CPUs and GPUs as well as Intel's neuromorphic chip, Loihi, to demonstrate two variations on this workflow. The first example is an implementation of an end-to-end spiking neural network in Nengo that controls a rover simulated in Mujoco. The network integrates a deep convolutional network that processes visual input from cameras mounted on the rover to track a target, and a control system implementing steering and drive functions in connection weights to guide the rover to the target. The second example uses Nengo as a smaller component in a system that has addressed some but not all of those challenges. Specifically it is used to augment a force-based operational space controller with neural adaptive control to improve performance during a reaching task using a real-world Kinova Jaco^2 robotic arm. The code and implementation details are provided[1], with the intent of enabling other researchers to build and run their own neurorobotic systems.

Keywords: Nengo, neuromorphic, neurorobotic, spiking neural networks, robotic control, adaptive control, embedded robotics

OPEN ACCESS

Edited by:
Subramanian Ramamoorthy, University of Edinburgh, United Kingdom

Reviewed by:
Jeffrey L. Krichmar, University of California, Irvine, United States
Yulia Sandamirskaya, Intel, Germany

***Correspondence:**
Travis DeWolf
travis.dewolf@appliedbrainresearch.com

Received: *01 June 2020*
Accepted: *01 September 2020*
Published: *09 October 2020*

Citation:
DeWolf T, Jaworski P and Eliasmith C (2020) Nengo and Low-Power AI Hardware for Robust, Embedded Neurorobotics. Front. Neurorobot. 14:568359. doi: 10.3389/fnbot.2020.568359

1. INTRODUCTION

Specialized AI hardware offers exciting potential for the development of low-power, highly responsive robotic systems with embedded control. Edge devices for accelerating neural networks are starting to become commercially available from companies, such as NVIDIA, BrainChip, GrAI Matter Labs, Google, Intel, and IBM. While the promise of low-power and low-latency embedded

[1] https://github.com/abr/neurorobotics-2020

processing and control is highly desirable, the process of implementing algorithms on the hardware generally remains a significant hurdle. Developing neural networks for processing sensory data and generating control signals is a difficult problem, and adding further constraints specific to a particular piece of hardware only increases the challenge. In this paper we focus on the development of spiking neural networks (SNNs) for the subset of devices known as neuromorphic hardware (Mead, 1990). Effectively using such hardware often requires additional expert knowledge outside of traditional machine learning and neural network methods to program effectively. In short, it is difficult to quickly and easily build robust, integrated neural models for controlling robots using neuromorphic hardware.

Building neurorobotic systems can be characterized as consisting of four tasks:

1. Developing infrastructure to send and receive signals from the environment. There are a multitude of different interface protocols for sensors, hardware, and simulators. To minimize development time, simple interfaces should be available and interchangeable with minimal changes to the model description.
2. Processing task specific sensory signals. Deep neural networks (DNNs) are the principle machine learning tool used for sensory processing, and it is important to take advantage of the extensive literature and solutions in this field. To that end, users need to be able to take DNNs, convert them to networks that can run on neuromorphic hardware, and integrate them into a neurorobotics control system. For systems using perception methods not rooted in neural networks, it is also important to be able to easily integrate their output with downstream networks.
3. Generating robust control signals with explainable neural networks. When generating control signals, having guarantees on performance is important, and often necessary. To accomplish this users needs to know exactly what operations are being implemented to guarantee stability. The Neural Engineering Framework (NEF; Eliasmith and Anderson, 2003) offers "white-box" neural network development methods that allow integration of these methods into neurorobotics control systems, making an API for building up such networks quickly desirable.
4. Compiling neural networks to run on multiple targeted hardware platforms. During the process of designing control and perception systems it is often desirable to develop neural network models on standard hardware with minimal compilation overhead. Once a prototype network is working, it should be straightforward to compile to targeted special purpose hardware. Being able to compile the same model to different hardware can greatly speed up the development of neurorobotics systems.

In this paper we present a neurorobotics development workflow for building neural networks that run on standard and neuromorphic hardware using the Nengo neural modeling platform (http://nengo.ai/; Bekolay et al., 2014). As part of this workflow, we take advantage of the NengoInterface package to streamline interfacing with the physics simulators, the NengoDL package for integrating Keras and TensorFlow models that process incoming sensory data, and the NengoLoihi package for compiling the model to run on Intel's Loihi neuromorphic chip (Davies et al., 2018).

We illustrate two variations on this workflow by describing two example neurorobotics applications in detail. The first example implements an end-to-end perception and action system in Nengo for tracking a target with a rover simulated in Mujoco (Todorov et al., 2012). The rover has four mounted cameras whose input is fed into a DNN built using Keras. The DNN estimates the distance to the target, and this estimate is sent to a control network which generates torques to apply to the steering wheel and drive wheels to move the rover to the target. This full system is then compiled onto Loihi. In the second example, we demonstrate how Nengo can be integrated with an existing system by augmenting a standard robotic arm force controller using a neural adaptive controller that learns online. We implement the adaptive component both on standard hardware and Loihi, where we take advantage of its on-chip learning. We compare implementations of the adaptive control system as it drives a physical Jaco^2 robot arm from Kinova to perform a reaching task while adapting to the unmodeled force of holding a two pound weight. We discuss the workflow bottlenecks and challenges that are encountered, addressed, and remaining.

2. BACKGROUND

2.1. Nengo and Supporting Development Packages

Nengo is a neural modeling development and simulation platform. Users specify the architecture of models using a Python-based API, referred to as the "front-end," and then compile their model for simulation on hardware using a "back-end." The API is designed such that the same model can be run on different hardware with few to no changes in the front-end script. Supported hardware includes CPUs, GPUs, FPGAs, and specialized neuromorphic hardware (such as Intel's Loihi chip).

Nengo users are able to quickly design and simulate neural networks, and use the NengoGUI package to visualize and interface with them during run-time. The NengoDL package extends Nengo's API to interface with and integrate deep and machine learning networks built in Keras or TensorFlow, as well as take advantage of TensorFlow's resource distribution manager for efficient simulation across multiple processors. The NengoInterfaces package provides easy interface access with the Mujoco simulator, abstracting out the setup and overhead involved in connecting, running, communicating, and restarting Mujoco simulations. The NengoLoihi package allows us to compile our models to run on the Loihi, and also handles communication to and from the chip. Additionally, the NengoLoihi package provides a Loihi emulator that allows users to run their models while simulating Loihi dynamics and computations on their computers, which aids efficient development. Nengo has more supporting packages, but in the interest of space we limit our review

to the above packages relevant to the work presented in this paper.

2.2. The Loihi Chip

We use Intel's Loihi chip (Davies et al., 2018) for demonstration in the examples below. The Loihi chip is a many-core mesh with 128 neuromorphic cores, 3 embedded × 86 processing cores, and off-chip communication interfaces. An asynchronous network-on-chip communicates packetized messages between cores, allowing write, read request and response, spike messages, and barrier messages for time synchronization to be sent between cores. Each neuromorphic core has 1,024 neural "compartments," where a compartment can be allocated to simulate a neuron or dendrite.

The NengoLoihi package allows users to compile their front-end script to the Loihi chip, handling all of the low-level mapping and communication. Some front-end scripts will require modification specifying low-level details, such as how to allocate neural populations across Loihi cores, but largely the details of this low-level mapping are abstracted out and handled automatically.

2.3. Other Neurorobotic Workflows and Toolkits

Most commonly, building neurorobotic applications involves hand-crafting and tuning SNNs for the task of interest. In Gutierrez-Galan et al. (2020) the authors build an SNN inspired by biology to implement a central-pattern generator that runs on the SpiNNaker neuromorphic board (Furber et al., 2014) and drives a hexapod robot to walk, trot, or run. In Kreiser et al. (2019), the authors hand craft an SNN to run on the Loihi to steer a small rover. In Stagsted et al. (2020), a PID controller is implemented in an SNN running on Loihi to steer an unmanned aerial vehicle. The authors accomplish this using one-hot encoding, such that only one neuron in a population is able to spike at a time, and each neuron represents a different possible variable value, to build up networks implementing addition and subtraction, at which point a PID controller can be built. Implementation of non-linear functions is listed as future work. We note that these models can be built using the Nengo API, and the NEF API makes non-linear function implementation straight-forward.

The authors of Taunyazov et al. (2020) implement an SNN visual tactile system that runs on Loihi and performs container classification and rotational slip detection. The network is a deep net trained with the SLAYER (Bam Shrestha and Orchard, 2018) method, which uses stochastic spiking neurons to overcome the undefined derivative in spiking neurons that prevents backpropagation from working. In Hwu et al. (2017), the authors train an Energy-Efficient Deep Neural Network (EEDN), a deep network designed specifically to run on IBM's TrueNorth neuromorphic chip (Sawada et al., 2016), on trail photos to train up a network that attaches to a real-world rover and guides it along a path. The authors mention that the trained weights work well in a standard convolutional neural network or in the EEDN, which can transfer its weights directly to the TrueNorth. As we detail in our examples below, the NengoDL package allows users to take advantage of similar deep learning methods for training SNNs to run on neuromorphic hardware.

Another way to program neuromorphic hardware is using the Python Neural Networks (PyNN) interface (Davison et al., 2009). PyNN is a front-end API that shares Nengo's goal of creating a high-level front-end API that specifies neural network architecture without being tied to the low-level implementations specifics. PyNN was developed to standardize scripting neural networks across several different low-level neural simulators, including Brian (Goodman and Brette, 2008), NEURON (Hines and Carnevale, 1997), and Nest (Gewaltig and Diesmann, 2007). Since its development, others have extended PyNN to include backends for other simulators and neuromorphic hardware. While Nengo allows the same low-level specificity of PyNN, Nengo also allows many of these details to be easily abstracted, and has more focus on high-level objects that speed the development of large or complex neural systems. For example, to implement a communication channel between two populations of neurons takes ∼12 lines in Nengo, and over 80 lines in PyNN (Bekolay, 2011). Importantly, Nengo also supports methods for generating connection weight matrices, including the NEF as well as Keras/TensorFlow techniques, which PyNN does not.

The Neurorobotics Platform (NRP; Falotico et al., 2017) is a web-based simulation environment for running SNNs hooked up to virtual robots. Developed as part of the Human Brain Project (Markram, 2012), the goal of the NRP is to streamline the process of running experiments with SNNs and robots. The NRP lets users quickly select a virtual environment, robot, and SNNs to run an experiment. The NRP has a broad scope, offering tools, such as the web-based robot designer, experimental workflow editor, and Gazebo simulation environment editor. In contrast, Nengo focuses primarily on the development, integration, and simulation of neural networks (both spiking and non-spiking), support of different neural network programming paradigms like the NEF and Keras/TensorFlow, compiling to different hardware backends, and systems interfacing through Python. There is potential for collaboration between Nengo and the NRP, expanding the neural network development and simulator interfacing of the NRP, and the experiment design and web interface of Nengo.

3. NEUROROBOTIC ROVER SYSTEM

In this example we develop an end-to-end perception and action system for tracking a target with a rover in Mujoco (Todorov et al., 2012). The simulated rover we use is a four wheeled vehicle, built using Mujoco's XML modeling language, with Ackerman steering and rear differential drive in a boundless environment with no obstacles. The rover has 4 RGB cameras mounted on its back, each with a 90° field-of-vision, that provide a full 360° view of the environment, and a sensor on the front wheels that provide steering angle information. The rover accepts two torque input signals, one to control the acceleration of the rear wheels, and one to turn the front wheels right or left. The target is a red sphere that floats in the air and warps to a new location (generated from

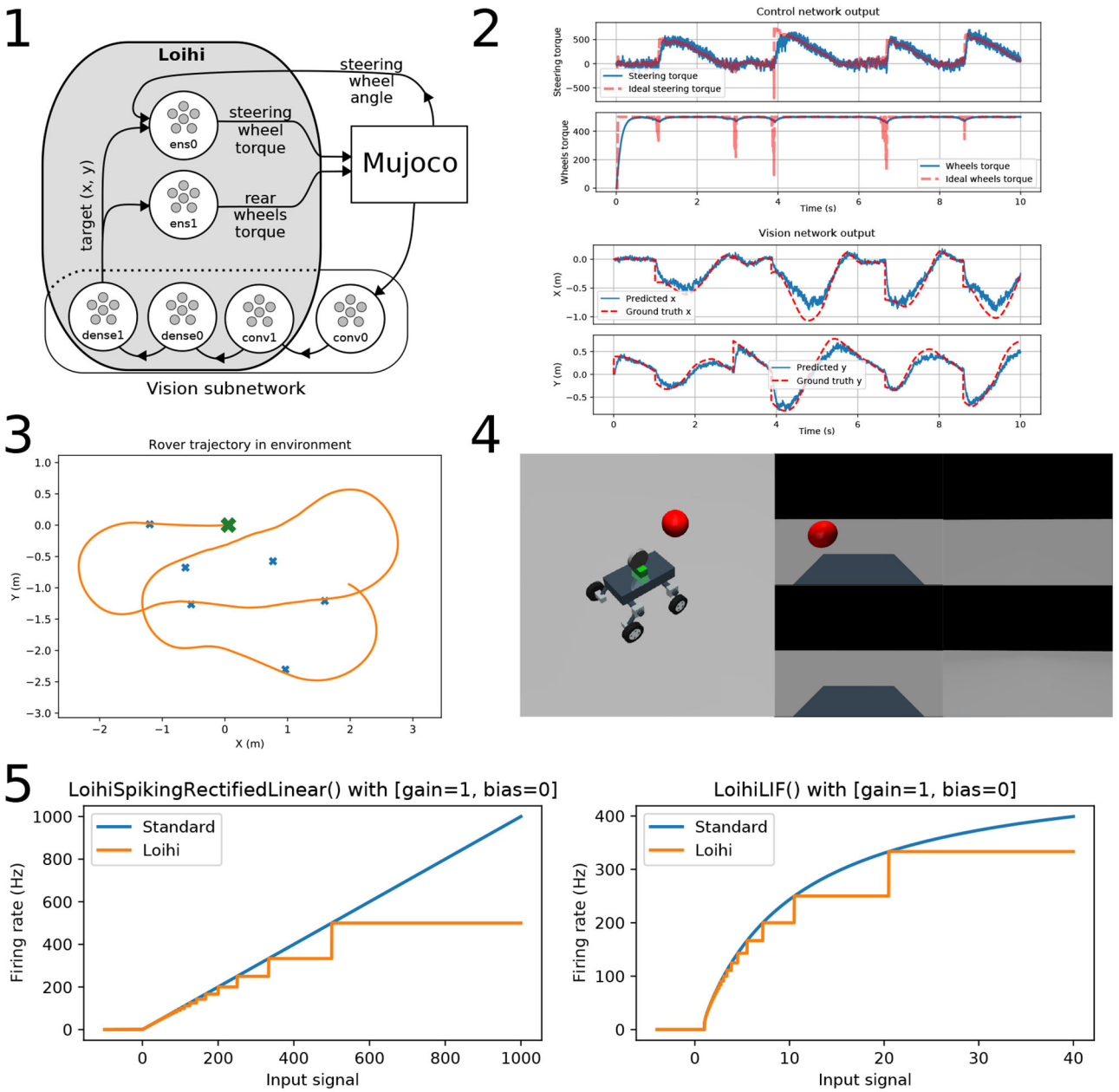


FIGURE 1 | A neurorobotic rover. **(1)** A diagram of the neural network implementing the rover's perception and action system. The components inside the gray rounded rectangle are run on the Loihi hardware. Arrows indicate the flow of information. The "conv" prefix denotes convolutional layers, the "dense" prefix denotes fully connected layers, and the "ens" prefix denotes NEF layers. **(2)** Top: The control network's approximation of the steering (top) and acceleration (bottom) signals are plotted in blue. The ideal function output is plotted as a dashed red line. Bottom: The vision network's estimate of the target x (top) and y (bottom) location relative to the rover is plotted in blue. Ground truth is plotted as a dashed red line. **(3)** The trajectory followed by the rover is plotted in orange as it approaches different targets, as seen from above. The rover starts in the center at the green X and drives to the different targets, plotted as blue Xs. **(4)** Left: An image of the rover and the target in the Mujoco environment. Right: The images from the four mounted cameras attached to the rover, which generate the input to the vision network. **(5)** The firing rate curve of standard and Loihi neurons, to show the effects of discretization on the on-chip activity profiles neurons. (Left) Spiking rectified linear neurons; (right) Leaky integrate-and-fire neurons.

a random distribution within 3 m of the origin) when the center-of-mass of the rover is within 50 cm of the center of the target. **Figure 1**–4 shows the rover in the world on the left, and the view from each of the 4 cameras on the right.

To begin developing our perception and action systems, we first build out the controller without using neural networks. We use the exact ground-truth information provided by Mujoco to identify the target location relative to the rover, and calculate

the torques for acceleration and steering in Python. Next, we address sensory perception by building a DNN that can identify target (x, y) location relative to the rover based on input from the four mounted cameras. This requires building a dataset using the simulator to train the DNN. We initially train the system using standard artificial rate neurons, to confirm that the desired level of performance can be achieved with our network architecture. We then us NengoDL to convert the network to spiking neurons and tune the parameters to optimize performance. Next, we replace each function in the control system with spiking analogs, testing each in isolation before finally integrating the entire spiking network. Once performance is achieved in a fully spiking neural network, we move the network simulation from Nengo into the NengoLoihi emulator, and tune the parameters again to optimize under the constraints of the Loihi. Finally we compile the network to the Loihi hardware, again using NengoLoihi. In the next sections we describe each of these steps in more detail.

3.1. Interfacing With Mujoco

Interfacing to Mujoco is done through NengoInterfaces, which uses the `mujoco-py` (Ray et al., 2020) library for Python bindings to the Mujoco C API. The interface accepts force signals from the neural network, applies them inside Mujoco and moves the simulation forward one time step, and then returns feedback from the rover. Environment information can be accessed directly from the NengoInterfaces API, and less common functions are available through the `mujoco-py` simulation and environment model parameters.

3.2. Processing Visual Input Using a Keras DNN Converted to a Nengo SNN

The network used for tracking the target location consists of two convolutional layers and three dense layers, and is shown in the bottom block of **Figure 1**–1. The first convolutional layer uses 1×1 kernels with a stride of 1, and a filter size of 3; its purpose is to convert the image signal into spikes to be sent to the layers running on Loihi[2]. To generate the input image, we take a 32×32 pixels resolution snapshot from each camera, and concatenate them horizontally to create a 32×128 pixels input to the network. This resolution was chosen as the smallest network size that could still identify targets at a distance of 3 m. To retrieve the signal from the last layer running on-chip we use neural probes, which monitor spiking activity.

The dataset used for training the model was generated by recording both input from the mounted cameras and the relative distance to the target. The data was collected while our non-spiking control system drove the rover to the targets, recording every 10th frame. The final dataset used consists of roughly 40,000 images and target (x, y) locations (the height of the target is constant and is not relevant to control so we ignore it).

Training the network with non-spiking ReLU activation functions using standard DNN tools (i.e., Keras) was the first step. This allows us to validate the network architecture. For all of our training we use the RMSprop optimizer from TensorFlow and the mean squared error loss function on network output to learn to output the target (x, y) locations associated with an image. When converting the network into spiking neurons take into account both the desired firing rates and the activation function of neurons running on the Loihi. We have found that if the average firing rates are <50 Hz, spiking neurons are not driven strongly enough to generate any activity. We target the 175 Hz range for firing rates, because it is large enough to ensure spiking, but still inside the range where the Loihi neurons approximate standard neurons well (discussed below). To achieve this, we initialize the weights of our network by setting the `scale_firing_rates` parameter of the NengoDL built-in Keras converter to `400`. This parameter encourages the optimizer to converge to firing rates that are higher or lower, based on the scaling. An alternative, and more fine-grained and reliable method, is to add a firing rate regularization term to the cost function that penalizes neurons firing outside of the desired range.

The second factor we need to account for is the activation function of neurons on the target neuromorphic hardware. Neurons on the Loihi have a unique activation profile because of discretization that occurs on-chip, as shown in **Figure 1**–5. We use NengoLoihi's model of the Loihi rectified linear neuron during training to ensure that the network is trained on the same kind of activation functions used during inference.

Finally, we set network synapses throughout the network as required to smooth out the signal and filter noise. In Nengo, you can set synapses to 0 to implement no filtering, or to None to collapse the computations of two connected layers into a single layer. We set each of the synapses on the connections between layers to None. This speeds up the propagation of spikes through the network, but also has the potential to decrease performance due increased noise in the signal. We found empirically, however, that applying a 0.05 s time constant low-pass filter on the vision network output smoothed the target location estimate and improved the control signal generated downstream.

3.3. Generating Robust, Explainable Control Signals Using the NEF

We described the NEF as a "white-box" approach to building neural networks because of its mechanistic approach. Briefly, the NEF uses populations of neurons to represent vectors, feed-forward connections between populations to implement functions on those vectors, and recurrent connections to implement differential equations. Rather than specifying a task-level cost function, as in standard machine learning methods, the user must first design a circuit that solves the problem, including specifying a state space representation, set of computations, and flow of information. The user then uses Nengo's NEF API to implement this circuit in neurons. These added top-down constraints give clear network structure that allows users to identify points of error and apply specific changes to debug and improve network performance. This is in contrast to "black-box" deep learning methods, which use training algorithms to find a network configuration that solves the problem, without knowledge of the function implemented. In situations, such

[2]It is also possible to send information to Loihi by setting the bias and current for neurons, but we have found this method to be slower for a dynamic input signal.

as complex visual analysis where the algorithmic solution is unknown this is very desirable, but in cases where proven solutions are available and we would like performance guarantees the methods of the NEF are preferable.

The motor system for the rover is implemented using the NEF. The first step in this implementation is deriving the functions for vehicle acceleration and steering wheel control. We calculate acceleration as distance to target multiplied by a gain term, clipped to a maximum magnitude empirically chosen to prevent the rover from flipping when traveling at top speed and turning sharply. Formally,

$$u_{acceleration} = k_a \min(\|(x^*, y^*)\|, 1) \tag{1}$$

where u is the control signal sent to the rover, k_a is the acceleration gain term, x^* and y^* are the target location relative to the rover, and $\|\cdot\|$ denotes the 2-norm.

The torque applied to the steering wheel is calculated with a simple proportional controller using the difference between the current and desired angle of the wheels multiplied by a gain term. The desired angle is calculated as the angle to the target using $\arctan2(-x^*, y^*)$, where order of the arguments and the negative sign in front of the x term account for the orientation of the rover relative to the environment. Formally,

$$u_{steer} = k_p(\arctan2(-x^*, y^*) - q) \tag{2}$$

where k_p is the proportional gain term, and q is the current angle of the steering wheel.

The neural circuit implementation of this controller is done in two parallel ensembles, with the connections weights on the outbound connections calculated to approximate the above equations, as shown in the top portion of **Figure 1**–1. In Nengo, we project the relevant variables into each population and specify the functions to be computed on their outbound connections using Python code. Nengo then solves for connection weights using the principles of the NEF. In this particular case, we compute these functions using separate ensembles, rather than having a single ensemble with two separate outbound connections, because the functions they are calculating depend on different sets of variables. The acceleration function only requires the estimated target (x, y) values for calculation, while the steering function requires the estimated target (x, y) and the current angle of the steering wheel. While the variables required by the acceleration function are a subset of the variables used in the steering function, we can achieve greater precision by using an ensemble that only encodes the target (x, y).

In the hardware implementation, each ensemble consists of 4,096 Loihi leaky integrate-and-fire neurons, spread across four cores. This specific number of neurons is chosen to satisfy hardware constraints on the number of inbound connections a population of neurons can receive. The neurons in the ensemble are set up to have maximum firing rates between 175 and 220 Hz, chosen because in general higher firing rates provide for more accurate function approximation.

3.4. Integration and Compiling to Hardware

Putting the vision and control networks together is a simple matter of connecting the output of one to the input of the other in Nengo. As both networks were built using Loihi-type neurons, they are also prepared to be mapped to neuromorphic hardware. During the initial building and debugging process running on a CPU backend, which is the Nengo default, greatly expedited development. The NengoLoihi backend can then be used to compile the network to run on the Loihi (we could also use NengoOCL or NengoDL to compile to GPU and run directly in Nengo). We used a workstation with and Intel Core i7-6700K CPU @ 4.00 GHz × 8 with 32 GB RAM and GeForce GTX 1070/PCIe/SSE2 running Ubuntu 18.04, and the Intel Nahuku board with 32 Loihi chips (Davies et al., 2018) running NxSDK 0.9. In this example, we are only using one of the Loihi chips on the board. When the system is running, Nengo provides the interface between the simulator and the hardware, but all computations are run on the Loihi chip.

3.5. Performance

Figure 1–3 shows the (x, y) trajectory of the rover moving throughout the environment to six different targets in the environment, starting from the green "x." **Figure 1**–2 shows the perception and action signals from the network, with the target (x, y) estimated in the top figures in blue and the ground truth shown in red. The lower figure shows the steering and acceleration control signals generated by the network in blue with the ideal values in orange. As can be seen, the rover drives accurately (to within 50 cm) over the course of the trial. We have not performed extensive testing of the accuracy, and do not provide quantitative results as our purpose here is to focus on the methods used to develop the system. There is clearly significant room for improvement and extension to this work. Nevertheless, this simple example demonstrates the implementation of an end-to-end perception and action spiking neural network running on neuromorphic hardware. All code is available online at https://github.com/abr/neurorobotics-2020. We have provided full code to serve as a starting point for those interested in exploring neurorobotic solutions that can leverage embedded neuromorphic hardware for next generation systems.

4. NEUROROBOTIC ADAPTIVE ARM CONTROL

In this second example we augment an existing force controller with an adaptive neural network implemented on Loihi, using on-chip learning to control a Kinova Jaco^2 physical robot in a reaching task while it holds an unexpected weight. The existing control system generates joint torques using a standard proportional derivative (PD) operational space controller (OSC; Slotine et al., 1988), designed to move the hand along a target path. The adaptive controller adds an adaptive signal trained online to account for any unexpected forces affecting movement, which is tuned online. We compare performance of the adaptive control system to a non-adaptive PD OSC, and an industry standard proportional integrated-error derivative (PID) OSC.

We also compare the neuromorphic implementation with CPU and GPU implementations of the adaptive control system. We compare all systems in terms of accuracy, power use, and control loop update latency.

Operational space control relies on an accurate model of the arm dynamics to generate torques that will move the arm as desired. If the arm picks up an object, is subjected to external forces or perturbances, or wears down over time, the dynamics have changed and OSC performance will degrade unless the changes can be accurately modeled. In general, these perturbations can not be predicted in advance, so updating the controller on-the-fly is desirable. This is the purpose of the adaptive controller we use here, implemented as a neural network. Intuitively, the adaptive controller acts as a context sensitive integrated error term. Where standard integrated error terms (such as the I in PID control) apply the same learned error regardless of the current joint angles or velocities, the adaptive controller learns to account for errors specific to different arm states. The difference becomes significant in the control of highly non-linear systems, where the error that needs to be compensated for changes significantly with system state.

4.1. Interfacing With the Jaco^2 Robotic Arm

In this example Nengo is called as a sub-function of the PD OSC. The PD OSC itself is implemented in Python and runs on a workstation that interfaces with a physical Kinova Jaco^2 6 degrees-of-freedom (DOF) arm. The interface implemented is through the ABR Jaco^2 repository, and includes no neural network infrastructure. To integrate neural computation with this standard Python code, at each time step in the control model, Nengo is called to run the neural network for a single step. The control code sends feedback from the arm to Nengo and receives back an adaptive control signal to add into the outbound set of joint torques sent to the Jaco^2. Nengo takes care of running the neural network on a CPU, GPU, or the Loihi neuromorphic hardware.

4.2. Processing Sensory Feedback Using the NEF

In this application we are augmenting an existing control system with an adaptive control signal generated by a neural ensemble. The ensemble requires sensory feedback related to joint positions and velocities as input in order to compute the necessary correction to the control signal. Because the sensory feedback is a relatively low-dimensional signal (e.g., compared to image input), it does not need to be processed by a deep neural network before we can use it to generate a corrective control signal. The adaptive controller requires that the effects of the unexpected force are predictable given the input provided to the ensemble to be able to learn to compensate for unexpected forces affecting the arm. If, for example, the force affecting the arm was a function of joint angle, and the ensemble only had inputs related to joint velocity, then the network would not be able to adapt to the force.

Given that we have appropriate inputs, we further need to make sure that the neurons are sufficiently sensitive to different states of the arm relevant for compensating for the unexpected force. In the NEF, the neural tuning properties are determined by a combination of the neuron encoders (or "preferred" direction vectors), gains, and biases. This tuning determines which parts of the input state space are represented by the neural ensemble. In this section, we present considerations that determine how to appropriately pick these tuning curves for adaptive control in a highly non-linear state space.

In particular, we need to ensure that neurons are not active over a large part of state space. If this is the case, the compensatory signal they learn in one part of state space may incorrectly generalize to other parts. In contrast, if neurons are active over a small part of state space, then the compensatory signal they learn in one area will not affect what learning occurs in other parts of state space. Unsurprisingly, there is a trade-off between the specificity of neural responses and their generalization abilities. In arm control, because the dynamics are highly non-linear, we generally want to ensure that neurons in our ensemble are sensitive to localized parts of state space. We also do not want to waste neural resources. Consequently, neurons should only be sensitive to parts of state space that are actually explored by the arm. In other words, we do not want our population to include neurons that never become active.

To handle both of these issues, we need to carefully choose neural tuning curves, and hence NEF encoders. To optimize neurons for the relevant parts of the state space, we begin by subtracting the mean and then normalizing the input signals for each dimension given their joint limits. Next, we project into the D + 1 unit hypersphere, where D is the number of dimensions represented by the ensemble. By doing this it becomes possible to carefully control the range of values that cause neurons in the ensemble to respond. Because we have 6 joints and two input signals from each (i.e., position and velocity), we project the normalized signals onto the 13-dimensional (i.e., 12 + 1) unit hypersphere and use that as input to the ensemble.

Figure 2 illustrates how this allows us to control neural responses by considering the simpler case of projecting 2D into 3D. Assuming we have inputs in the −1 to 1 range along each dimension, the inputs are going to lie somewhere in the unit square. In the NEF, by default, each of our neurons will initially have an encoder that is a vector pointing from the origin to somewhere inside unit circle. Neurons with encoding vectors that point in the same direction will have similar firing rates responses to the same input, as shown in the top half of **Figure 2**. By projecting our encoding vectors and input signal into the 2 + 1 dimensional unit hypersphere, our neurons are still sensitive to all parts of the original 2D input signal, but co-linear encoding vectors can also generate distinct activity, as shown in the bottom half of **Figure 2**. For example, we can have neurons sensitive to an input signal of (0, 0.5) but not (0, 1), which was not possible with our 2D preferred direction vectors. Essentially, this method allows us to have neurons sensitive to more specific parts of state space.

4.3. Online Learning for Adaptive Control

The neural adaptive controller presented here is a neuromorphic implementation of the control system presented in DeWolf et al. (2016), where it is proven that this adaptive controller performs as well or better than a PID controller. The neural adaptive

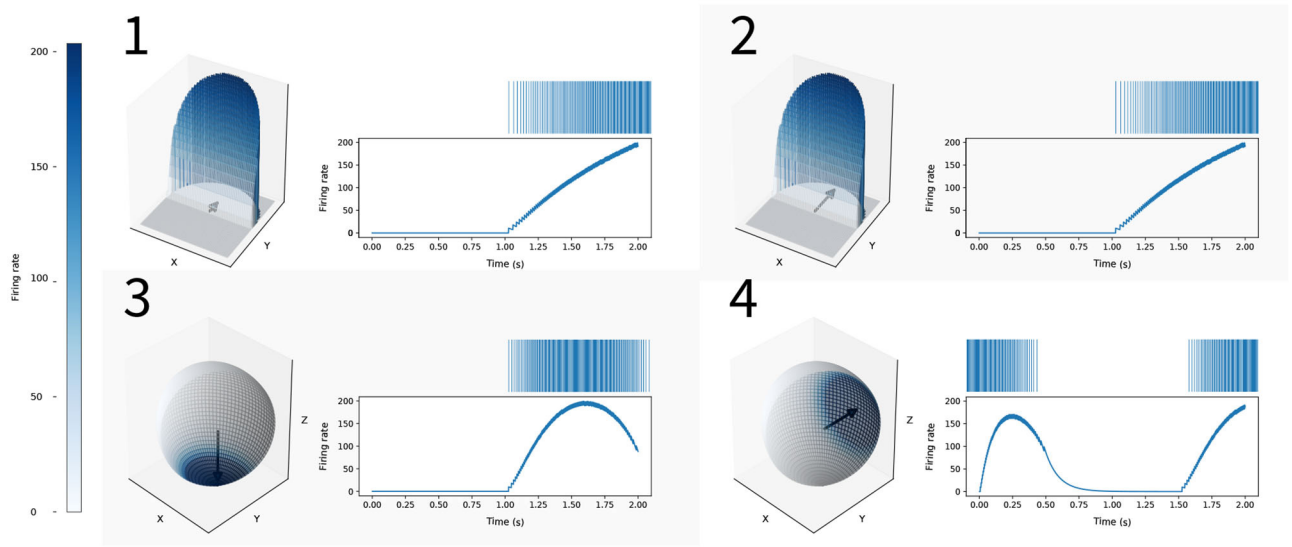


FIGURE 2 | An example of how increasing encoder dimensionality can create neurons that are more selectively responsive. On the right side of all figures the spike raster (top) and filtered output (bottom) is plotted, showing the firing rates of these neurons for the same 2D input signal that moves from [0, −1] to [0, 1], projected into 3D for **(3,4)**. In the top row we show two neurons with 2D encoding vectors (0, 0.5) [shown in **(1)** on the left] and (0, 1.0) [shown in **(2)** on the left]. The arrows in the figures represent the neuron's encoding vectors. The z-axis and color reflect the firing rates of these neurons given different (x, y) input. As can be seen, the activity of these neurons with co-linear encoding vectors is indistinguishable. In the bottom row, we project these same 2D encoders, (0, 0.5) [shown in **(3)** on the left] and (0, 1.0) [shown in **(4)** on the left], into 3D. The arrows in the figures represent the neuron's encoding vectors projected into 3D. The firing rates of the neurons given different (x, y, z) input is represented by color, showing that the neurons are responsive to different parts of 3D state space.

controller uses the Prescribed Error Sensitivity (PES; MacNeil and Eliasmith, 2011) learning rule, which is a local, spiking or non-spiking, error-driven Hebbian rule. The Loihi chip supports several different kinds of online learning, providing the ability to use microcode to define different kinds of rules. NengoLoihi implements the PES learning rule using this feature of the chip, which allows weight updates to be calculated on-chip. For the other hardware, core Nengo includes a definition of the PES rule.

The adaptive control signal is calculated via

$$\mathbf{u}_{\text{adapt}} = \mathbf{a}\,\mathbf{d}, \tag{3}$$

where $\mathbf{a}$ is the vector of neural activities (i.e., filtered neural spike trains) and $\mathbf{d}$ denotes a vector of "decoders" which are the output weights from the ensemble. The resulting $\mathbf{u}_{adapt}$ is a vector of the same dimensionality as the OSC control signal. We initialize $\mathbf{d}$ to a vector of zeros, and use the learning rule

$$\Delta\mathbf{d} = -\kappa\ \mathbf{a} \otimes \mathbf{u}, \tag{4}$$

to update the decoder weights, where κ is a learning rate, $\mathbf{u}$ is the OSC's outbound control signal (acting as the error in the PES rule), and $\otimes$ denotes the outer product. This training signal, $\mathbf{u}$, was chosen based on Lyapunov stability analysis. Details, derivation, and proof of stability of this adaptive neural controller are provided in DeWolf et al. (2016).

In the system diagram shown in **Figure 3**–1, the hollow triangle denotes the connection providing the training signal, $\mathbf{u}$, for the learning rule.

4.4. Compiling to Neuromorphic Hardware

The NengoLoihi backend is used to instantiate the neural ensemble on the Loihi with the parameters discussed above, and implement the PES learning rule on-chip. Core Nengo is used for the CPU implementation and the NengoOCL backend package is used for the GPU implementation. In this example we used a workstation with and Intel Core i7-6700K CPU @ 4.00 GHz × 8 with 32 GB RAM and GeForce GTX 1070/PCIe/SSE2 running Ubuntu 18.04 and the Intel Kapoho Bay board with 8 Loihi chips running NxSDK 0.9 In this example, we are only using one of the Loihi chips on the board.

4.5. Performance

Figure 3–2 shows the arm performing a reaching task while holding an unexpected two pound mass. In this task the arm repeatedly starts from the same position and reaches to the same target 50 times, with continuous learning between reaches. We perform the 50 reaches with each controller five times, using different randomly generated neuron ensemble parameters (such as bias and maximum firing rates), and calculate the mean error and 95% confidence intervals. The adaptive controller is run while simulating the neurons on the Loihi, and the CPU and GPU of our workstation (specifications are in section 4.4), and compared against a standard PID controller running on the same workstation. We normalize our performance results using the performance of a PD operational space controller. We consider that controller reaching under normal conditions with nothing in the hand as 0% error, and the results of that controller reaching while holding the unaccounted-for two pound weight as 100%

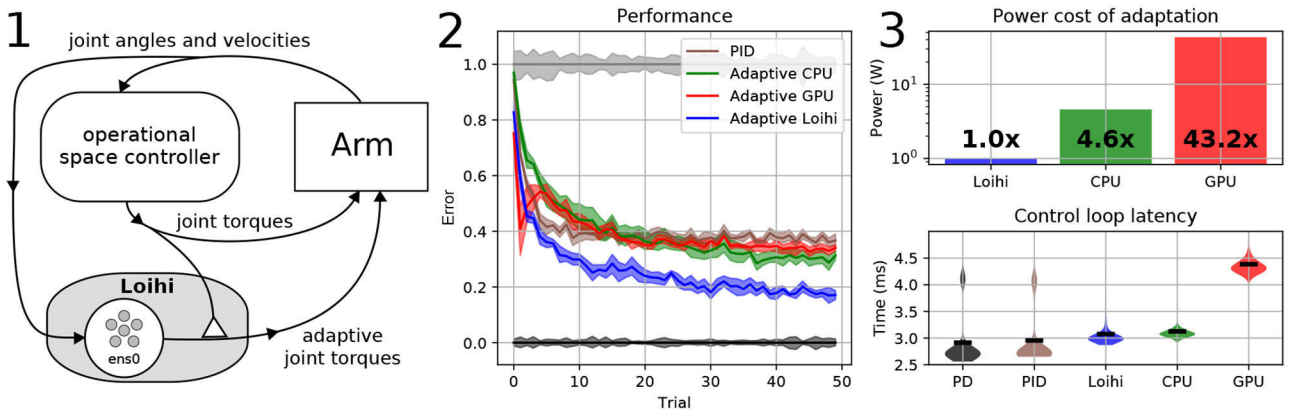


FIGURE 3 | A neurorobotic adaptive arm controller. **(1)** The system diagram of the neurorobotic adaptive control system. The hollow triangle denotes the connection providing the training signal to the learning rule. Note that Nengo is only used to run the neural part of the system. The "ens" prefix denotes NEF layers. **(2)** Performance error while reaching to a target while holding an unexpected two pound mass. Black: PD controller with no extra mass, used as a 0% reference error. Gray: PD controller reaching while holding the two pound mass, used as 100% reference error. Brown: PID controller reduces error to 65.3%. Green: Adaptive controller CPU implementation with 1,000 neurons reduces error to 37.6%. Red: Adaptive controller GPU implementation with 1,000 neurons reduces error to 36.2%. The initially high error is due to the increased latency of the GPU implementation. Blue: Adaptive controller Loihi implementation with 1,000 neurons reduces error to 26.7%. Results are averaged over five sets of 50 trials with 95% confidence intervals shown. The adaptive controller demonstrates a 2.45 times improvement in accuracy over PID, and 1.49 and 1.57 times improvement over the CPU and GPU adaptive controller implementations, respectively. **(3)** Top: A power comparison between adaptive controller implementations. Running adaptation on the CPU and GPU requires 4.6× and 43.2× more power than Loihi. Bottom: Latency measurements of the controllers. PD: 2.91 ms, PID: 2.95 ms, Adaptive Loihi: 3.08 ms, Adaptive CPU: 3.13 ms, Adaptive GPU: 4.38 ms.

error. As can be seen, the Loihi system outperforms all other controllers after 50 trials of training. Unsurprisingly, the CPU and GPU perform similarly, and better than the PID controller.

Figure 3–3 shows the power and latency measurements of the controllers during the task, where the neuromorphic implementation consumes the least energy of the adaptive controllers, with a minimal increase in latency compared to the non-adaptive controllers. Specifically, the CPU uses 4.6× more power, and the GPU 43.2× more. As well, the CPU is a similar latency (i.e., 2% slower), while the GPU is 42% slower than the Loihi.

To measure the power use of the CPU, we used the software package `s-tui`, available online at https://github.com/amanusk/s-tui. To measure the power use of the GPU, we used the `nvidia-smi` software. To measure the Loihi power use, we used the Linear Tech DC1613A dongle and LTpowerPlay software, which provides current and voltage measurements for the chip's two power supplies, from which we calculated the total power use.

5. DISCUSSION

We have demonstrated how to take advantage of neuromorphic technology to fully implement or augment existing robotic control systems. In particular, we showed how a set of tools in the Nengo ecosystem allows efficient execution of four central tasks for building neurorobotic systems.

1. The NengoInterfaces library provides an easy API for interfacing with the Mujoco simulation, used in the first example, both for sending in control signals and receiving feedback. While the second example is not directly providing simple API access, it illustrates the flexibility of Nengo to be incorporated into already developed Python programs and interfaces.
2. In the rover example, we show how NengoDL provides a natural way to integrate Keras and TensorFlow models. For non-neural perception algorithms, Python code can be directly executed from Nengo or the code can be run outside Nengo and sent into a Nengo model, as is done in the adaptive arm example.
3. We illustrated how a circuit design that solves the problem of interest can be implemented in a neural network using Nengo, providing white-box neural network systems. In the first example this was shown with the rover control network that steered and drove the rover to the target, and the second example showed the use of a vector-space training signal with stability guarantees to implement non-linear adaptive control.
4. The Nengo development toolkit allows users to compile their model to run on multiple different hardware platforms, including CPU (Bekolay et al., 2014), GPU (Rasmussen, 2019), FPGA (Morcos, 2019), Intel's Loihi (Hunsberger et al., 2018), and SpiNNaker (Mundy et al., 2015).

We have made all of the code used in these examples publicly available, to provide practical, reproducible examples for the community. We believe this set of tools and examples helps address the core challenge of making neurorobotic systems easier to build, and a wide variety of architectures easier to explore.

In the first example, while we did not benchmark or quantitatively characterize the result, it provides a demonstration of how our chosen tools allow the development of complete perception-action systems for neurorobotics. In particular,

it demonstrated how to couple white box (i.e., NEF) and black box (i.e., DNN) techniques and implement them on a single underlying spiking neuromorphic hardware platform (i.e., Loihi).

Nengo is unique in its ability to make such integration easily accessible. Nengo has several advantages in this context: (a) it is not vendor specific, and supports hardware from several sources; (b) it allows a high-level model specification for easy portability, while also allowing hardware specific details to be incorporated (e.g., via its configuration system); and (c) Nengo removes the need to have detailed knowledge about SNNs, neuromorphic hardware, simulator interfaces, embedded programming, and so on, while allowing those with such knowledge to leverage it (e.g., by easily defining new neuron models, using the configuration system, building new hardware specific backends, and so on).

Our second example provided a more quantitative characterization of the advantages of neurorobotics. From a tools perspective, it demonstrated how Nengo can be integrated into existing systems and run the same neural model across multiple kinds of hardware. But, more importantly, this example demonstrates the kinds of advantages we expect from neuromorphics: an increase in speed and accuracy, and several-fold decrease in power compared to traditional hardware. As is well-established, low latency and energy efficiency are critical for many mobile robotics applications.

Possible extensions to the examples provided here are many and varied. Perhaps one of the more obvious ones is to implement the entire adaptive controller on neuromorphic hardware. To the best of our knowledge this has only been done with a 3-link planar arm (DeWolf et al., 2016). Since the adaptive controller has performance guarantees, and the white box methods of the NEF allow us to implement it on neuromorphic hardware while preserving those guarantees, a full implementation would be a rare example of an adaptive, fully neurorobotic controller with clear performance guarantees.

While we believe the Nengo ecosystem is useful for the development of neurorobotic systems, there remain a variety of challenges and directions for future development that stand to improve it. For instance, Nengo backends that target non-spiking AI acceleration hardware, such as Google's Coral chip, would expand the community able to use the methods we have discussed because spiking neuromorphic hardware, such as Intel's Loihi chip, is not commercially or otherwise widely available. Extending the interfaces offered by the NengoInterfaces package to improve accessibility, as well as offering the same automatic conversion from DNNs to SNNs for PyTorch users also remains important future work. Perhaps most importantly, continuing to increase the number of available tutorials, ready-to-use models, and online examples is critical to reducing the startup overhead for new users and better supporting the neurorobotics, neural networks, and edge AI communities.

In conclusion, the Nengo ecosystem makes it possible for users to quickly develop applications for neuromorphic hardware, while taking advantage of already developed neural or non-neural machine learning solutions. We have shown two examples that demonstrate how the ecosystem can be used to address four core stages of the development workflow. We encourage interested researchers to use the code and tools that we have made available, and look forward to exploring the vast space of robust, embedded neurorobotics systems with the emerging research community.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found in the article/supplementary material.

AUTHOR CONTRIBUTIONS

TD directed the project. TD, PJ, and CE implemented the examples. PJ and CE collected the data for the arm example. TD wrote the manuscript. CE provided research guidance and edited the manuscript. All authors contributed to the article and approved the submitted version.

FUNDING

The NengoLoihi interface development was funded by Intel. All other funding was provided by Applied Brain Research, Inc.

ACKNOWLEDGMENTS

The authors would like to thank Daniel Rasmussen, Eric Hunsberger, and Xuan Choo for their help in the development of this project. We would also like to thank Intel for access to the Nahuku board and Kapoho Bay used in the examples, and Mike Davies for reviewing an early draft of the paper.

REFERENCES

Bam Shrestha, S., and Orchard, G. (2018). Slayer: spike layer error reassignment in time. *arXiv* 1810.

Bekolay, T. (2011). *Learning in large-scale spiking neural networks* (Master's thesis), University of Waterloo, Waterloo, ON, Canada.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., et al. (2014). Nengo: a python tool for building large-scale functional brain models. *Front. Neuroinform.* 7:48. doi: 10.3389/fninf.2013.00048

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Davison, A. P., Brüderle, D., Eppler, J. M., Kremkow, J., Muller, E., Pecevski, D., et al. (2009). Pynn: a common interface for neuronal network simulators. *Front. Neuroinform.* 2:11. doi: 10.3389/neuro.11.011.2008

DeWolf, T., Stewart, T. C., Slotine, J.-J., and Eliasmith, C. (2016). A spiking neural model of adaptive arm control. *Proc. R. Soc. B Biol. Sci.* 283:20162134. doi: 10.1098/rspb.2016.2134

Eliasmith, C., and Anderson, C. H. (2003). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT Press.

Falotico, E., Vannucci, L., Ambrosano, A., Albanese, U., Ulbrich, S., Vasquez Tieck, J. C., et al. (2017). Connecting artificial brains to robots in a comprehensive simulation framework: the neurorobotics platform. *Front. Neurorobot*. 11:2. doi: 10.3389/fnbot.2017.00002

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The spinnaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gewaltig, M.-O., and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2:1430. doi: 10.4249/scholarpedia.1430

Goodman, D. F., and Brette, R. (2008). Brian: a simulator for spiking neural networks in python. *Front. Neuroinform*. 2:5. doi: 10.3389/neuro.11.005.2008

Gutierrez-Galan, D., Dominguez-Morales, J. P., Perez-Peña, F., Jimenez-Fernandez, A., and Linares-Barranco, A. (2020). Neuropod: a real-time neuromorphic spiking CPG applied to robotics. *Neurocomputing* 381, 10–19. doi: 10.1016/j.neucom.2019.11.007

Hines, M. L., and Carnevale, N. T. (1997). The neuron simulation environment. *Neural Comput*. 9, 1179–1209. doi: 10.1162/neco.1997.9.6.1179

Hunsberger, E., Bekolay, T., Rasmussen, D., Voelker, A., Stewart, T., Patel, K., et al. (2018). *Nengoloihi*. Available online at: https://github.com/nengo/nengo-loihi

Hwu, T., Isbell, J., Oros, N., and Krichmar, J. (2017). "A self-driving robot using deep convolutional neural networks on neuromorphic hardware," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK: IEEE), 635–641. doi: 10.1109/IJCNN.2017.7965912

Kreiser, R., Waibel, G., Sandamirskaya, Y., and Renner, A. (2019). "Self-calibration and learning on chip: towards neuromorphic robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2019)* (Macao).

MacNeil, D., and Eliasmith, C. (2011). Fine-tuning and the stability of recurrent neural networks. *PLoS ONE* 6:e0022885. doi: 10.1371/journal.pone.0022885

Markram, H. (2012). The human brain project. *Sci. Am*. 306, 50–55. doi: 10.1038/scientificamerican0612-50

Mead, C. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636. doi: 10.1109/5.58356

Morcos, B. (2019). *NengoFPGA: an FPGA backend for the nengo neural simulator* (Master's thesis), University of Waterloo, Waterloo, ON, Canada.

Mundy, A., Knight, J., Stewart, T. C., and Furber, S. (2015). "An efficient spinnaker implementation of the neural engineering framework," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280390

Rasmussen, D. (2019). NengoDL: combining deep learning and neuromorphic modelling methods. *Neuroinformatics* 17, 611–628. doi: 10.1007/s12021-019-09424-z

Ray, A., McGrew, B., Schneider, J., Ho, J., Welinder, P., Zaremba, W., et al. (2020). *mujoco-py*. Available online at: https://github.com/openai/mujoco-py

Sawada, J., Akopyan, F., Cassidy, A. S., Taba, B., Debole, M. V., Datta, P., et al. (2016). "Truenorth ecosystem for brain-inspired computing: scalable systems, software, and applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, UT: IEEE), 130–141. doi: 10.1109/SC.2016.11

Slotine, J.-J. E., Khatib, O., and Ruth, D. (1988). Robust control in operational space for goal-positioned manipulator tasks. *Int. J. Robot. Autom*. 3, 28–34.

Stagsted, R. K., Vitale, A., Binz, J., Larsen, L. B., Sandarmirskaya, Y., and Renner, A. (2020). "Towards neuromorphic control: a spiking neural network based PID controller for UAV," in *Robotics: Science and Systems* (Corvalis, OR). doi: 10.15607/RSS.2020.XVI.074

Taunyazov, T., Sng, W., See, H. H., Lim, B., Kuan, J., Ansari, A. F., et al. (2020). Event-driven visual-tactile sensing and learning for robots. *Perception* 4:5. doi: 10.15607/RSS.2020.XVI.020

Todorov, E., Erez, T., and Tassa, Y. (2012). "Mujoco: a physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Vilamoura: IEEE), 5026–5033. doi: 10.1109/IROS.2012.6386109

Conflict of Interest: The authors are employees of Applied Brain Research, Inc., which develops and distributes Nengo free for academic and non-commercial use. The authors declare that this study received funding from Applied Brain Research, Inc. The funder had the following involvement with the study: paid for publication fees. The NengoLoihi interface development was funded by Intel.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 19 October 2020
doi: 10.3389/fnbot.2020.568319

Perception Understanding Action: Adding Understanding to the Perception Action Cycle With Spiking Segmentation

Paul Kirkland[1]*, *Gaetano Di Caterina*[1], *John Soraghan*[1] and *George Matich*[2]

[1] Neuromorphic Sensor Signal Processing Lab, Centre for Image and Signal Processing, Electrical and Electronic Engineering, University of Strathclyde, Glasgow, United Kingdom, [2] Leonardo MW Ltd., London, United Kingdom

Traditionally the Perception Action cycle is the first stage of building an autonomous robotic system and a practical way to implement a low latency reactive system within a low Size, Weight and Power (SWaP) package. However, within complex scenarios, this method can lack contextual understanding about the scene, such as object recognition-based tracking or system attention. Object detection, identification and tracking along with semantic segmentation and attention are all modern computer vision tasks in which Convolutional Neural Networks (CNN) have shown significant success, although such networks often have a large computational overhead and power requirements, which are not ideal in smaller robotics tasks. Furthermore, cloud computing and massively parallel processing like in Graphic Processing Units (GPUs) are outside the specification of many tasks due to their respective latency and SWaP constraints. In response to this, Spiking Convolutional Neural Networks (SCNNs) look to provide the feature extraction benefits of CNNs, while maintaining low latency and power overhead thanks to their asynchronous spiking event-based processing. A novel Neuromorphic Perception Understanding Action (PUA) system is presented, that aims to combine the feature extraction benefits of CNNs with low latency processing of SCNNs. The PUA utilizes a Neuromorphic Vision Sensor for Perception that facilitates asynchronous processing within a Spiking fully Convolutional Neural Network (SpikeCNN) to provide semantic segmentation and Understanding of the scene. The output is fed to a spiking control system providing Actions. With this approach, the aim is to bring features of deep learning into the lower levels of autonomous robotics, while maintaining a biologically plausible STDP rule throughout the learned encoding part of the network. The network will be shown to provide a more robust and predictable management of spiking activity with an improved thresholding response. The reported experiments show that this system can deliver robust results of over 96 and 81% for accuracy and Intersection over Union, ensuring such a system can be successfully used within object recognition, classification and tracking problem. This demonstrates that the attention of the system can be tracked accurately, while the asynchronous processing means the controller can give precise track updates with minimal latency.

Keywords: spiking, convolution, segmentation, tracking, STDP, neuromorphic, neural network, asynchronous

OPEN ACCESS

Edited by:
Joe Hays,
United States Naval Research Laboratory, United States

Reviewed by:
Garrick Orchard,
Intel, United States
Sumit Bam Shrestha,
*Institute for Infocomm Research (A*STAR), Singapore*
Ed Lawson,
United States Naval Research Laboratory, United States

***Correspondence:**
Paul Kirkland
paul.kirkland@strath.ac.uk

Received: *31 May 2020*
Accepted: *20 October 2020*
Published: *19 October 2020*

Citation:
Kirkland P, Di Caterina G, Soraghan J and Matich G (2020) Perception Understanding Action: Adding Understanding to the Perception Action Cycle With Spiking Segmentation. Front. Neurorobot. 14:568319. doi: 10.3389/fnbot.2020.568319

1. INTRODUCTION

Understanding and reasoning is a fundamental process in most biological perception action cycles. It is through understanding of our visual perception that helps to inform our basic decision-making processes like 'friend or foe" and "edible or inedible," which ultimately is key to progression or survival. Adding some level of understanding into this cycle can help to deliver a robust robotic system that could perform more complex variations of simple following and tracking tasks. Computer Vision (CV) has made this understanding a reality for robotics systems, with traditional CV methods providing simple feature extraction at low latency, or modern deep learning-based Convolutional Neural Networks (CNN) providing state of the art results in almost every task with high precision and accuracy, but at the cost of higher latency and computation throughput. This often leaves the CNN out of the reach of the small robotic system world due to its lower power and computational specifications. Modern research looks toward biological inspirations to help solve these tasks, by bringing forward neuromorphic robotics, which seeks to merge the computational advantages of system, such as the neuromorphic event-based vision sensor (NVS) and neuromorphic processors together, combined with Spiking Neural Network (SNN) which can allow for processing and control system structures. Typically a robotic system in this domain might aim to reach a Perception, Cognition, Action cycle, while the simpler approach of Understanding as a step toward cognition could be realized in an easier way, using the Perception Understanding Action (PUA) cycle as a stepping stone toward this goal.

Perception using neuromorphic vision sensors has become a promising solution. An NVS, as for example the Dynamic Vision Sensor (DVS) (Lichtsteiner et al., 2008), mimics the biological retina to generate spikes in the order of microseconds, in response to the pixel-level changes of brightness caused by motion. NVSs offer significant advantages over standard frame-based cameras, with no motion blur, a high dynamic range, and latency in the order of microseconds (Gehrig et al., 2018). Hence, the NVS is suitable for working under poor light conditions and on high-speed mobile platforms. There has been considerable research detailing the advantages of using an NVS in various vision tasks, such as high-speed target tracking (Lagorce et al., 2015; Mueggler et al., 2017) and object recognition (Kheradpisheh et al., 2018). Moreover, due to the fact that a pixel of an NVS is a silicon retinal neuron represented by an asynchronously generated spiking impulse, this can be directly fed into Spiking Neural Networks (SNNs) as input spikes for implementing target detecting and tracking in a faster and more neuromorphic approach.

Understanding through asynchronous spiking event-based computations like SNNs, often seen as the low latency biologically inspired alternative to CNNs, could provide an alternative solution to tracking and segmentation problems, through the ability to only compute on the currently active parts of the network, which in comparison to Artificial Neural Networks (ANN) and CNNs can require orders of magnitude less power consumption (Park et al., 2014). SNNs differ from normal computation processing and take inspiration from closer to biology, where expensive memory access operations are negated due to computations and memory being exclusively local (Paugam-Moisy and Bohte, 2012). Instead of using numerical representations like traditional methods, SNNs use spikes to transmit information with a key emphasis on the timing of those spikes. Several methods exist to train SNNs, with recent implementations seeing a conversion from CNN to SNN (Cao et al., 2015; Hunsberger and Eliasmith, 2015; Kim et al., 2019; Sengupta et al., 2019) yield promising results and open SNN architectures to the wider Machine and Deep Learning (ML-DL) audience. However, this method is still burdened with the training computational overhead and does little to utilize the efficiency of event driven computations. The SNN's Spike Time Dependent Plasticity (STDP) and spike-based back-propagation learning have been demonstrated to capture hierarchical features in SpikeCNNs (Masquelier and Thorpe, 2007; O'Connor et al., 2013; Panda et al., 2017; Kheradpisheh et al., 2018; Masquelier and Kheradpisheh, 2018; Falez et al., 2019). Both of these methods better equip the network to deal with event driven sensors, where the significant gains over CNNs could be realized.

This work aims to build on the already successful perception-action models (Nishiwaki et al., 2003; Xie, 2003; Bohg et al., 2017; Masuta et al., 2017) and add some semantic understanding to the robotic system. With image segmentation seen as a critical low-level visual routine for robot perception, a semantic understanding of the scene can play an important role for robots to understand the context in their operational environment. This context can then lead to a change in the action that could be undertaken. In this article, we show how using a spiking fully convolution neural network for event-based segmentation of a neuromorphic vision sensor can lead to accurate perception and tracking capabilities with low latency and computation overhead. Leveraging this spiking event-based segmentation framework to feed a spiking control system allows the low latency to continue from the perception to the action.

The PUA system presented builds on SpikeSEG, a spiking segmentation network from previous work (Kirkland et al., 2020), and extends it with a systematic approach to spike-based object recognition with tracking, lateral inhibition classifications, a new thresholding mechanism and modification to STDP learning process. Moreover, differently from Kirkland et al. (2020), the novel work presented is applied to a different application context, i.e., object recognition with attention. In light of this the novel contributions of this work include:

- SpikeSEGs segmentation output is integrated into a spike-based control system to produce the Perception-Understanding-Action system where the segmentation infers the attention of the system to allow controller track updates.
- The revised network includes more features to enhance the segmentation ability, including:

- Lateral inhibition pseudo classification mechanism for semantic segmentation-based attention.
- A new Pre-Empt then Adapt Thresholding (PEAT) approach designed to deal with potentially noisy, corrupt or adversarial inputs.
- A modification to the STDP learning rules to include feature pruning (resetting) if under/over utilized.

The rest of the paper is organized as follows. Section 2 reviews related research topics covering each of the PUA framework individual sections. Section 3 presents the methodology, with an insight to each of the proposed system components. The results are detailed in section 4 and section 5 provides the conclusion.

2. RELATED WORK

The allure of low latency object recognition and localization has brought the attractive features of the NVS (mainly the DVS) to the forefront of research. Early low latency control examples, such as the Pencil Balancer (Conradt et al., 2009) and the Robotic Goalie (Delbruck and Lang, 2013), help to highlight the latency advantages that an NVS can provide. Exploiting the sparse and asynchronous output of the sensor allow successful applications to these low latency reactive tasks. However, both systems fall short of fully capitalizing on the event-driven asynchronous output, through a processing and control regime of similar nature.

The concept of exploiting the NVS low latency continues into object tracking. Low latency tracking relies upon robust feature detection, with geometric shapes being ideal features to detect. A number of methods have been implemented successfully, such as geometric constraints (Clady et al., 2015) along with advanced corner detection methods, as for example Harris (Vasco et al., 2016) and FAST (Mueggler et al., 2017). The use of more complex features, such as Gaussians, Gabors, and other hand crafted kernels (Lagorce et al., 2015) provides a pathway to modern Convolutional Neural Network feature extraction approaches (Li and Shi, 2019), that implement a correlation filter from the learned features of the CNN. This allows a multi-level approach whereby correlations of intermediate layers can also be performed to improve the inherent latency disadvantage of the CNN approach, albeit with an accuracy trade-off.

Spiking Neural Networks have seen success with NVS data used for object detection and classification (Bichler et al., 2012; Stromatias et al., 2017; Paulun et al., 2018). Recent work has implemented Spiking Convolutional Neural Networks (Kheradpisheh et al., 2018; Falez et al., 2019) with NVS-like data created using a difference of Gaussian filter, suggesting the combination of SNNs and Deep Learning could yield successful results (Tavanaei et al., 2019). SNNs have also been utilized for tracking with an NVS through implementations inspired by the Hough Transform (Wiesmann et al., 2012; Seifozzakerini et al., 2016; Jiang et al., 2019), to be able to detect and track lines and circles. Spiking Neural Networks can also be utilized to implement control systems, from simple altitude control (Levy, 2020) to an adaptive robotic arm controller (DeWolf et al., 2016). Ultimately the majority of research only utilizes one aspect of the SNN, either processing or control. Even though SNNs have been shown to implement a full perception cognition action cycle with Spaun (Eliasmith et al., 2012), underpinning the ideology of a fully spike-based neuromorphic system similar to that proposed with the Perception Understanding Action framework in this paper.

3. METHODOLOGY

3.1. Perception-Understanding-Action Framework

The Perception-Understanding-Action framework specifies how the system will utilize the asynchronous event driven nature of the Neuromorphic spiking domain, and it is illustrated in **Figure 1**. In the Perception block, the NVS is used to sparsely and asynchronously encode the luminosity changes within the scene. In the Understanding block, inputs are understood through the use of the Encoder-Decoder SpikeCNN [SpikeSEG (Kirkland et al., 2020)] contextualizing and building understanding of the scene through semantic segmentation. In the Action block, the segmented output is used to provide an input to the spike counters at the edge of the field of view, allowing a simplistic semantic tracking controller to be realized. This control output would then be able to influence motors or actuators to allow an asynchronous end to end Neuromorphic system. This system aims to provide a low latency competitor to the Perception Action robotic system where the sensor input is directly fed to the controller, while providing an upgraded feature representation to the more complex line and edge detection-based approaches. The system can even provide benefits or replace some computer vision-based robotic tasks which utilize CNNs for complex feature extraction, while providing lower latency and computational overhead. Furthermore, compared to the CNN, the SCNN provides a more readily understandable processing stage, where features are sparse and more visually interpretable.

3.2. Perception

A key element in producing a low latency system with a low computational overhead is to have a sensor that can exploit the sparse and asynchronous computational elements of an SNN while still giving a detailed recording of the scene. Neuromorphic Vision Sensors (NVS) *(event-based Vision Sensors*) (Lichtsteiner et al., 2008; Brandli et al., 2014) have recently become more popular and widespread. These camera-like devices are bio-inspired vision sensors that attempt to emulate the functioning of biological retinas. They differ from conventional cameras in that, they don't record all the information the sensor sees at set intervals. Instead these sensors produce an output only when a change is detected. This in turn means they are capturing the luminosity at a set point in time, meaning a continuous temporal derivative of luminosity is output. Whenever this happens, an event $e = [x, y, ts, p]$ is created, indicating the x and y position along with the time ts at which the change has been detected and its polarity, where $p \in \{1, -1\}$ is a positive or negative change in brightness. This change in operation not only increases the sparsity of the

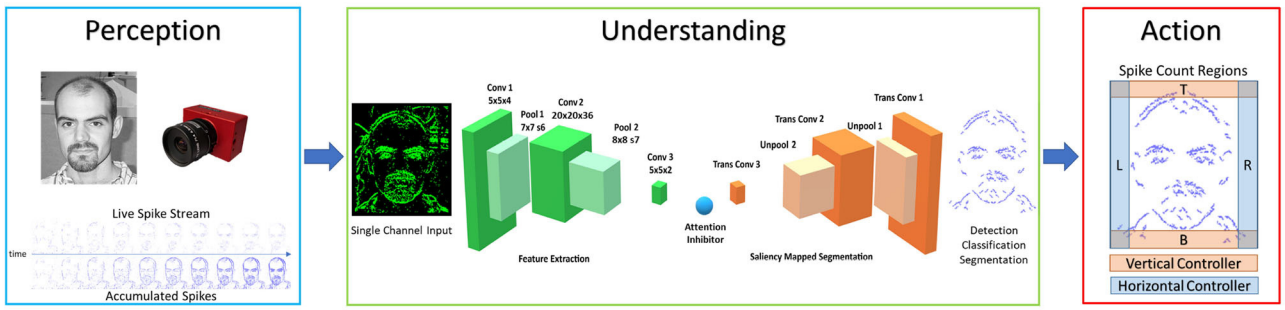


FIGURE 1 | Perception understanding action framework, with internal system diagrams showing the perception input [image from Caltech Dataset (Li et al., 2004), the Understanding network SpikeSEG (Kirkland et al., 2020), and the Action controller method].

signal but allows for it to output asynchronously. Resulting in microsecond temporal resolution and considerably lower power consumption and bandwidth. These parameters make the NVS an ideal candidate for object tracking, especially of fast moving objects (Delbruck and Lichtsteiner, 2007; Glover and Bartolozzi, 2017), however many methods are still yet to utilize this spiking sensor within a match spiking processing, such as SNNs.

3.3. Understanding Through Spiking Segmentation

The Understanding of this system is inferred from the semantic segmentation operation carried out by the SpikeSEG network (Kirkland et al., 2020), seen in **Figure 1** within the Understanding block. The SpikeSEG segmentation network has received a number of improvements and upgrades along with its integration within the PUA framework.

3.3.1. Network Architecture

The network architecture illustrated within **Figure 1** (Understanding) is made up of two main sections seen in green and orange, that relate to the encoding and decoding layers, respectively. The network is split into these two sections where training only occurs on the encoding side, while the weights are tied to the mirrored decoding layers. This allows a integrate and fire neuron with layer-wise STDP mechanism with adaptive thresholding and pruning to be used to help compress the representation of the input to allow the decoding layer to segment the image based on the middle pseudo classification layers. This encoding-decoding structure symbolizes a feature extraction then shape generation process. The learning of the encoding process aims to extract common spatial structures as useful features, then decode those learned features over to the shape generation process, unraveling the latent space classification representation but with a reduction in spike due to the max pooling process. The network has nine computational layers *(Conv1-Pool1-Conv2-Pool2-Conv3-TransConv3-UnPool2-TransConv2-UnPool1-TransConv1)* as seen in **Figure 1**. Between the Conv3 and TransConv3 layers, there is a user-defined attention inhibition mechanism, which can operate in two manners: No Inhibition, which allows semantic segmentation of all recognized classes from the pseudo classification layer; or With Inhibition, that only allows one class to propagate forward to the decoding layers. This attention not only provides a reduction in the amount of computation, but also simplifies the input to the controller.

3.3.2. Encoding

The encoding part of this system is derived from a basic SpikeCNN with a simplified STDP learning mechanism (Kheradpisheh et al., 2018). To allow the network to better suit the framework and encoding decoding structure a number of modification are applied. As the structure of the network is now fully convolutional there is no longer a requirement for a global pooling layer for classification. Instead the final convolution layer is utilized as a mock classifier by mapping the number of known classes to the number of kernel used for feature learning. This method is also used to help the interperitability of the system as having one kernel per classes allows for better visualization of the network features. Through the use of a modified STDP rule and adaptive neuron thresholding, the encoder aims to capture the reoccurring features that are most salient through the event stream inputs. The input events are fed into the network via a temporal buffering stage, to allow for a more plausible current computing solution, such as on the Intel Loihi Neuromorphic chip (Davies et al., 2018), while ideally they would just be a constant stream. To internally mimic the continuous data, 10 ms of event data is buffered into 10 steps, representing 1 ms each (this value of 10 ms is chosen to empirical testing and based on the input spike count of the N-Caltech Dataset); this input data stream is shown in **Figure 2**. **Figure 2**, also illustrates what 1 ms of data looks like over the 10 ms (A) and how it looks if accumulated over 10 ms (B). **Figure 2** then demonstrates how added noise affects the input stream, repeating the images in **Figures 2A,B** with noise in 1 ms steps in (C) and accumulated over 10 ms in (D). For each time step in the encoding processing, a spike activity map Sk_{mt} is also produced, where m is the feature map and t is the time step. This allows an account of the exact spatial time location of each active pixel used in the encoding processing, which helps allow the decoder to map these active areas back into the pixel space.

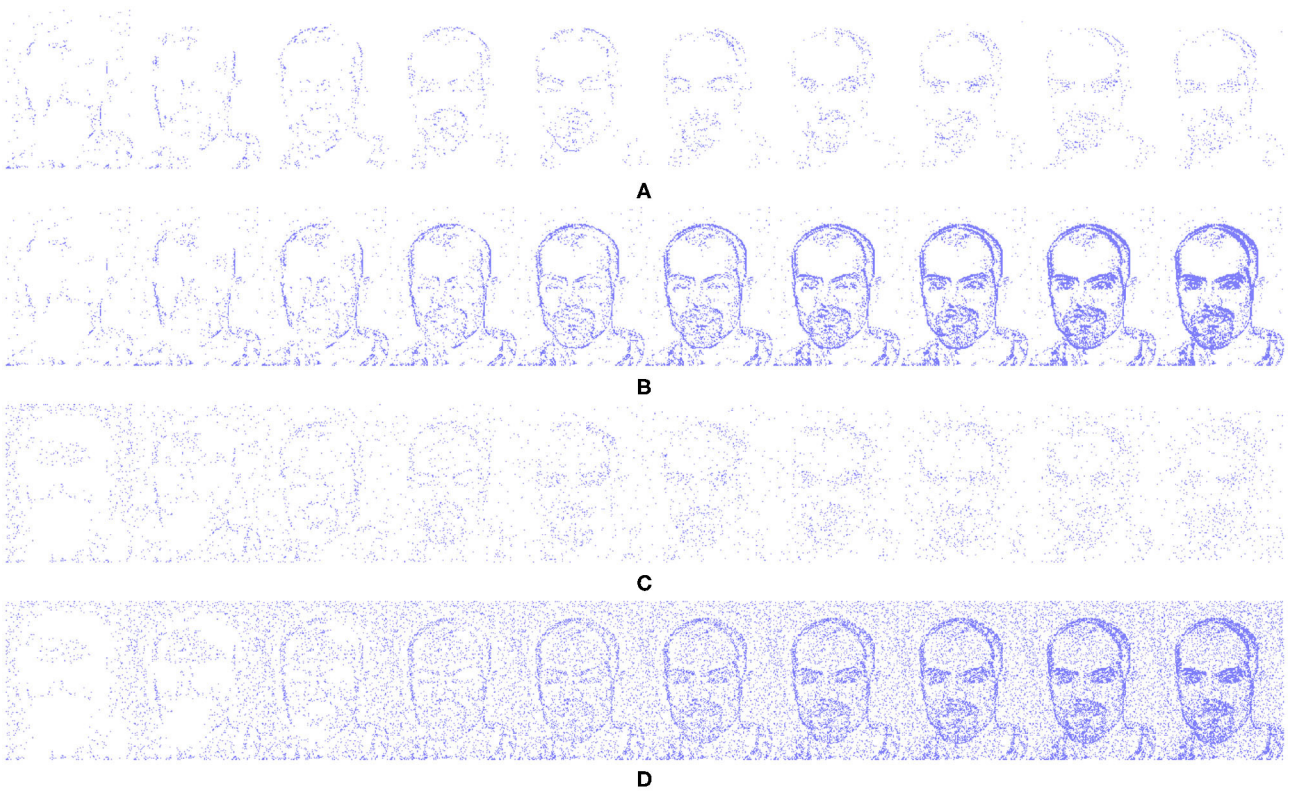


FIGURE 2 | Input event streams from N-Caltech Dataset "Face," with **(A,B)** showing a 10 ms clip over 10 steps going from left to right. **(A)** Showing the input to the network per step and **(B)** showing the accumulated inputs for easier visualization. **(C,D)** Show a 10 ms clip over 10 steps with additive noise to show how extra noise affects the input stream, with **(C)** showing per step and **(D)** showing accumulated.

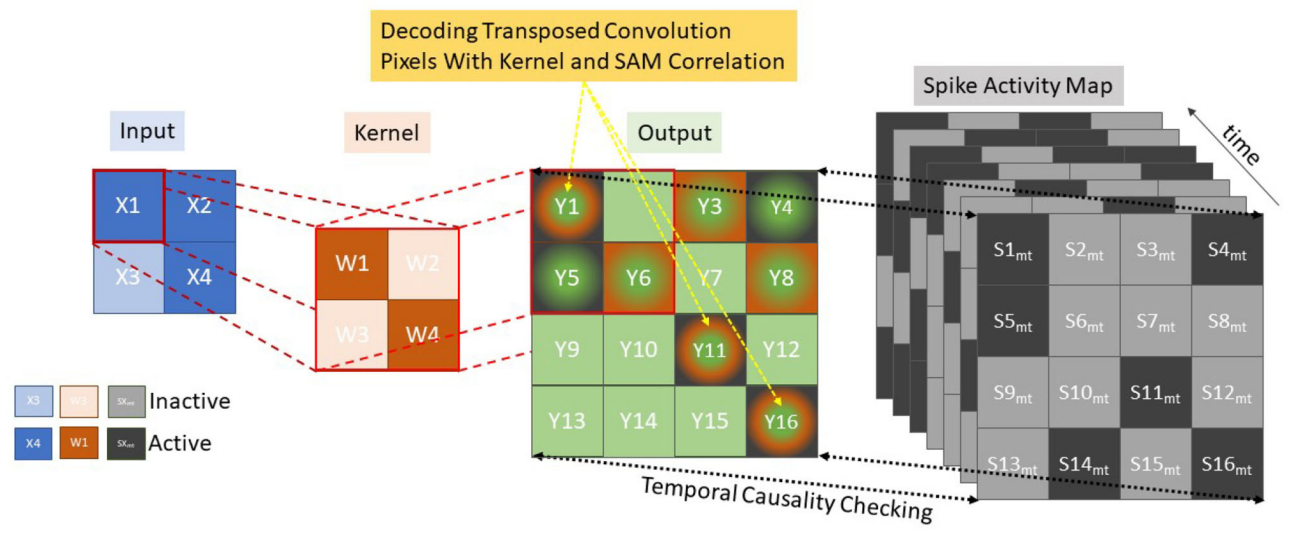


FIGURE 3 | Decoding using transposed convolutions with spike activity mapping, resulting in active pixel saliency mapping.

3.3.3. Decoding

The Decoding Process makes use of the same unpooling and transpose convolutions as (Simonyan et al., 2013; Zeiler and Fergus, 2014; Long et al., 2015; Badrinarayanan et al., 2017) taking pixels in the latent classification space back into the original pixel space. However, no learning mechanism is used,

as the mapping is based on temporal activity and pixel saliency mapping, utilizing a similar method to tied weights (Hinton et al., 2006) and switches (activations within the pooling layers) from the encoding layer to map directly to the decoding such that $W_{ij(encoding)} = W_{ji(decoding)}$. This modification is required to deal with the temporal component of the spiking network, as now the latent pixel space representation must be unraveled with the constraints and context of space and time. Changes are made to both the transposed convolutions and the unpooling layers. The transposed convolution still functions as a fractionally strided convolution of the weight kernel as normal. However, now an extra step of comparing the output mapping with a temporal spike activity map of the post-convolution pixel space is required as illustrated in **Figure 3**, where the conventional Input via Kernel to Output stage remains, with an added Spike Activity Map check on each term in the output for temporal causality.

Since the encoding neurons emit at most one spike per buffered time input, the Spike Activity Map is used to keep track of the first spike times (in time-step scale) of the neurons. Every stimulus is represented by M feature maps, each constitutes a grid of neurons seen as a kernel value K, equal to the row-major linear indexing of the kernel. Let Tp be the processing steps between the tied encoding and decoding layer with a maximum possible difference of nine processing time-steps (five encoding and decoding layers each). While each encoding layer has a value $Te_{m,k}$, which denotes the spike time of the encoding neuron placed at position (k) of the feature map m, where $0 \leq m < M, 0 \leq k < K$. The individual decoding layer then considers this stimulus as a three-dimensional binary spike tensor S of size $Tp_{max} \times M \times K$ where a spike in the decoding layer Sd is a function of:

$$Sd(Tp, Te, m, k) = \begin{cases} 1 & Td_{m,k} = Te_{m,k} + Tp \\ 0 & otherwise \end{cases} \quad (1)$$

Where the decoding time $Td_{m,k}$ for each map and kernel value is compared to the equivalent encoding layer $Te_{m,k}$ offset by the processing time Tp. It is this $Te_{m,k} + Tp$ that is represented by the Spike Activity Map shown in **Figure 3** where $Sk_{m,t}$ is illustrated as the process ensuring $Td_{m,k} = Te_{m,k} + Tp$ while "Output" demonstrates an example of the transposed convolution process. To reduce memory overhead only the last 9 Spike Activity Maps as this is the minimum requirement to ensure temporal causality. Within **Figure 3**, the green and orange squares represent the transposed convolution outputs and the green, orange and black outputs represent the outputs from the transposed convolution decoding that also matched up with encoding layer, through correlation with the Spike Activity Map. This demonstrates how the Spike Activity Map reduces the "Output" values to only those with equivalent temporal values. The saliency mapping occurs within the unpooling layers which operate on a similar manner in order to keep temporal causality, but due to the max pooling operation working in reverse only one pixel per pooling kernel is processed. With reference to **Figure 3**, this would mean the orange kernel would only have one active square, which reduces the output significantly. The measure allows only the most salient features to propagate through the decoding layers, resulting in the segmentation with only those features that best fit the pseudo classification. A verbal illustration being, if there are nine time steps between Conv-1 and TConv-1, while only five steps between Conv-2 and TConv-2 and one step between Conv-3 and TConv-3. So, if a spike occurs at time step 2 within Conv-1, the temporal check will only allow TConv-1 to allow a spike at that location at time step 11.

3.3.4. Adaptive Neuron Thresholding

The adaptive neuron thresholding used within this paper builds upon the Pre-Emptive Neuron Thresholding (Kirkland et al., 2019, 2020). Improvements are made by no longer solely relying on synaptic scaling from the input number of spikes as a means of homoeostasis. Although this was successful in stopping the progression of less structured noise features within the first convolution layer and structured noise when synaptic scaling was applied to all layers. Along with the structured noise filtering process, this homoeostasis rule also accidentally removes some of the less common desired features from propagating as discrimination between these and noise from input spike count is insufficient. The update to the algorithm sees an adaptive element in the form of intrinsic layer-wise synaptic scaling (a layer-wise spike counter) added to the thresholding parameter to potentially counter this less common feature removal. During training the thresholding is set as follows

$$V_{thr}(S_{in}, S_l) = \begin{cases} \frac{K_l}{4} & \text{for } S_{in} < S_{in(min)} \\ \left.\begin{cases} c + mV_{thr} + h^- & \text{for } S_l < H_l \\ c + mV_{thr} + h^0 & \text{for } S_l = H_l \\ c + mV_{thr} + h^+ & \text{for } S_l > H_l \end{cases}\right\} & \text{for } S_{in(min)} < S_{in} < S_{in(max)} \\ \frac{K_l}{2} & \text{for } S_{in} > S_{in(max)} \end{cases} \quad (2)$$

Where V_{thr} is the neuron threshold, dependent on both the spiking input rate, S_{in}, and the layer-wise spike rate, S_l. m is gradient of the linear relationship between V_{thr} and S_{in}, with c being the y-intercept. h the homoeostasis offset is determined to be either positive, negative or zero dependent on the layer-wise spike count, S_l when compared to the set homoeostasis value H_l. While K_l is the convolution kernel size within that layer. The equation follows a piecewise function such that V_{thr} is described as $\{V_{thr} \in \mathbb{N} \mid \frac{K_l}{4} < V_{thr} < \frac{K_l}{2}\}$. When the spike input rate S_{in} is within a normal range, the function is then defined by the bounded linear relationship with the homoeostasis offset. The values of h^-, h^0, h^+ and H_l are set through empirical testing by monitoring the range of S_l and S_{in} values from the N-Caltech dataset.

Once training is complete and the features within the convolution kernels are known, the thresholding changes to take into account the size of the feature, as the range of threshold values might now be smaller than in the training stage. This modification changes the outer bounds of the threshold as shown

$$V_{thr}(S_{in}, S_l) = \begin{cases} \frac{F_{min}}{2} & \text{for } S_{in} < S_{in(min)} \\ \left.\begin{cases} c + mV_{thr} + h^- & \text{for } S_l < H_l \\ c + mV_{thr} + h & \text{for } S_l = H_l \\ c + mV_{thr} + h^+ & \text{for } S_l > H_l \end{cases}\right\} & \text{for } S_{in(min)} < S_{in} < S_{in(max)} \\ F_{min} & \text{for } S_{in} > S_{in(max)} \end{cases} \quad (3)$$

Where F_{min} is the smallest feature size within that layer. This parameter change ensure the threshold value does not exceed the smallest feature size, which would result in that neuron being unable to reach firing potential. In both cases the training and testing the input spike count S_{in} value affects the threshold for each input spike buffer, while the layer-wise spike count S_l is average over 10 inputs.

This allows a layer-wise adaptability dependent on the amount of spiking within the previous layer. The algorithm now permits a high volume of spiking activity at the input to be initially pre-emptively dealt with, ensuring a large amount of spiking activity does not reach the controller, causing an undesired response. Then adapting the thresholds to allow sufficient spiking activity ensures a smoother and more robust controller output of the system. The key element of this method is to ensure a more robust and predictable outcome when a noisy, corrupt or adversarial input is received. With this being more of a concern due to the system be asynchronous end to end, a high volume incoherent input could directly lead to a wild or undesired response from the controller. This approach errs on the side of caution with the sudden increase in input spikes being inhibited first, and then excited to a desired level, in contrast to a typical intrinsic response of allowing the activity, and then inhibiting to a desired response.

3.3.5. Changes to STDP Training With Active Pruning

A simplified unsupervised STDP rule (Bi and Poo, 1998; Kheradpisheh et al., 2018) is used throughout the training process, including a Winner Take All (WTA) approach to STDP, that operates by only allowing one neuron (feature) in a neuronal map (feature map) to fire per time constant; this is viewed as an intra map competition. This WTA approach then moves onto the inter map inhibition, only allowing one spike to occur in any given spatial region, typically the size of the convolution kernel, throughout all the maps. As a result of these inhibition measures, two features can tend toward representing the same feature until such point where one becomes more active, while the other gets inhibited to the point of infrequent or no use. At this stage the feature representation has become obsolete and can be pruned or reset, allowing the opportunity to form another more useful feature. To capture this information the layer-wise training method make use of the training layers convergence values

$$C_l = \sum_k \sum_i \frac{w_{ki}(1 - w_{ki})}{n_{w_{ki}}} \tag{4}$$

Where C_l is the convergence score for the layer and w_{ki} is the ith synaptic weight of the kth convolution kernel. The $n_{w_{ki}}$ is the number of individual weights contained with the layer calculated by kernel size and the number of kernels in the previous and current layers, $n_{w_{ki}} = K \times k_{pre} \times k_{cur}$. The pruning function makes use of the convergence score that is typically used to indicate when training is complete, as the convergence tends to zero due to the weights tending to 0 or 1. Noticing that the layer-wise convergence is just a sum across all the kernels allows a modification to calculate the convergence across each kernel within that layer with respect to all previous maps.

$$C_{k_{cur}} = \sum_{k_{pre}} \sum_i \frac{w_{k_{pre}i}(1 - w_{k_{pre}i})}{n_{w_{k_{pre}i}}} \tag{5}$$

This new terms $C_{k_{cur}}$ allows monitoring of each kernel during the learning process, as previously mentioned obsolete kernels that learned similar features are less active, resulting in higher convergence numbers while maintaining a high spiking activity. The high spiking activity is due to the kernel maintaining the high starting weight value which are random values drawn from a normal distribution with the mean of $\mu = 0.8$ and standard deviation of $\sigma = 0.05$. However, the kernel does not exhibit a feature that allows it to spike quick enough to receive a weight update from the STDP WTA rule. As the kernel had already started a convergence to a particular feature, once under-active it then attempts to convergence to another commonly occurring feature. However, the kernel often convergences to a useless feature representation that is unhelpful to the final result of the network. This pruning method, rather than simply removing the kernel, gives it the chance to learn a new feature from scratch by resetting the kernels weights. Thus, allowing the best chance of convergence to a useful feature. This pruning process takes place once the convergence value of the layer C_l drops below the original starting value. As initially the weights are deconverging from the mean weight initialization, before returning to the original convergence value on the way to zero. Once this milestone has been reached the pruning function in activated

$$Prune_{k_{cur}}(C_{k_{cur}}, C_l, S_k) = \begin{cases} 1 & \text{for} \quad C_{k_{cur}} > \bar{C}_l + 1\sigma_{C_l} \quad \text{and} \quad S_k > \bar{S}_l + 3\sigma_{S_l} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $\bar{C}_l$ is the mean convergence for that layer, σ_{C_l} is the standard deviation of that layers values, S_k is the spike activity within an individual kernel. $\bar{S}_l$ is the mean spike count of that layer and σ_{S_l} is it standard deviation. If a kernel value has a convergence score higher than 1 STD from the mean while having a spiking activity 3 STD higher than the mean spike rate in that layer, the kernel is reset with the initial weight distribution. Since many of the kernels are already converging to useful features this newly reset kernel will convergence to a new unrepresented feature.

3.3.6. Latent Space Inhibition for Attention

In order to have the network change its focus or attention, the latent space pseudo classification layer also acts as an inhibition layer for this mechanism. This operates by inhibiting other neurons in that layer if a specific neurons feature is chosen to be the attention. This is an external mechanism to the network as otherwise, the network will give equal attention to the full scene and semantically segments all known objects within a scene. This allows a simplification of the output of the network fed to the controller, allowing the attention of the system to be narrowed to that particular pseudo-class. This segmentation-based attention can then be used to follow a given class dependent on the output of the controller. It operates between convolution layer 3 and trans-convolution layer 3 with the same principals as the inter map inhibition with the encoder, though now the spatial

region is the whole latent space. This inhibition can also work autonomously where the pseudo-class with the most activity is the attention of the network, allowing the network to switch attention to known classes based on their prevalence within the scene.

3.4. Tracking With Attention

The Action part of the system with its spiking controller is directly influenced by the attention mechanism, as when no attention is chosen the controller acts on all the segmented data being output by the SpikeSEG network. This could cause unwanted control output if the scene contained more than one known class, as unknown classes should still be removed by the process. Once a class has been chosen as the attention, the segmentation output is reduced to only that class, as illustrated in **Figure 1** (Action), which allows for simple spike counter controller to produce a more robust and reliable output. The reduction in information initially by the NVS which then further reduces through the semantic segmentation and attention, allow the implementation of this simple spike counter. This is due to the segmentation output only containing information relating to the attention of the network, the controllers task is just to keep this in the center of the field of view. The simplicity of the controller also allows it to take advantage of the asynchronous event-driven system to provide low latency tracking updates a key element of the system. However, if there was more than one instance of a class in a scene there is no way to separate the two instances, so tracking would be based off all instances of a class. Nevertheless, this system would make an improvement over the purely spiking activity tracking systems by adding some semantic context to the activity, while the simplified spike counter in this instance allows class based tracking could be enhanced with more complex spike tracking, such as dynamic neural fields (Renner et al., 2019)

4. RESULTS

In this section, a series of experiments on individual and multi event-stream recordings are presented. The metric used in this paper is the Intersection over Union (IoU, also known as the Jaccard Index) to grade the segmentation, which guides the control system of the network and ultimately, with user choice, the Attention of the system. This metric was used due to the availability of the bounding box annotations within the subset of the N-Caltech dataset that was used within the experiments. The feasibility of the attention-based tracking is also encapsulated within the IoU value, though due to the small saccade movements of the camera within the N-Caltech dataset, it is infeasible to use this to highlight spike-based tracking. This is due to two issues throughout the movements. The first is the IoU value only receives a small change as the displacement is often <10 pixels. The second is that the occurrence of segmented spike activity in the controlled regions, is due to the tight field of view around the class in scene. This results in the testing of the Perception and Understanding system only with this data. To ensure testing of the full Perception, Understanding, and Action system, two further experiments were carried out. The first with multi input streams on a large input space and the second using our own captured DVS data of a desk ornament with a hand held sensor. Lastly, the results sections show how the system is more robust and interpretable than alternative models, with the use of the Pre-Empt and Adapt Thresholding and the contour like sparse features within the weights of SpikeSEG.

Within these experiments the step time for any processing is now linked to the input time step, meaning internal propagation of spikes takes one step (or 1 ms) per layer, resulting in a 11 ms lag to get the segmented results. This allows for better visualization of the asynchronous manner of the processing and control for each step. However, this does not reflect the actual processing time of the network which, given its complexity compared to similar models ran on neuromorphic hardware, would most likely be able to execute this task in real-time for the 1 ms step, meaning a full pass through the network per input step. However, testing in this manner would not fully highlight the asynchronous advantage especially within a dynamic environment.

One further note is that throughout all the testing the features of Convolution Layer 1 are pre-set to best found features for initial edge detection, which results in a horizontal, vertical and two diagonal lines which can be see later in the Interpretablity section 4.3.2 within **Figure 14**.

4.1. Perception to Understanding With Segmentation

Initially two subset classes from the N-Caltech dataset (Orchard et al., 2015) are used to evaluate the Understanding section of the system. On this single stream input typically only containing a singular class with variable amounts of background noise and clutter, the network is able to gain an accuracy of 96.8% within the pseudo classification layer and a 81% mean Intersection over Union score over each of the 10 ms buffered input that resulted in successful segmentation, results are also shown in **Table 1**. This is an improvement on the single results seen within (Kirkland et al., 2020) of 92 and 67% for accuracy and IoU, with the improved feature creation allowing a more detailed representation allowing an improvement in both the accuracy and segmentation. The test results are based on training with 200 samples from the Face and Motorbike classes with another 200 used for testing. This number was limited as the "Easy Faces" has just over 400 images and was converted into "Faces" within the N-Caltech dataset with the "Faces" category being removed. Four hundred images provided an equal training set between the Face and Motorbike classes. The images in **Figure 4** shows how the segmentation process was completed firstly through encoding the event stream input through three convolution and two pooling layers (**Figures 4B–D,I–K**), resulting in a sparse latent space representation used to provide a classification of this binary task (**Figures 4D,K**). **Figure 4**, then shows how the classification locations are then mapped back onto the pixel space through the undoing of the three convolutions and two pooling layers (**Figures 4E–G,L–N**). For illustrative purposes, both the face and motorbike are accumulations of the network activity according to 10 ms input buffer and full propagation of spikes through the network. Each convolution process is shown, with

TABLE 1 | Results from each of the experimental setup, listing both the accuracy and intersection over union.

Dataset	Classification accuracy (%)	Intersection over union (%)
N-CalTech (2 class)	96.8	81
N-CalTech (5 class)	86	76
N-CalTech (10 class)	75	71
Multistream N-CalTech	96.8	81
Multistream N-CalTech with noise	95.1	79
Panda	94	75

pooling omitted, Convolution Layer 1 is shown in **Figures 4B,I** while layer 2 (**Figures 4C,J,D,K**) showing the third convolution also used as pseudo classification. **Figures 4E,L** show the second transposed convolutional layer, named to mirror the encoding side, while **Figures 4F,M** show transposed convolution 1 and **Figures 4G,N** display the segmented outputs. This segmentation result is shown overlapped onto the input for two examples within **Figure 4**. The colors used within **Figure 5** are linked to the corresponding feature that was activated in that layer with **Figures 4C,J** showing different colored features active for each the face and motorbike, with section 4.3 exploring what the feature maps contain. This output from the SpikeSEG network can feed directly into the spiking controller of the PUA system, guiding any movement that would be required to follow the attention of the system. Although the controller in this context is unable to operate due to the narrow field of view and limited movement, the Understanding section of the system does still capture this small saccade movement of the camera within the segmented output as seen in this overlapped output image, **Figure 6A** showing a downward and right shift of the segmented pixels over time, relating to the inverse movement performed by the camera, while **Figures 6B,C** show the two further saccade movements. The segmentation also maintains an IoU value of above 0.7 throughout the movement, meaning the segmentation is of good quality throughout (0.5 being acceptable, 0.7 being good, and 0.9 being precise) (Zitnick and Dollár, 2014), for reference if the full input size is used for IoU the average output is ∼0.57. Consequently, this means tracking would still be possible through alternative non-spiking methods such bounding boxes or centroid/center of mass calculation, but would remove the all spiking asynchronous feature of this system.

4.1.1. N-Caltech Dataset Extended

To further evaluate the scalability of the model, a further two experiments are carried out with 5 and 10 classes. This allowed testing the model with 2, 5, and 10 classes within the same experimental parameters, that being 16 features per class in second convolution layer and 1 per class in the third convolution layer, with active thresholding and pruning. Sixteen features was found to be a suitable value for number of features through prior empirical testing, where more features gave no further improvement, while less features was unable to capture the variation of some classes. The further classes added are: Inline Skate, Watch and Stop Sign for the 5 class, while Camera, Windsor Chair, Revolver, Stegosaurus and Cup are added for the 10 class experiment. These classes are chosen due to low variability in image spatial structure. As the network is only looking for natural spatial structural similarity avoidance of classes which have a large intraclass variance compared to the overall interclass relationship (Zamani and Jamzad, 2017). With this in mind and due to some the additional classes having a smaller number of sequences, the number of training and testing instances was changed to suit, at 20 training and 10 testing. Overall the network was able to achieve classification accuracies of 86 and 75% and IoU values of 76 and 71% for the 5 and 10 classes, respectively, results are shown in **Table 1**. The decrease in overall accuracy with additional classes is to be expected at the features built in the second convolution layer tend to get more similar. This is visually detailed in section 4.3.2 with the Interpretability showing the different features learned in the convolution layers. With this closer similarity of layer-wise features, an example of how the active pruning mechanism is shown in **Figure 7**, where a number of the features within the second convolution layer have a slower convergence rate while maintaining a high spike activity. This typically suggests the feature is not very discriminative and is an ideal candidate for being reset to learn a new feature. **Figure 7**, shows the original features just prior to reaching the pruning check point within (A), then indicates which features are chosen to prune with the feature being reset to random initialization within (B), the finally resulting in new features shown in (C).

Drawing insight from the result, within the 5 class experiment the inter class variance was high. However, once the 10 classes were added this inter and intra class variances seems to overlap. Resulting in many of the classes relying on similar features constructed from circles, with Motorbike, Cup, Camera, Watch, Stop Sign, and Face at times producing features are that undistinguishable from one another. It was also noted that as the number of classes increased the difference between average number of features in a kernel per class (that is ones that can be recognized as belonging to a particular class) leads to a higher likelihood that the class with the highest average feature number will be the most active. Within the last experiment with the 10 classes this was prevalent within the Revolver class as it had an average feature count in convolution layer 2 of around 200, while the average for camera was 110. This results in a higher chance that the revolver was classified by mistake ultimately bringing the overall accuracy down.

4.2. Perception, Understanding, and Action

This section is split into two parts both further testing the full PUA system, the first continues using the N-Caltech Dataset, however with multiple simultaneous inputs. The second part makes use of recorded data of desk ornament from a hand-held NVS to provide a further example of how the system works within another test environment and how the action part of the system deals with a moving class.

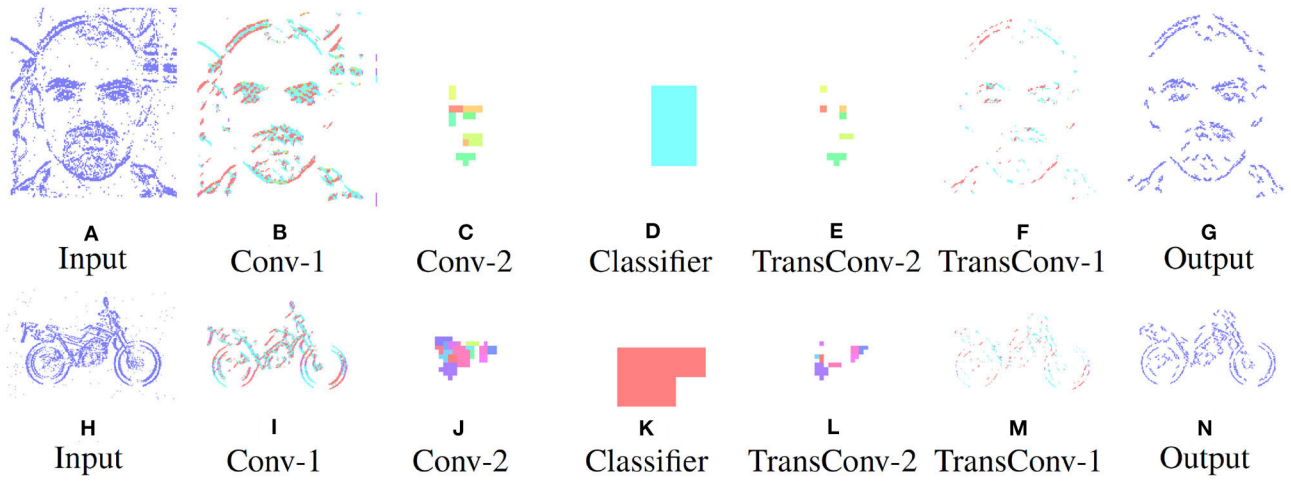


FIGURE 4 | Segmentation performance of the network on an example face **(A–G)** and motorbike **(H–N)**, highlighting the encoding transition into the latent space used for pseudo classification **(A–D,H–K)**, then retracing of chosen features back to pixel level **(D–G,K–N)**.

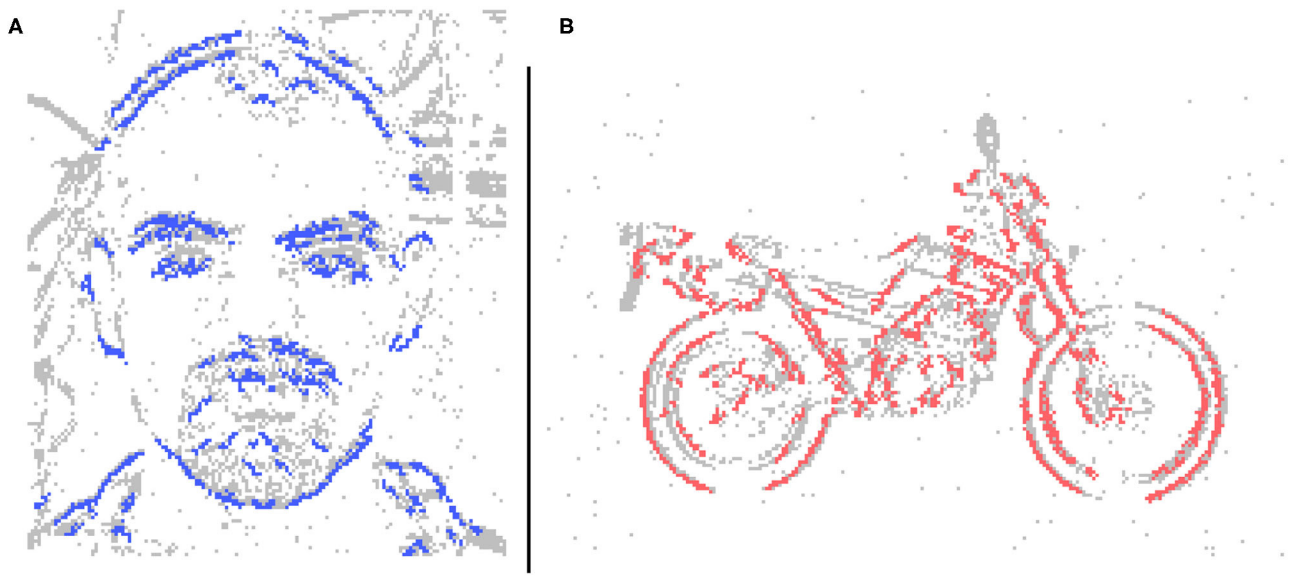


FIGURE 5 | Segmentation overlays for the **(A)** Face and **(B)** Motorbike class from the N-CalTech dataset.

4.2.1. N-Caltech Mutli-Stream Input

Building upon the results gathered from the successful process in section 4.1, this experiment looks at how the system would deal with multiple input streams. This allows the network to demonstrate the segmentation ability in the face of multiple distractors and spatio-temporal Gaussian noise with an average PSNR of 18 dB, an example of the input with and without noise is shown in **Figures 8A,B**, respectively. **Figure 8** also demonstrates the layout of the new input image, which is based on the Face class subset, but is three times the size to make a 3 × 3 grid where each corner and the center will host an input stream. Each stream is presented for 300 ms (dictated by the recording length in the dataset) then some of the locations are changed and the next stream is played. The input streams illustrated in **Figures 8A,B**, consist of one face and two motorbikes for the known classes and two Garfield streams for the unknown, with **Figure 8B** demonstrating the affect of noise on the input. This gives an opportunity to display the asynchronous layer-wise spike propagation once thresholds have been surpassed, while also offering an insight into how an SNN reduces computational throughout with this thresholding value.

Figure 9 displays both this asynchronous throughput of activity and how the network reduces the numbers of computations, even when presented with noise and distractors, with the time axis showing an accumulation of spikes to ease with visibility. **Figure 9** shows that by Conv 1 the added noise is mostly

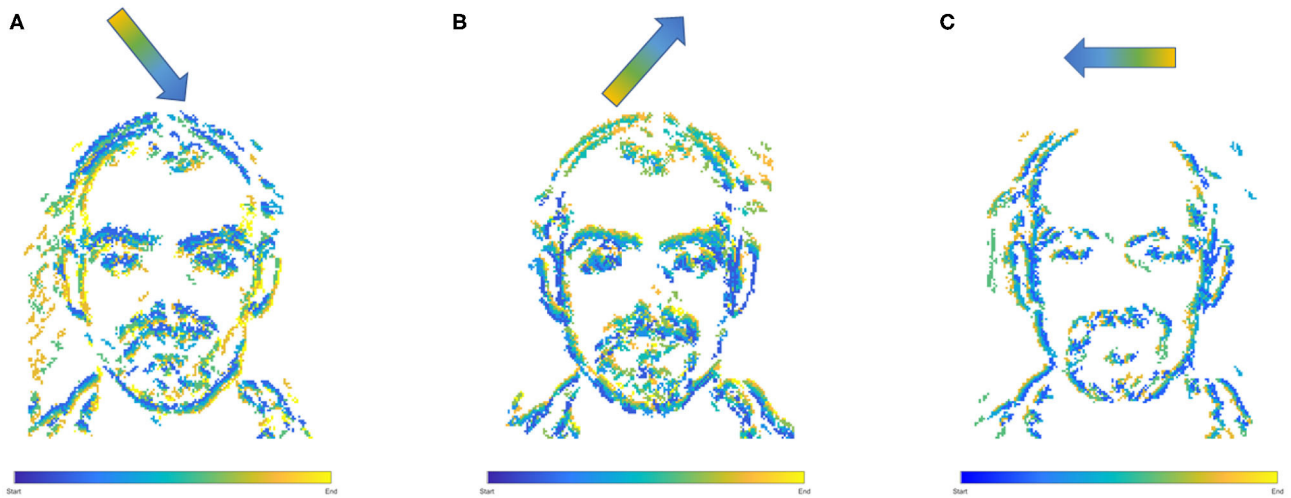


FIGURE 6 | Overlapped Segmentation output over the complete event stream, showing the triangle of movements over the three saccades, **(A)** first movement, **(B)** second movement, **(C)** third movement.

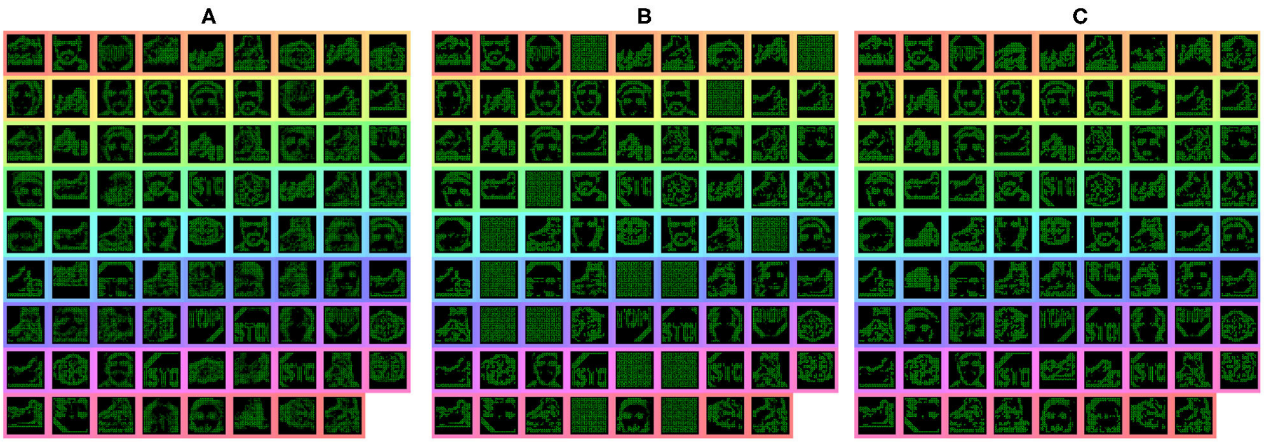


FIGURE 7 | Features from the second convolution layer during training highlighting the pruning process. **(A)** Highlights the features prior to pruning, **(B)** shows which feature were reset to initial parameters, and **(C)** shows the newly learned features.

removed as it lacks any real structured shape, but the distractor, Garfield, remains and progresses onto Conv 2. During this layer though, due to its low saliency with any of the learned features for the classes of Face or Motorbike the distractor is removed from the processing pipeline. This leaves only the two known classes, which then progress onto the Conv3 layer, then through the decoder layers to the output where they are successful segmented. When testing the multi-stream input without any noise the accuracy and IoU value is identical to the single stream instance at 96.8 and 81%. Then with added noise this value sees a slight reduction to 95.1 and 79% for accuracy and IoU, these results are also shown in **Table 1**. The decreases being attributed to the noisy pixels directly contacting or occurring within the class boundary, as the network has no real way to discern this noise from actual data. This is clearly shown within the segmented output comparisons shown in **Figure 10**, where the noiseless output (A) and the noisy (B) show considerable difference in their respective segmentations with far more diagonal lines present in the noisy output (B) in comparison to (A). This outcome could have been predicted and will be highlighted in section 4.3 as the first layer of the network has a larger feature representation for the diagonal line when compared to the horizontal and vertical lines, with more pixels allocated to representing the diagonal lines rather than horizontal and vertical, due to the larger variety of edges this feature had to represent. Meaning relatively with the same threshold the diagonal feature is more likely to be activated than the horizontal and vertical.

With the segmentation successfully output the spiking controller now has less spiking activity so should find it easier to be able to track a given class. The tracking starts once the user has made a selection of which class is to become the attention of the network. Experimentally this was tested by selecting the

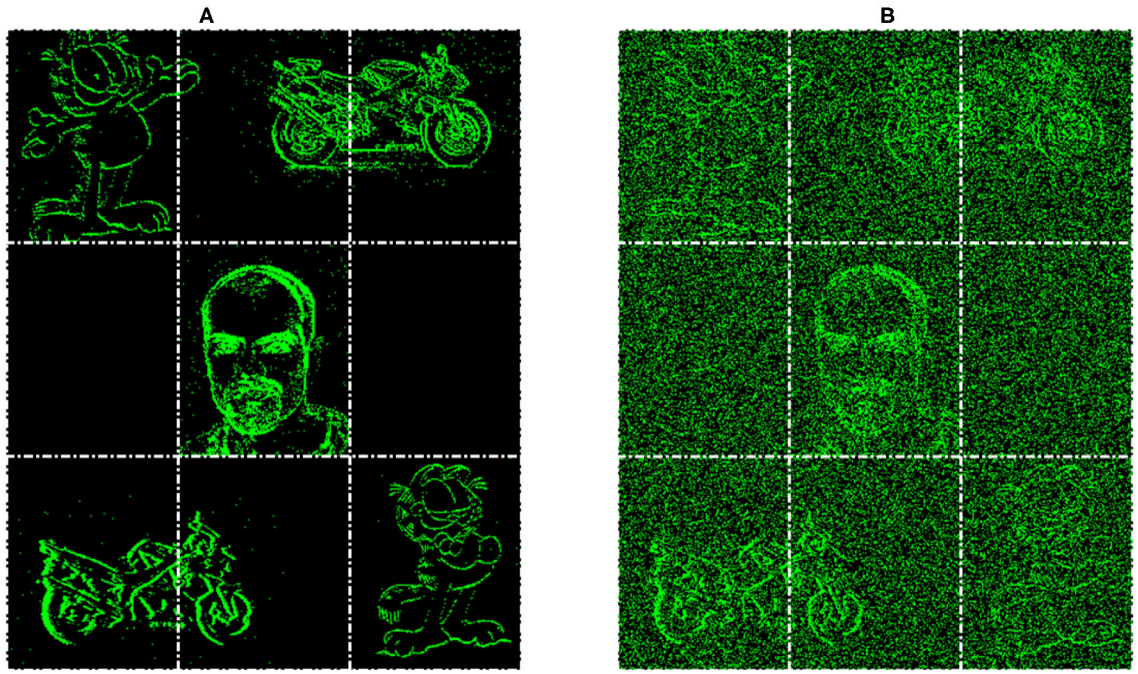


FIGURE 8 | Example of input for the Multi-Stream Input without noise **(A)** and with noise **(B)**, both with extra gridlines indicating the 3 × 3 grid which determines the initial location of the inputs.

attention after two successful multi class segmentation examples where the stream inputs were repositioned. **Figure 11** displays the outputs of the three inputs (A–C) with their subsequent paths to segmentation. **Figure 11** shows that for inputs (A,B) the network is correctly segmenting the input and displaying an output with a highlighted segment displayed in the 3 × 3 grid. It is only in **Figure 11C** that the guided attention mechanism is triggered causing the inhibition of the other class in the propagation between layer Conv 3 and T-Conv 3. This feature is highlighted with the red circle showing which neurons are now no longer represented in the subsequent layer and thus no longer computed out to the segmentation, highlighting part of the efficiency in SNN. The last section of the diagram in **Figure 11** highlights the attention of the network being drawn to the face located on the bottom left of the grid, which in the spiking controller would result in an output of left and down to ensure the face is located within the central region. The arrow within the **Figure 11C** also indicated the movement of the track update, which is based off the central region as within the previous two sequences the multiclass attention doesn't give a control output. This attention-based tracking update is delivered within 34 ms or 34 input steps, which with the 11 ms processing lag with each layer to propagate through the network results in a 31 ms delay within the 300 ms input stream. This may seem like a considerable amount of time, but as shown in both **Figures 2**, **9** due to the way the N-Caltech dataset was recorded, the first 30 ms of the recording contains very little information due to the lack of movement with the main concentration of spiking activity during the middle of each of the saccade movements. To test this the first 30 ms of events were removed from all the input streams which result in a reduction in track update to 15 ms and with the offset of 11 ms to progress through the network means a 4–5 ms latency to get from input to a control output if the processing could be done in real-time. However, even this latency is mainly from the initial delay in spiking activity within the network first layer, suggesting once running the latency would decrease. This would make it a highly competitive alternative or efficient middle ground between highly precise CNNs and low latency edge detection systems. Furthermore, the total number of average calculations represented by the images seen in **Figure 11** is only ∼9% of the total available calculations (equivalent CNN) due to the sparse nature of both the features and the SNN thresholding processing. Approximately 10% of capacity is used in the encoding process and ∼5% in the decoding process, which is visualized in both **Figures 9**, **11**.

4.2.2. Tracking From Handheld NVS

For this section, the SpikeSEG network was retained to be able to identify a panda desk ornament and aims to better highlight the control and tracking aspects of the PUA system. The input stream recorded from DVS346 NVS has the panda start on the far left in the field of view then the camera pans to the left resulting in the panda being on the far right, with an example of the input images shown in **Figure 12A**. The results detail how well the segmentation would work within this example, with the extra complexity of 3D shapes and natural indoor lighting conditions. Overall the results of the 1 s test stream, show that only 60 ms (6%) of streaming footage failed

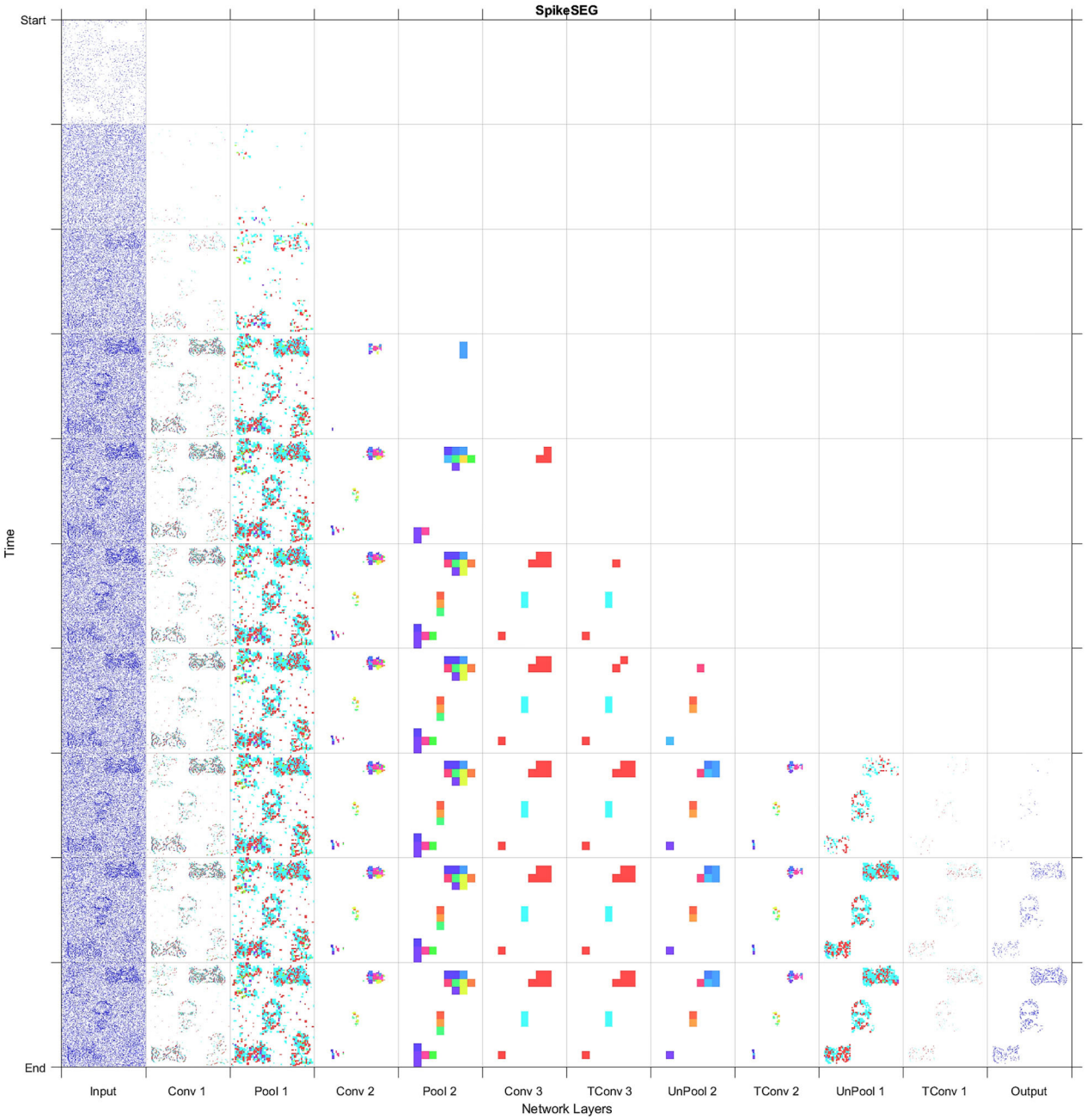


FIGURE 9 | Full Layer-wise spiking activity for the system, showing the progression of spikes through the network encoding then decoding section into the segmentation output.

to produce a segmentation output. This also occurs at the points where the least amount of movement of the camera happens, the turning points, subsequently producing fewer output spikes. Nevertheless, this results in no actual loss in tracking accuracy as the panda object stayed within the previous segmentations IoU bounding box. Furthermore, the IoU for this test stream was 75%, shown in **Table 1**, perhaps lower than expected given the high level of accuracy within the classification/segmentation process. This is illustrated in **Figure 12A** where the middle section of the panda is not well-resolved by the sensor, meaning on occasion the segmentation output was only of the top or bottom section. **Figures 12B–D** also show the full system process for the two different control outputs of moving left (D) and right (C), that is when the segmentation area enters the proximity of the spike counter at the edges of the output image. Within **Figure 12D** there is also an example of how the system

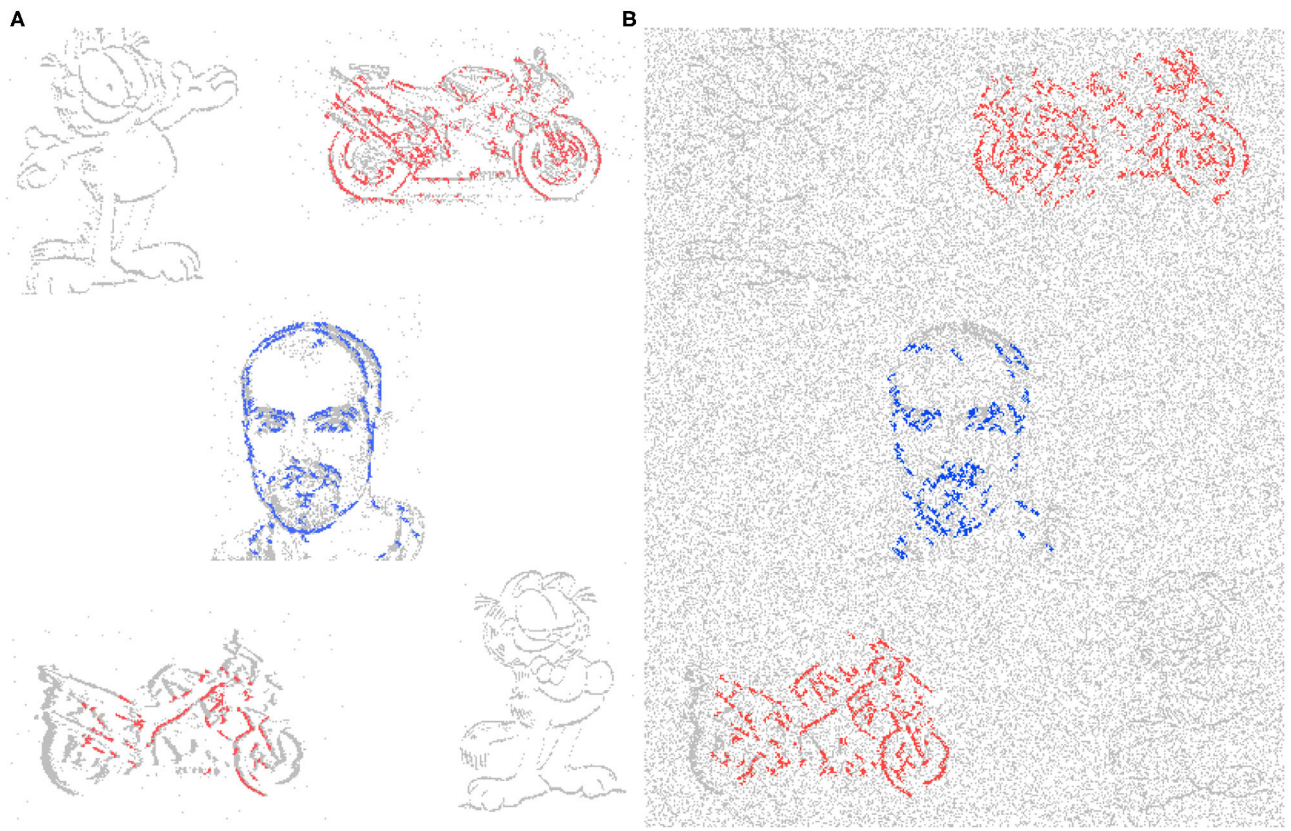


FIGURE 10 | Segmentation overlays for the **(A)** Multi-Stream Input and **(B)** Multi-Stream Input with noise, including the classes Face, Motorbike, and Garfield from the N-CalTech dataset.

overcame a background object that could have affected simpler approaches, as originally the input image had a background object on the right hand side of the image. Due to the feature extraction and segmentation, the background object was unable to influence the controller which without the Understanding-based segmentation would have had spiking activity in both left and right spike counters.

4.3. Robustness and Interpretability

This section highlights two key features of utilizing an SNN approach for this framework, the first is system robustness, especially that pertaining to Perception and Understanding (the sensor and processing) and how that affects the Actions of the system. The second feature is that of interpretability something that is not often not associated with CNN type approaches.

4.3.1. Robustness

The added robustness of the PUA approach comes from the Understanding section within the PEAT (Pre-Empt then Adapt Thresholding) mechanism. As mentioned in section 3.3.2, the buffering of input spikes allows a spike counter to be implemented, allowing a pre-emptive rather than reactive approach to the thresholding within the network. Permitting synaptic scaling homoeostasis to increase the threshold values on all layers, ensuring noisy or adversarial inputs are mitigated first. Subsequently, if the spike level persists the threshold levels using an intrinsic homoeostasis may be adapted. An example of this system at work is illustrated within **Figure 13**, with (A) showing a multi-stream input with no noise, then the input is corrupt with noise in (B–D) showing the resulting effects of the noise throughout the system with and without the PEAT mechanism active. The PUA framework implements the regime that no output is better than an incorrect output, especially when the input is degraded due to noise or adversarial sensor values. This robustness features is highlighted in the output of **Figure 13B** which is incorrect and if passed to the controller could cause an undesired response, meanwhile in **Figure 13C** the PEAT is seen to allow the network to threshold the noise level resulting in no segmentation output. Incidentally, **Figure 13D** could be the adaptive outcomes of both approaches (B,C), it is just intermediate control output suppression that adds an extra level of robustness to the system.

4.3.2. Interpretability

The interpretability of a system is often overlooked when values of accuracy or precision appear to be high. But understanding or gaining some insight into how the system got to an answer could be a valuable advantage for SCNN compared to conventional

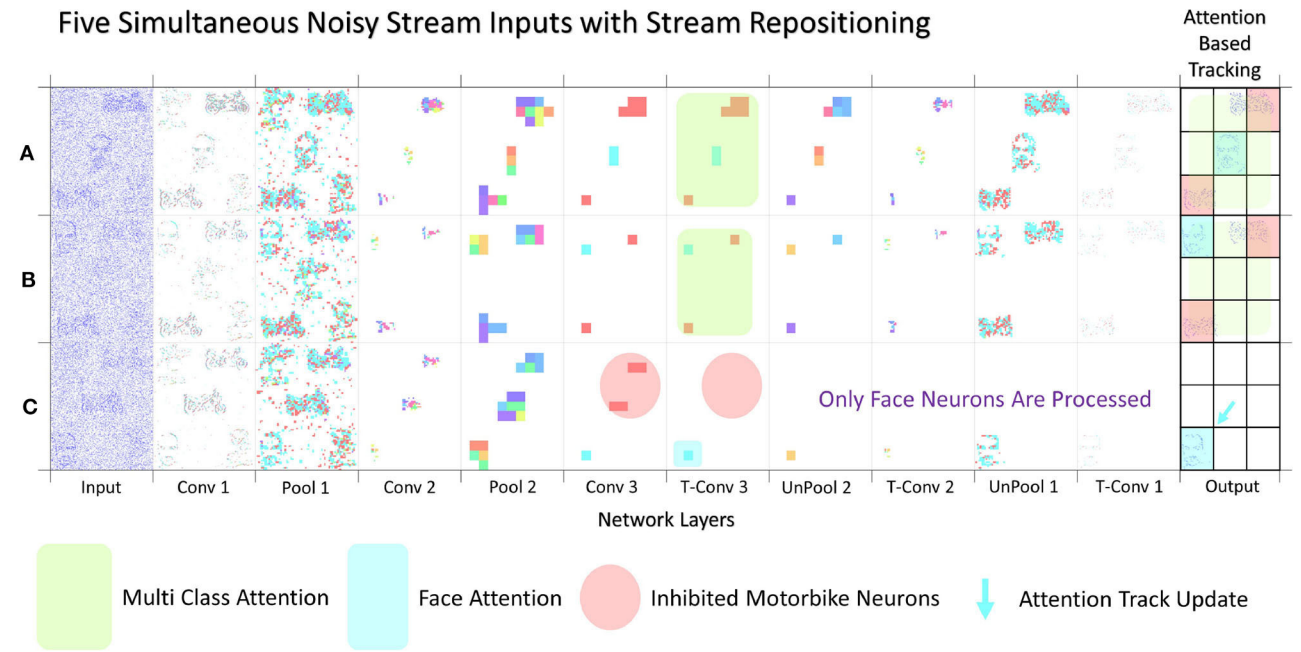


FIGURE 11 | Image showing three separate multi input data streams. **(A,B)** both representing the full system layer-wise computations when no attention is selected, while **(C)** shows the layer-wise computation after the Face class has been selected as the attention of the network, thus enabling a simplification of the output and activating the action part of the system with a tracking controller update.

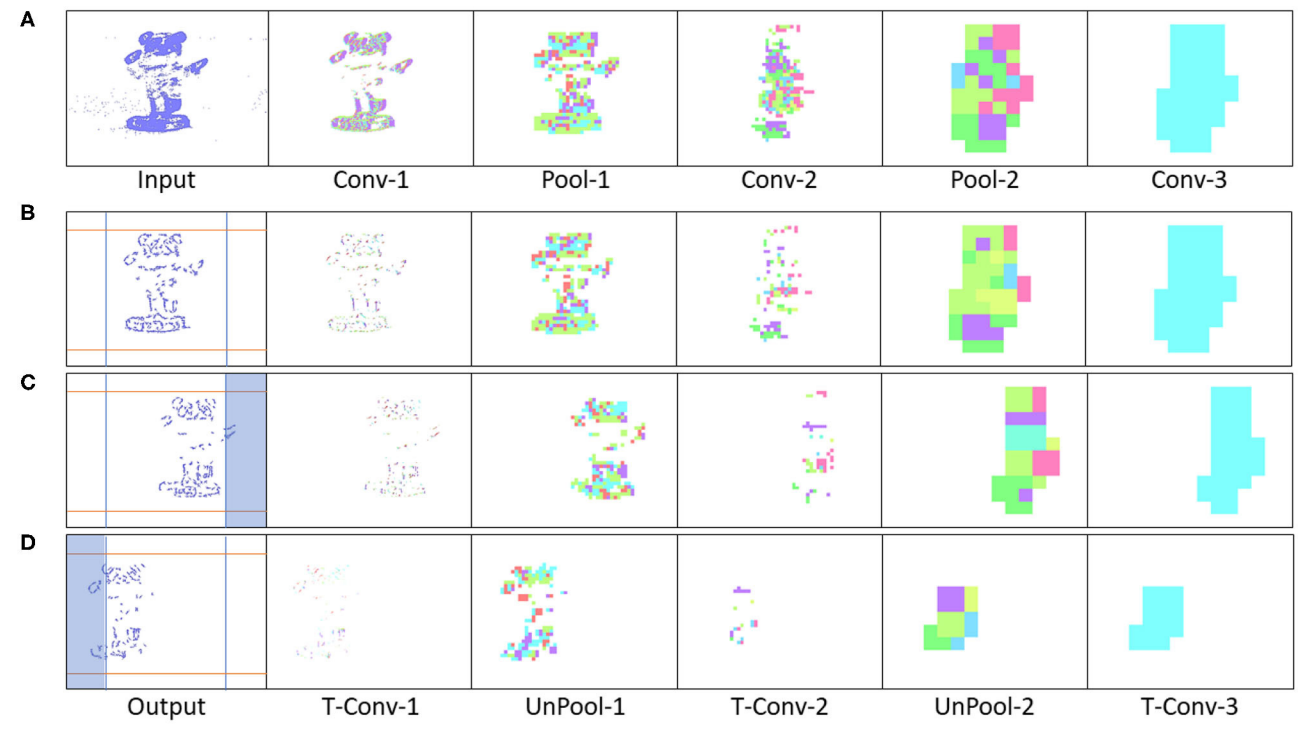


FIGURE 12 | (A) Panda Input Image, **(B)** Panda being in the middle in the field of view, **(C)** Panda reaching rightmost boundary triggering a control action, **(D)** Panda reaching leftmost boundary triggering a control action.

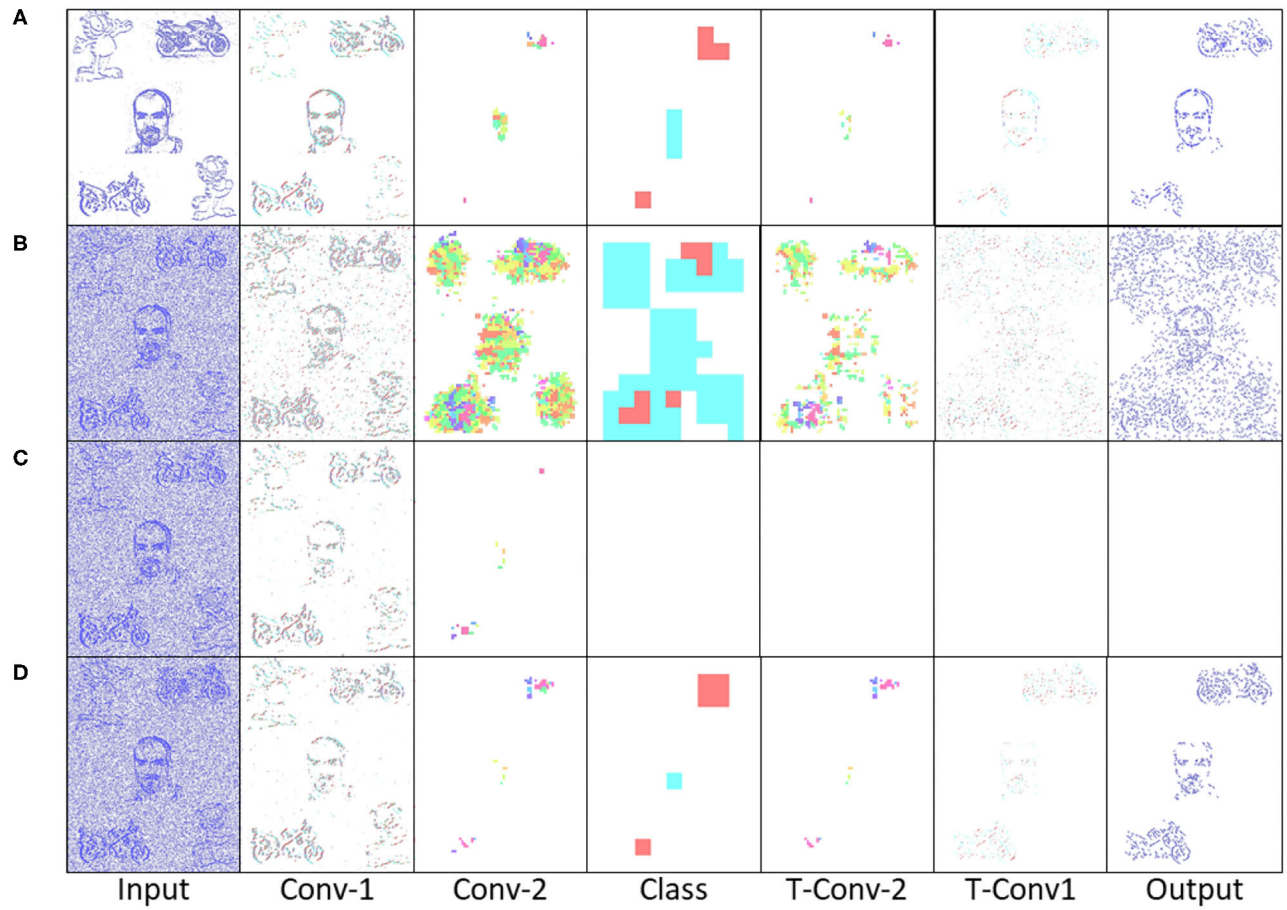


FIGURE 13 | Highlighting the Robust noise suppression with the Pre-Empt then Adapt Thresholding mechanism. **(A)** Showing no noise input as reference, **(B)** Showing noisy input without PEAT active, resulting in noisey output, **(C)** Pre-Emptive Thresholding causing over suppression of neurons resulting in no output, **(D)** Shows the input to output with PEAT active, suppressing the noise but allowing the useful data to pass through.

CNNs. As SCNN trained using STDP happen to produce a sparse feature variation of typical CNN outputs, the SCNN results in features that are more akin of those from contour matching papers (Barranco et al., 2014) while CNNs typically take on features that resemble textures (Olah et al., 2017). These texture maps are often hard to interpret, although modern approaches have found ways to highlight the most salient parts of an input with reference to these texture maps. Nevertheless, it is still often difficult to predict how the system might react to an unknown input. The features that were learned for the testing of the N-Caltech dataset used within this work is shown in **Figure 14**. **Figure 14A**, illustrates the differences between the previous version of the model and the current implementation with PEAT and pruning improving the feature extraction, using the same Conv-1 features representing simple edge detection structure of horizontal, vertical and two diagonal lines. **Figure 14A** then shows the mapping those features onto the weights of the Conv-2 resulting in the features that resemble shapes and objects before the classification stage in Conv-3. It can be seen that half of the 36 features in Conv-2 relate to the Face class and the other half the Motorbike, with these features helping to build up the classification layers with two features either Face or Motorbike. This image helps to explain what the network has learned and how it appears to be looking for contour like shapes to help it distinguish between inputs. Along with this insight into how the network operates, it also allows the user to perhaps understand why it might not always give the correct answers. Similar to how if creating a system using hand-crafted contours features, you would understand the limitation this allows a similar understanding to be had. This could allow manual manipulation of features or manual pruning throughout the training if the user happens to have expert knowledge of the task, bringing neural networks closer to known computer vision-based techniques, which could provide an interesting overlap, especially in the robotics domain.

In order to perceive how the additional classes affects the interpretability of the system **Figures 14B,C** highlights a sub-selection of the features within the 5 and 10 class models. This highlights how the interpretability is still there for some of the

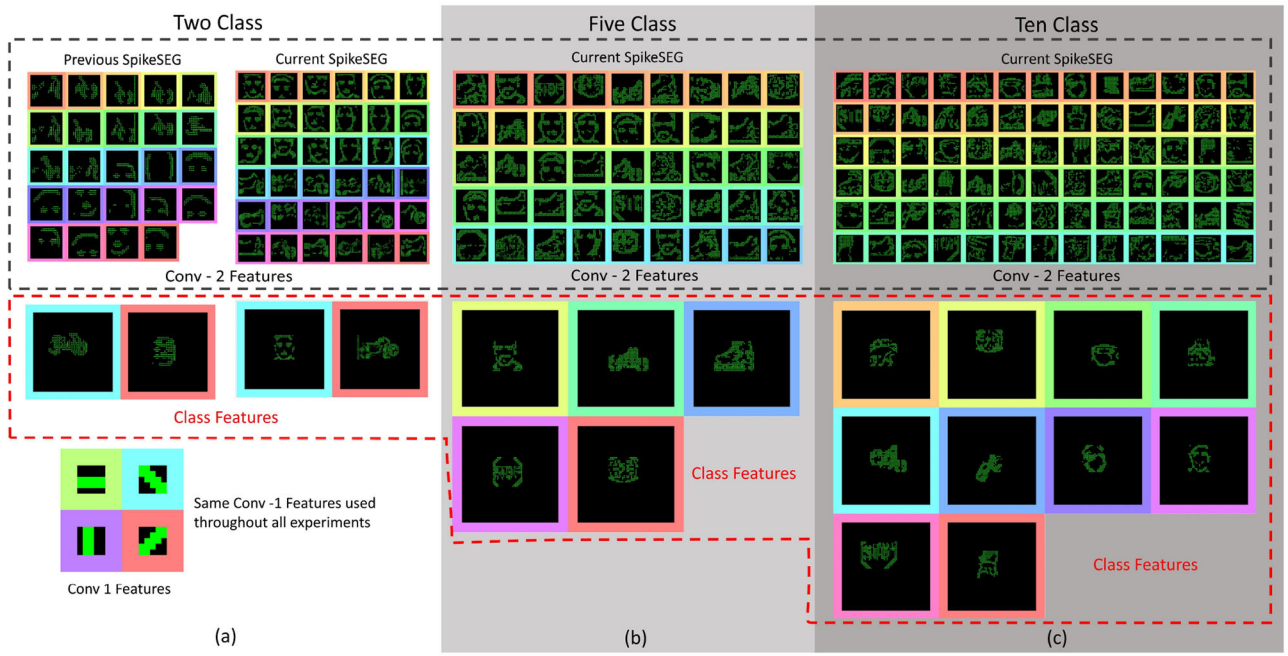


FIGURE 14 | (a) Features map representations of the convolution layers, with coloring to match the latent space representation from the two class experiment, showing prior and current results of the Conv-2 features and Class features. The Figure also shows a selection of features from both the Five Class **(b)** and Ten Class **(c)** experiments. Top half showing the Conv-2 features and the bottom showing the Class Features. **(a)** Classes shown in Class Features are Motorbike-Face then Face-Motorbike for the previous and current results. **(b)** Classes shown in Class Features order are: Face, Motorbike, In-line Skate, Stop Sign, and Watch. **(c)** Classes shown in Class Features order are: Stegosaurus, Watch, Cup, In-line Skate, Motorbike, Revolver, Camera, Face, Stop Sign, and Windsor Chair.

features while others have become more difficult to understand, perhaps due to overlapping features from two classes. Overall, **Figures 14B,C** highlight how reviewing of the features within a Spiking Neural Network can help to gain understanding about parts of the network, with the classification layers features representing each of the 5 and 10 classes. The visualizations help to explain why certain classes might struggle vs. others due to similar sub classification features.

5. DISCUSSION

The understanding method shown in this work details an unsupervised STDP approach. To fully utilize the spiking nature of the processing it is paired with the perception method of spiking input sensor. Together this perception understanding pair can successfully semantically segment up to 10 classes of the N-Caltech dataset. The output of this process is a spiking grid indicating the location of the class within the scene, which can be interpreted by the action system to allow the objects to be followed if attempting to leave the field of view.

The full PUA process is completed in a spiking and fully convolutional manner. This ensures all calculations are either spiking or spike counting. Allowing the network to maintain the temporal and processing advantages, along with the asynchronicity associated with neuromorphic vision sensors, from input to output. However, this method of processing is not without its drawbacks, as there is an overall decrease in accuracy associated with this adding of extra classes. That perhaps indicates the limitation with this unsupervised method in terms of problem scaling. For instances with the 100 classes available within the N-Caltech dataset, the system would only be able to learn the most common features that occur within each class, but only if they present a large enough variance. That is it will only learn common class features as long as they look different enough from the other classes. Which is essentially what can be seen happening with the 5 and 10 class experiments visualized in **Figures 14**, **7C**. **Figure 7C** highlights that even with a high inter class variance the kernels sometimes learn differentiating features from all other classes, while other times learns features that are an amalgamation of two or more classes. The 5 class experiment displays this most prominently with the Bike and In-Line Skate classes, as there are similarities between the outline shape of both objects.

Nevertheless, this ability to find most common features that express the highest variance from others, is both the limitation and strength of this STDP approach. Limiting in that this approach might not scale to larger datasets, but a strength in that it made the network asynchronous, adaptable, computational sparse and visually interpretable. This highlights that the STDP method used might not be suitable for all problems, but serves as a indication of the benefits if the problem is appropriate. This work demonstrates that STDP

alone can be used to find the most common features of a dataset. Which in turn, can be used to successfully perform image classification and semantic segmentation. However, a further learning rule to help focus on more discriminative features, such as R-STDP (Izhikevich, 2007; Legenstein et al., 2008; Mozafari et al., 2018) would be a useful extension. This could help in tackling the main issue of inter to intra class variance differentiation. This could allow not only the most common feature to be discovered, but the most common discriminative feature.

6. CONCLUSION

We proposed a new spiking-based system, the Perception Understanding Action Framework, which aims to exploit the low latency and sparse characteristic of the NVS in a fully neuromorphic asynchronous event driven pipeline. Using the understanding gained through the SpikeSEG segmentation, the network is able to detect, classify and segment classes with high accuracy and precision. Then from this understanding, the system makes a more informed decision about what action is to be taken. In this context, the framework was able to show a semantic class tracking ability that combines feature extraction capability of CNNs and low latency and computation throughput of line and corner detection methods. The framework also explores the unique benefits that can be gained through utilizing SNNs with interpretability and robustness, with the use of thresholding algorithms and sparse feature extractions. The PUA framework also shows off the unique attention mechanism, emphasizing how simple local inhibition rules when combined with an encoder decoder structure; this can help reduce the computation overhead of the semantic segmentation process. This research highlighted the series of benefits when utilizing a fully neuromorphic approach with a pragmatic engineering and robotics outlook, looking at the biologically inspired mechanisms, features and benefits, then combining them with modern deep learning-based structures.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

PK contributed to the framework and experimental work. PK, GD, and JS contributed to the paper writing. All authors contributed to the article and approved the submitted version.

FUNDING

This work was support in part by Leonardo MW Ltd., as part of a Ph.D. sponsorship programme.

ACKNOWLEDGMENTS

We would like to thank GM for discussion, reviewing, and his helpful feedback.

REFERENCES

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). SegNet: a deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 2481–2495. doi: 10.1109/TPAMI.2016.2644615

Barranco, F., Fermuller, C., and Aloimonos, Y. (2014). Contour motion estimation for asynchronous event-driven cameras. *Proc. IEEE* 102, 1537–1556. doi: 10.1109/JPROC.2014.2347207

Bi, G., and Poo, M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *J. Neurosci.* 18, 10464–10472. doi: 10.1523/JNEUROSCI.18-24-10464.1998

Bichler, O., Querlioz, D., Thorpe, S. J., Bourgoin, J.-P., and Gamrat, C. (2012). Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Netw.* 32, 339–348. doi: 10.1016/j.neunet.2012.02.022

Bohg, J., Hausman, K., Sankaran, B., Brock, O., Kragic, D., Schaal, S., et al. (2017). Interactive perception: leveraging action in perception and perception in action. *IEEE Trans. Robot.* 33, 1273–1291. doi: 10.1109/TRO.2017.2721939

Brandli, C., Berner, R., Minhao, Yang, Shih-Chii, Liu, and Delbruck, T. (2014). A 240 × 180 130 dB 3 \mui s latency global shutter spatiotemporal vision sensor. *IEEE J. Solid State Circuits* 49, 2333–2341. doi: 10.1109/JSSC.2014.2342715

Cao, Y., Chen, Y., and Khosla, D. (2015). Spiking deep convolutional neural networks for energy-efficient object recognition. *Int. J. Comput. Vis.* 113, 54–66. doi: 10.1007/s11263-014-0788-3

Clady, X., Ieng, S.-H., and Benosman, R. (2015). Asynchronous event-based corner detection and matching. *Neural Netw.* 66, 91–106. doi: 10.1016/j.neunet.2015.02.013

Conradt, J., Cook, M., Berner, R., Lichtsteiner, P., Douglas, R., and Delbruck, T. (2009). "A pencil balancing robot using a pair of AER dynamic vision sensors," in *2009 IEEE International Symposium on Circuits and Systems* (Taipei: IEEE), 781–784. doi: 10.1109/ISCAS.2009.5117867

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Delbruck, T., and Lang, M. (2013). Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor. *Front. Neurosci.* 7:223. doi: 10.3389/fnins.2013.00223

Delbruck, T., and Lichtsteiner, P. (2007). "Fast sensory motor control based on event-based hybrid neuromorphic-procedural system," in *2007 IEEE International Symposium on Circuits and Systems* (New Orleans, LA: IEEE), 845–848. doi: 10.1109/ISCAS.2007.378038

DeWolf, T., Stewart, T. C., Slotine, J.-J., and Eliasmith, C. (2016). A spiking neural model of adaptive arm control. *Proc. Biol. Sci.* 283:1843. doi: 10.1098/rspb.2016.2134

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science* 338, 1202–1205. doi: 10.1126/science.1225266

Falez, P., Tirilly, P., Marius Bilasco, I., Devienne, P., and Boulet, P. (2019). "Multi-layered spiking neural network with target timestamp threshold adaptation and STDP," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest: IEEE), 1–8. doi: 10.1109/IJCNN.2019.8852346

Gehrig, D., Rebecq, H., Gallego, G., and Scaramuzza, D. (2018). "Asynchronous, photometric feature tracking using events and frames," in *Proceedings of the European Conference on Computer Vision (ECCV)* (Munich), 750–765. doi: 10.1007/978-3-030-01258-8_46

Glover, A., and Bartolozzi, C. (2017). "Robust visual tracking with a freely-moving event camera," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Vancouver: IEEE), 3769–3776. doi: 10.1109/IROS.2017.8206226

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.* 18, 1527–1554. doi: 10.1162/neco.2006.18.7.1527

Hunsberger, E., and Eliasmith, C. (2015). Spiking deep networks with lif neurons. *arXiv* 1510.08829.

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cereb. Cortex* 17, 2443–2452. doi: 10.1093/cercor/bhl152

Jiang, Z., Bing, Z., Huang, K., and Knoll, A. (2019). Retina-based pipe-like object tracking implemented through spiking neural network on a snake robot. *Front. Neurorobot.* 13:29. doi: 10.3389/fnbot.2019.00029

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kim, S., Park, S., Na, B., and Yoon, S. (2019). Spiking-yolo: Spiking neural network for real-time object detection. *arXiv* 1903.06530. doi: 10.1609/aaai.v34i07.6787

Kirkland, P., Di Caterina, G., Soraghan, J., Andreopoulos, Y., and Matich, G. (2019). "UAV detection: a STDP trained deep convolutional spiking neural network retina-neuromorphic approach," in *International Conference on Artificial Neural Networks* (Munich: Springer), 724–736. doi: 10.1007/978-3-030-30487-4_56

Kirkland, P., Di Caterina, G., Soraghan, J., and Matich, G. (2020). "Spikeseg: spiking segmentation via STDP saliency mapping," in *2020 International Joint Conference on Neural Networks (IJCNN)* (Glasgow), 1–8.

Lagorce, X., Meyer, C., Ieng, S.-H., Filliat, D., and Benosman, R. (2015). Asynchronous event-based multikernel algorithm for high-speed visual features tracking. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 1710–1720. doi: 10.1109/TNNLS.2014.2352401

Legenstein, R., Pecevski, D., and Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Comput. Biol.* 4:1000180. doi: 10.1371/journal.pcbi.1000180

Levy, S. D. (2020). Robustness through simplicity: a minimalist gateway to neurorobotic flight. *Front. Neurorobot.* 14:16. doi: 10.3389/fnbot.2020.00016

Li, F. F., Fergus, R., and Perona, P. (2004). "Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories," in *2004 Conference on Computer Vision and Pattern Recognition Workshop* (Washington: IEEE), 178.

Li, H., and Shi, L. (2019). Robust event-based object tracking combining correlation filter and CNN representation. *Front. Neurorobot.* 13:82. doi: 10.3389/fnbot.2019.00082

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 120 dB 15micro s latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Long, J., Shelhamer, E., and Darrell, T. (2015). "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Boston, MA: IEEE), 3431–3440. doi: 10.1109/CVPR.2015.7298965

Masquelier, T., and Kheradpisheh, S. R. (2018). Optimal localist and distributed coding of spatiotemporal spike patterns through STDP and coincidence detection. *Front. Comput. Neurosci.* 12:74. doi: 10.3389/fncom.2018.00074

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3:e30031. doi: 10.1371/journal.pcbi.0030031

Masuta, H., Motoyoshi, T., Sawai, K., Koyanagi, K., and Oshima, T. (2017). "Perception and action cycle for cognitive robotics," in *2017 International Symposium on Micro-NanoMechatronics and Human Science (MHS)* (Nagoya: IEEE), 1–7. doi: 10.1109/MHS.2017.8305180

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., and Ganjtabesh, M. (2018). First-spike-based visual categorization using reward-modulated STDP. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 6178–6190. doi: 10.1109/TNNLS.2018.2826721

Mueggler, E., Bartolozzi, C., and Scaramuzza, D. (2017). *Fast Event-Based Corner Detection*. University of Zurich.

Nishiwaki, K., Sugihara, T., Kagami, S., Kanehiro, F., Inaba, M., and Inoue, H. (2003). "Design and development of research platform for perception-action integration in humanoid robot: H6," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)* (*Cat. No.00CH37113)*, Vol. 3 (Takamatsu: IEEE), 1559–1564.

O'Connor, P., Neil, D., Liu, S.-C., Delbruck, T., and Pfeiffer, M. (2013). Real-time classification and sensor fusion with a spiking deep belief network. *Front. Neurosc.* 7:178. doi: 10.3389/fnins.2013.00178

Olah, C., Mordvintsev, A., and Schubert, L. (2017). *Feature Visualization*. Distill. Available online at: https://distill.pub/2017/feature-visualization

Orchard, G., Jayawant, A., Cohen, G. K., and Thakor, N. (2015). Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.* 9:437. doi: 10.3389/fnins.2015.00437

Panda, P., Srinivasan, G., and Roy, K. (2017). *Convolutional Spike Timing Dependent Plasticity Based Feature Learning in Spiking Neural Networks. arXiv preprint arXiv:1703.03854.*

Park, J., Ha, S., Yu, T., Neftci, E., and Cauwenberghs, G. (2014). "A 65k-neuron 73-Mevents/s 22-pJ/event asynchronous micro-pipelined integrate-and-fire array transceiver," in *IEEE 2014 Biomedical Circuits and Systems Conference, BioCAS 2014–Proceedings* (Lausanne: Institute of Electrical and Electronics Engineers Inc.), 675–678. doi: 10.1109/BioCAS.2014.6981816

Paugam-Moisy, H., and Bohte, S. M. (2012). "Computing with spiking neuron networks," in *Handbook of Natural Computing*, eds G. Rozenberg, T. Back, and J. Kok (Springer-Verlag), 335–376. doi: 10.1007/978-3-540-92910-9_10

Paulun, L., Wendt, A., and Kasabov, N. (2018). A retinotopic spiking neural network system for accurate recognition of moving objects using neucube and dynamic vision sensors. *Front. Comput. Neurosci.* 12:42. doi: 10.3389/fncom.2018.00042

Renner, A., Evanusa, M., and Sandamirskaya, Y. (2019). "Event-based attention and tracking on neuromorphic hardware," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (California, CA: IEEE), 1709–1716. doi: 10.1109/CVPRW.2019.00220

Seifozzakerini, S., Yau, W.-Y., Zhao, B., and Mao, K. (2016). "Event-based hough transform in a spiking neural network for multiple line detection and tracking using a dynamic vision sensor," in *Procedings of the British Machine Vision Conference 2016* (York, PA: British Machine Vision Association), 94.1–94.12. doi: 10.5244/C.30.94

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. (2019). Going deeper in spiking neural networks: VGG and residual architectures. *Front. Neurosci.* 13:95. doi: 10.3389/fnins.2019.00095

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: visualising image classification models and saliency maps. *arXiv* 1312.6034.

Stromatias, E., Soto, M., Serrano-Gotarredona, T., and Linares-Barranco, B. (2017). An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data. *Front. Neurosci.* 11:350. doi: 10.3389/fnins.2017.00350

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neural Netw.* 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Vasco, V., Glover, A., and Bartolozzi, C. (2016). "Fast event-based Harris corner detection exploiting the advantages of event-driven cameras," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Daejeon: IEE), 4144–4149. doi: 10.1109/IROS.2016.7759610

Wiesmann, G., Schraml, S., Litzenberger, M., Belbachir, A. N., Hofstatter, M., and Bartolozzi, C. (2012). "Event-driven embodied system for feature extraction and object recognition in robotic applications," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*

Workshops (Providence, RI: IEEE), 76–82. doi: 10.1109/CVPRW.2012.6238898

Xie, M. (2003). *Fundamentals of Robotics, Volume 54 of Series in Machine Perception and Artificial Intelligence*. Singapore: World Scientific.

Zamani, F., and Jamzad, M. (2017). A feature fusion based localized multiple kernel learning system for real world image classification. *EURASIP J. Image Video Proc.* 2017:78. doi: 10.1186/s13640-017-0225-y

Zeiler, M. D., and Fergus, R. (2014). "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision* (Springer), 818–833. doi: 10.1007/978-3-319-10590-1_53

Zitnick, C. L., and Dollár, P. (2014). *Edge Boxes: Locating Object Proposals From Edges*. Cham: Springer.

Conflict of Interest: GM was employed by the company Leonardo MW Ltd.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 05 November 2020
doi: 10.3389/fnbot.2020.567991

Echo View Cells From Bio-Inspired Sonar

Jacob D. Isbell[1]* *and Timothy K. Horiuchi*[1,2,3]

[1] Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, United States, [2] Institute for Systems Research, University of Maryland, College Park, MD, United States, [3] Program in Neuroscience and Cognitive Science, University of Maryland, College Park, MD, United States

Place recognition is naturally informed by the mosaic of sensations we remember from previously visiting a location and general knowledge of our location in the world. Neurons in the mammalian brain (specifically in the hippocampus formation) named "place cells" are thought to reflect this recognition of place and are involved in implementing a spatial map that can be used for path planning and memory recall. In this research, we use bat-inspired sonar to mimic how bats might sense objects in the environment and recognize the views associated with different places. These "echo view cells" may contribute (along with odometry) to the creation of place cell representations observed in bats. Although detailed sensory template matching is straightforward, it is quite unlikely that a flying animal or robot will return to the exact 3-D position and pose where the original memory was captured. Instead, we strive to recognize views over extended regions that are many body lengths in size, reducing the number of places to be remembered for a map. We have successfully demonstrated some of this spatial invariance by training feed-forward neural networks (traditional neural networks and spiking neural networks) to recognize 66 distinct places in a laboratory environment over a limited range of translations and rotations. We further show how the echo view cells respond between known views and how their outputs can be combined over time for continuity.

Keywords: bat, echolocation, place cells, place fields, robotics, sonar, neural network, skim

OPEN ACCESS

Edited by:
Subramanian Ramamoorthy, University of Edinburgh, United Kingdom

Reviewed by:
Ninad B. Kothari, Johns Hopkins University, United States
Nicolas Cuperlier, Equipes Traitement de l'Information et Systèmes, France

***Correspondence:**
Jacob D. Isbell
jisbell@umd.edu

Received: *31 May 2020*
Accepted: *08 October 2020*
Published: *05 November 2020*

Citation:
Isbell JD and Horiuchi TK (2020) Echo View Cells From Bio-Inspired Sonar. Front. Neurorobot. 14:567991. doi: 10.3389/fnbot.2020.567991

INTRODUCTION

The hippocampal formation in the mammalian brain is well-known for its population of "place cells," a type of neuron that responds when an animal is in a particular place in its environment. Studies in the rat suggest that these cells use internal odometry signals (allowing the system to operate in darkness) as well as external sensory cues (allowing the system to recognize places and correct the odometry system) (O'Keefe, 1976; Jung et al., 1994). In the flying, echolocating bat, neurons with very similar properties have been found (Ulanovsky and Moss, 2007; Yartsev et al., 2011; Yartsev and Ulanovsky, 2013; Geva-Sagiv et al., 2015). Unlike rats, bats have the uncommon ability to perceive the three-dimensional locations of objects by actively emitting sounds and localizing the reflections (Wohlgemuth et al., 2016), allowing the bat to navigate where other sensory systems, such as vision, are ineffective. Although the signal processing and neural mechanisms with which bats *recognize* places is still largely unknown, modeling this capability with biologically-plausible sensors and robotics can give us insights into problems that bats encounter and motivate future behavioral and neurophysiological experiments with bats.

Although most robotic explorations into mapping and navigation have focused on variants of the SLAM (simultaneous localization and mapping) algorithm using light-based sensors (e.g., computer vision or LIDAR) (Strösslin et al., 2005; Bachelder and Waxman, 2011) for metrically-accurate maps, little work has been done exploring how a bat might use sonar to accomplish the same task. One good example is that of Steckel and Peremans (2013) that used a biomimetic sonar device on a mobile, ground robot to navigate and map different office and laboratory environments, however, the SLAM algorithm used was not meant to be a model of a biological system. The work presented here addresses the question whether place cells can be recognized over an extended region using only a narrow-band (∼40 kHz) sonar in a laboratory environment. Unlike the place cells that signal when the animal is in a particular area (i.e., the "place field") based on a combination of odometry and sensory inputs, we are constructing "echo view cells" that recognize previously encountered views (i.e., an "echo fingerprint") based solely on sonar. Phenomenologically similar to primate "spatial view cells" that are active when the animal is gazing at a particular set of objects (over a limited field-of-view), these echo view cells recognize previously memorized echo patterns. Unlike primate spatial view cells, however, object range is included in the pattern and thus the echo view cells fire over a small region of the environment.

A neural network model was used to implement echo view recognition that incorporates concepts from machine learning related to pattern separation and classification. A key aspect of this investigation is the attempt to bridge the gap from high-dimensional, low-level, sensory inputs to the more symbolic, discrete nature of place recognition that is critical to higher-level cognitive models of path planning (Koul and Horiuchi, 2019). A key goal is to ensure that the echo view cells respond over a wider area and not just to a single coordinate in space. One limitation of the work is that only limited information is available from the narrowband sonar (typical objects are represented by only a few echoes) and object recognition was difficult, preventing a landmark-based approach, as is common for visual place recognition algorithms. Instead, views were recognized based solely on the spatiotemporal pattern of echoes allowing the memorization of views in a variety of environments without prior training of an object recognition layer. From view recognition, direction-independent place recognition can be constructed in convergence with odometric information. Such approaches to place recognition with sonar have been used (Ollington and Vamplew, 2004; Vanderelst et al., 2016). One challenge with sonar is that small changes in the position and angle (particularly in man-made environments) can produce large changes in the resulting echo pattern. Multi-path reflections are also sensitive to positioning. To explore this, data was taken with a large variety of small changes to the positioning of the sonar.

This work explores two very different neural networks that can achieve this: a single layer neural network operating on a recorded echo pattern presented as an image, and a biologically-realistic, spiking neural network (SNN) presented with echoes in the time domain to simulate live sonar signals. In addition to our motivations to ultimately model and understand the biological implementation of sonar-guided behavior (mentioned above), this work has applications for mobile, autonomous robotics. There are many circumstances when a drone may need to navigate in a dark building for stealth, through a building filled with smoke, through a forest with dense fog, or through tunnels filled with dust. Since standard cameras and LIDAR do not work well in these environments, sonar is a reasonable alternative or complementary sensor. Sonar has been shown to be useful for obstacle avoidance (Eliakim et al., 2018). Currently, the most common use of sonar systems is underwater. Since the speed of sound is much faster underwater, the effective range and efficiency of sonar is greatly increased underwater. Current laser and radar systems consume much more energy than a sonar system; this would reduce the robot's field time and potential range (Jiang et al., 2010). The weight and cost of radar systems can also reduce their feasibility of use. One can imagine a lightweight, flying drone that can quickly maneuver through a dark house and provide a map based more on sensory features and not metrical details, closer to the way humans communicate with each other.

MATERIALS AND METHODS

Hardware

The sonar system used in the work presented here consists of three custom-modified MaxBotix® sonar transducers, similar to the MaxBotix XL-MaxSonar®-EZ™ commercial series of sonar range finders, a custom PIC® 18F2620 microcontroller-based sonar controller board, a Futaba S148 hobby servo, and a computer interface to both record and display echo signals and control the servo to orient the sonar (shown in **Figure 1**). The transducers act as both a speaker and a microphone. They resonate at 40 kHz and will only detect signals near this frequency. The custom sonar boards report a logarithmically-compressed envelope signal as an analog voltage. This compression allows the output to report the very wide dynamic range of amplitudes that occurs with sonar without saturating. The maximum working range of this sonar is 7.65 m. These transducers were custom modified to provide more control over the timing and duration of the outgoing pulse, a louder outgoing pulse, and access to a log-compressed envelope of the transducer response. All these functionalities are now commercially available through MaxBotix. The transducers are placed in a 3-D printed mount (shown in **Figure 2**) on the servo motor. In this demonstration system, the transducers transmit and receive over a cone of about +/− 30 degrees (−6 dB beam width), so the transducers are held facing 30 degrees apart to ensure sufficient overlap and coverage of the area in front of the transducers for binaural localization based on interaural level differences. The ultrasonic pulse trigger-timing and analog-to-digital (A/D) conversion is done by the microcontroller. The majority of the data processing is performed on the microcontroller to ensure a quick response.

The sonar system executes four steps: pulsing, sampling, processing, and communicating. A short duration pulse voltage (∼0.25 ms) is supplied to the transducer, however, due to the resonant quality of the transducer, the emitted sound has a ring-up and ring-down period, resulting in an extended pulse duration of about 1 ms. Following the pulse, the transmitting transducer

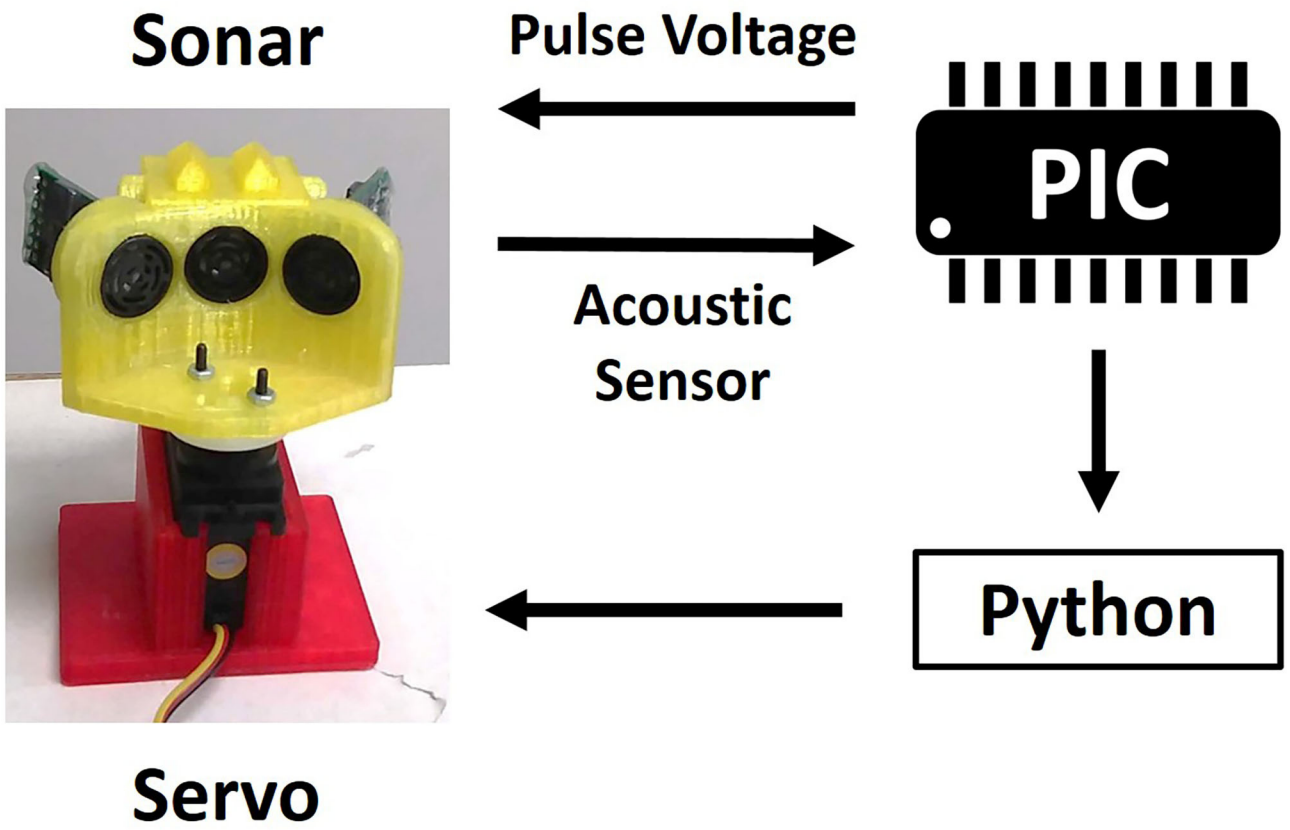


FIGURE 1 | The flow of signals through the hardware. The microcontroller sends pulse voltages to the transducers and reads the acoustic voltage off the transducers. This data is sent to Python (van Rossum, 1995) on a PC, which also controls the servo motor.

continues to ring for several milliseconds. Echoes can be detected during this ringing period once the amplitude has diminished sufficiently, so a short two millisecond delay is incorporated before sampling begins. The log-compressed envelope voltage is sampled every 8th of a millisecond, a sampling frequency of 8 kHz. An object is detected when the temporal derivative of the envelope switches from positive to negative, denoting a peak. The range is determined by finding the time when the envelope reaches its peak value. Envelope voltages on all transducers are recorded at the time of the peak. Our sampling time of an 8th of a millisecond gives us a range resolution of 2.14 cm or 0.84 in. We sample for 255 time bins, giving us a range of 5.5 m or 18 ft. Following the sampling period, echo data is transferred via serial interface to a PC and all further processing on the information is performed on the PC. An example of this data is shown in **Figure 3**.

The code used on our microcontroller is available at https://github.com/jacob-isbell/sonarPIC/blob/master/PICcode.asm.

Dataset Description

Data was recorded in our laboratory and the adjoining hallway. 66 different recording locations were spread throughout this environment. Locations were spaced 2 feet apart where possible, forming a grid-like placement (**Figure 4A**). No attempt was made to restructure the objects in the lab to accommodate the sensing; things were left as they were. No objects were moved during the recording at different locations.

To capture a broader view, a variety of data was collected at different translations and rotations within each square at each of the 66 locations. Across 1 square foot, data was recorded at 25 different translations inside a 5 × 5 square grid with a 3 inch (7.6 cm) spacing. At each of these 25 points, data was recorded at 11 different angles, ranging from −5 to +5 degrees in 1 degree increments (**Figures 4B,C**). Ten samples were taken at each angle. In total, each square location has: (25 translations) × (11 angles) × (10 repetitions) = 2,750 sonar images per location. With 66 locations, the full data set consists of 181,500 sonar images.

Echo Fingerprint Recognition

Two different neural network architectures were tested for their ability to recognize which of the 66 locations a sonar pulse came from. A conventional, single layer network was used and a biologically-plausible, temporally-based architecture called the Synaptic Kernel Inverse Method (SKIM) (Tapson et al., 2013) was used. The inputs and outputs of both networks were similar. The inputs consisted of one sonar image. 255 range bins were

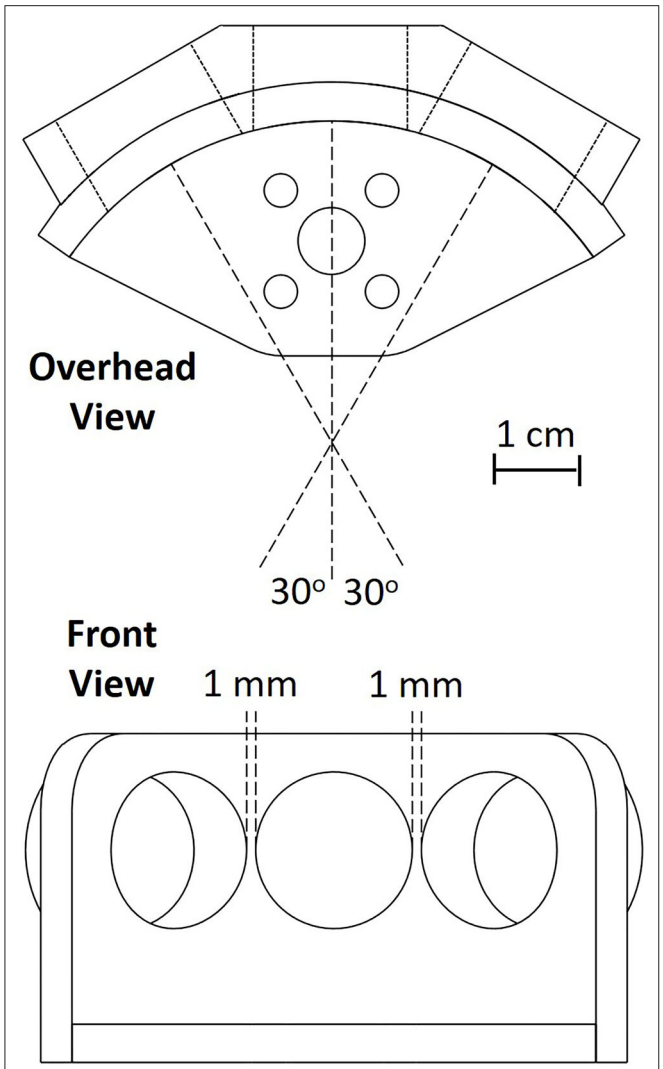


FIGURE 2 | The schematic used for the 3-D printing of the sonar transducer holder. The sonar system consists of three sonar transducers oriented with 30 degree angular separation mounted on a rotating servo motor (not shown).

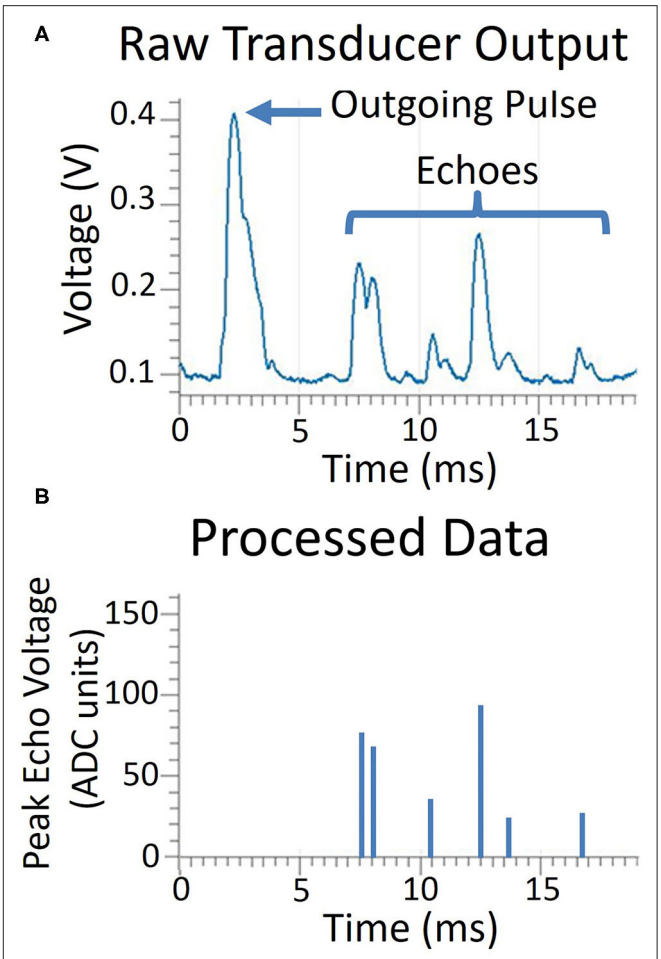


FIGURE 3 | A comparison of the raw transducer data and the processing done on the microcontroller. In the top graph **(A)**, the transducer voltage is shown as it produces the outgoing pulse and receives the echoes. Only peak magnitude and time of echo peak are processed and recorded, shown in the bottom graph **(B)**.

used with data from the 3 transducers, resulting in a 765-dimensional input vector. The envelope amplitude data was supplied to the network. If there was no echo in a time bin, the value was kept as zero. The resolution of each range bin was 2.14 cm or 0.84 in. Each sonar image was L2 normalized before being fed to the network. While normalizing means the network doesn't have direct access to the echo magnitudes, the relative magnitude between echoes contains more reliable and reproducible information, such as the magnitude difference between transducers which relates to echo direction. Each output corresponds to a different location, so with 66 locations there are 66 outputs. In both networks, a form of supervised learning was used to train the network.

Although the angle of an arriving echo could be calculated using the magnitude difference between the transducers (e.g., using interaural level differences) to reduce the dimensionality, we chose to retain the raw values and let the network learning rules determine how this information would be used.

Single Layer Feedforward Network

In this experiment, a very simple neural network was used to process the data. The network consisted of the input layer fully connected by weights to the output layer (**Figure 5**). The non-linear logistic function was applied to the summation of weighted inputs to provide the output. Learning was performed by a modified version of gradient descent that uses an adaptive momentum term to speed learning, called the AdamOptimizer algorithm (Kingma and Ba, 2014). This was implemented in the machine-learning software package, TensorFlow (Abadi et al., 2015) on Google's Colaboratory cloud computing platform (Bisong, 2019), allowing us to speed up the training with free use of their GPUs.

In this task, the single layer network performed as effectively as multiple-layer networks and its simplicity led to an easier

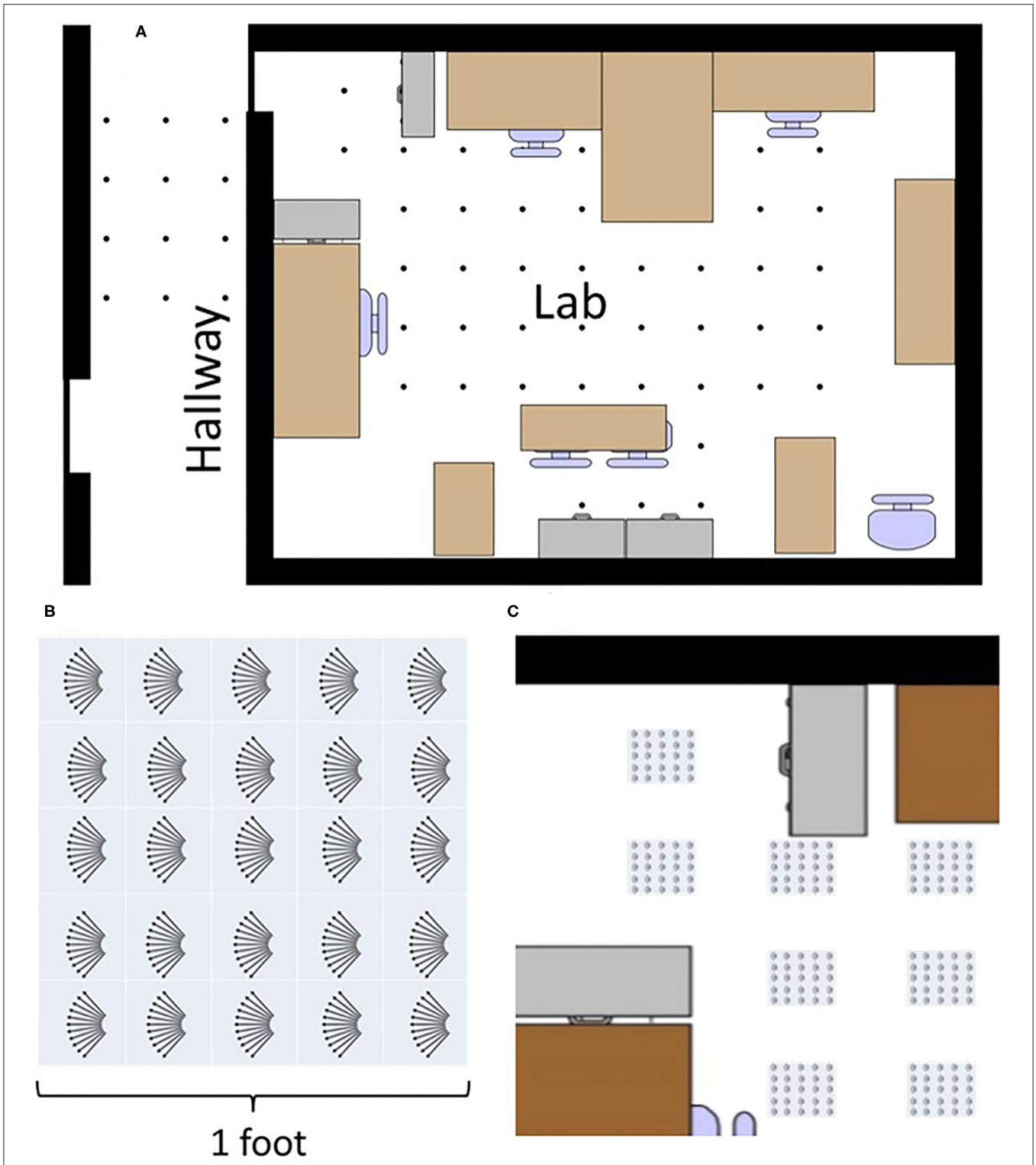


FIGURE 4 | The top image **(A)** shows a map of places data was recorded. Every dot is a recorded place. Locations and objects are approximately to scale. Bottom left **(B)** shows how a variety of data was recorded at different translations and rotations at each point in the lab. Eleven angles were recorded along five rows and five columns giving 275 recordings at each place. Bottom right **(C)** shows explicitly how data was recorded at each place, capturing a large variety of data throughout the lab.

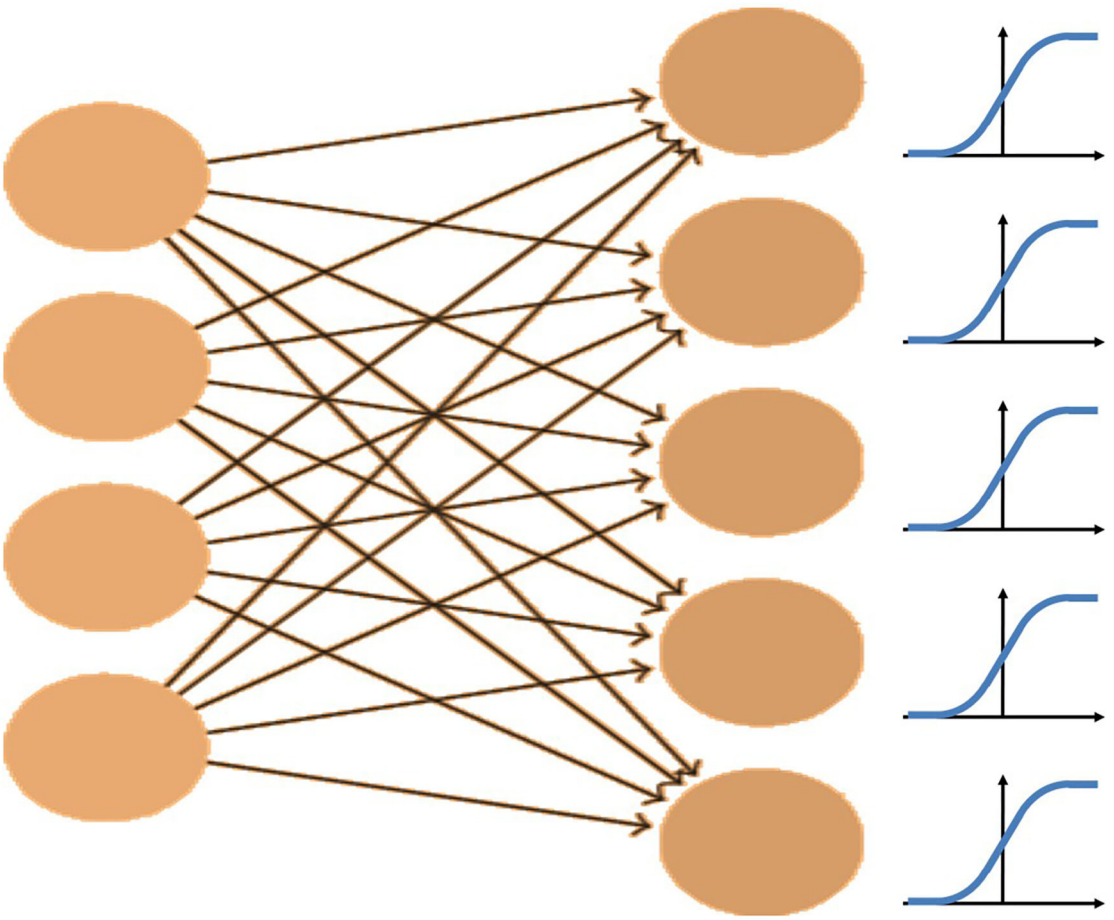


FIGURE 5 | The network architecture for the single layer network. There is one layer of fully connected weights from the inputs to the outputs. Each output has a logistic non-linearity applied to it to maintain outputs between 0 and 1.

observation and analysis of how the network was solving this problem.

Synaptic Kernel Inverse Method (SKIM)

SKIM is a multi-layer network architecture that combines the benefits of Extreme Learning Machines (ELM) but with spiking neuron (temporal) representations. Sonar lends itself to being represented in the spiking domain because echoes themselves are inherently time-based signals and typically pulsatile in nature. The temporal nature of this network suggests a real-time implementation using spiking neuromorphic hardware (Moradi et al., 2017). **Figure 6** illustrates the SKIM network architecture (Tapson et al., 2013).

The first layer of weights in the SKIM network consists of fixed, random weights connecting the inputs to the hidden layer. These weights can be positive or negative. The fanout here is usually 10–20 (or a hidden layer that has 10–20 times more neurons than the input layer), resulting in a very large hidden layer. This is typical of an ELM approach, which aims to expand the dimensionality of the input data to make pattern separation easier (Huang et al., 2011). There is also a non-linearity applied at each hidden unit. Every hidden unit has a randomly selected temporal synaptic kernel associated with it that consists of a time delayed alpha function. If A is the activation of the unit, t is the time, ΔT is the delay, and τ the width of the alpha function, the equation is:

$$\mathbf{f(t) = tanh(A}\ \frac{\mathbf{t - \Delta T}}{\tau}\mathbf{e}^{-\frac{\mathbf{t-\Delta T}}{\tau}}\mathbf{)}$$

where different hidden units have different delays (ΔT) and widths of the alpha function (τ) (**Figure 7**). The time delay is essential to recognizing patterns that occur over time,

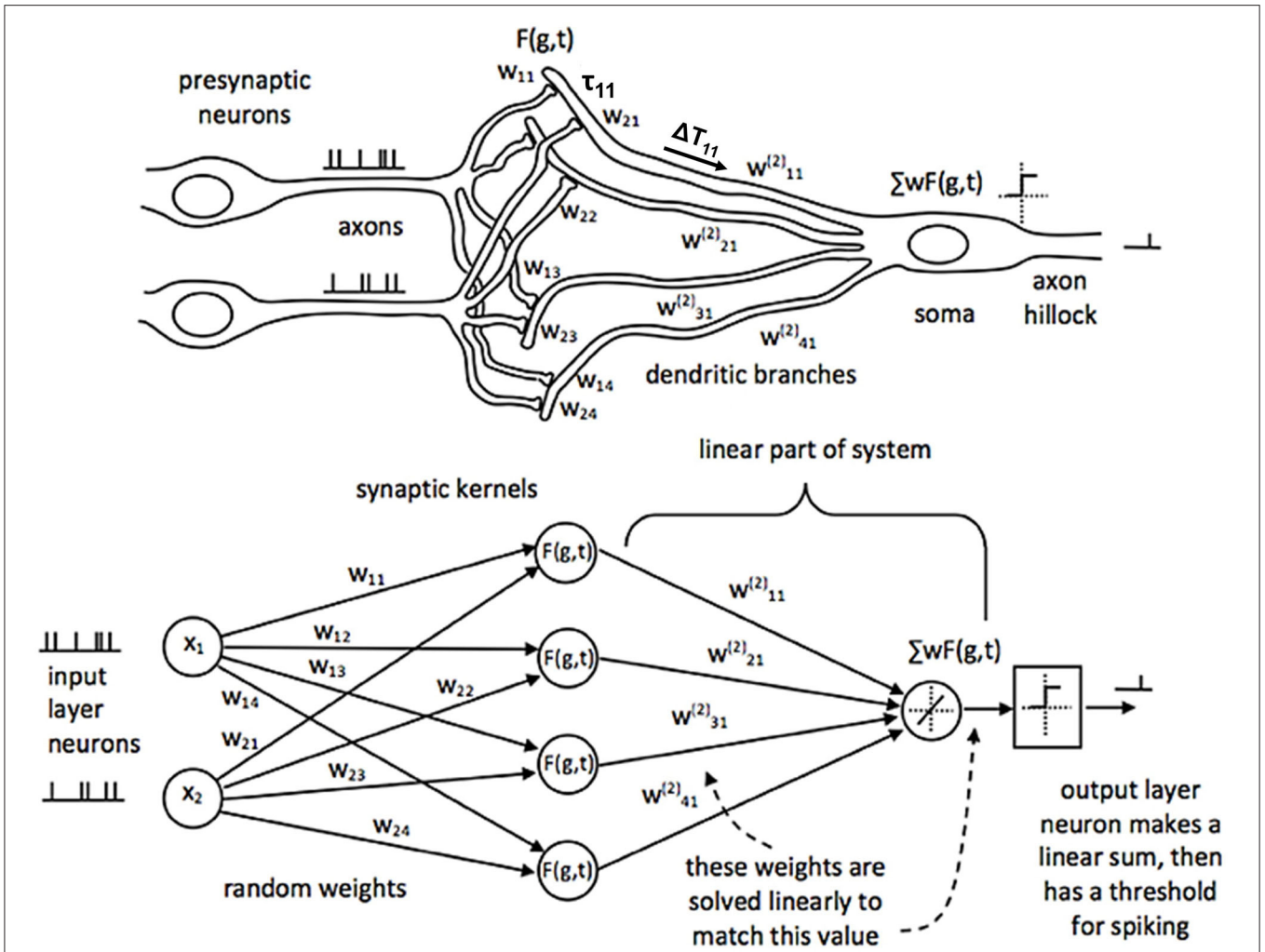


FIGURE 6 | Adapted from Tapson et al. (2013). The architecture for the SKIM neural network. The top of the figure shows what a corresponding biological system would look like, while the bottom shows this network from a computational perspective. Inputs from the presynaptic neurons are summed onto the dendrites of the postsynaptic neurons. Each dendrite has an associated non-linear, synaptic kernel [$F(g,t)$] with a time constant (τ), and dendritic delay (ΔT). The dendritic activity is summed onto the soma and creates a spiking output when above a threshold. The weights from the input layer to the hidden layer are static (w_{xy}); the linear connection from the hidden layer to the output has weights that are trained ($w^{(2)}_{yz}$).

and gives the network a form of memory, a way to be influenced by data in the past. A compressive non-linearity (the hyperbolic tangent, tanh) is applied as well. These hidden units create a high-dimensional, non-linear transformation of the input data that has occurred recently in time. This allows for complex, temporal patterns to be more easily recognized and separated.

The next layer of this network is linear. There are a set of fully-connected weights from the hidden layer units to the output. These are the weights that are modified during learning. Since this is the only dynamic part of the network, the learning is simplified. As this is a linear transformation with a known hidden-unit activation and a known output (since we are performing supervised learning), the weights can be solved for analytically.

If M is the number of hidden units and k is the number of time steps in our dataset, we obtain a matrix describing the hidden unit activation over time, $H \varepsilon \mathrm{R}^{Mxk}$. If N is the number of outputs, we have the output activation matrix, $Y \varepsilon \mathrm{R}^{Nxk}$. The weights connecting the two layers will be $W \varepsilon \mathrm{R}^{NxM}$, such that $WA = Y$. To find the weights we simply have to solve for W, giving $W = YA^{+}$, where A^{+} can be found by taking the Moore-Penrose pseudoinverse of A.

To solve this analytically, we use the Online PseudoInverse Update Method (OPIUM) (Tapson and van Schaik, 2013). This is an application of Greville's method, which shows an incremental solution to finding the pseudoinverse, but is adapted and simplified for this specific problem to reduce the needed computation without losing accuracy.

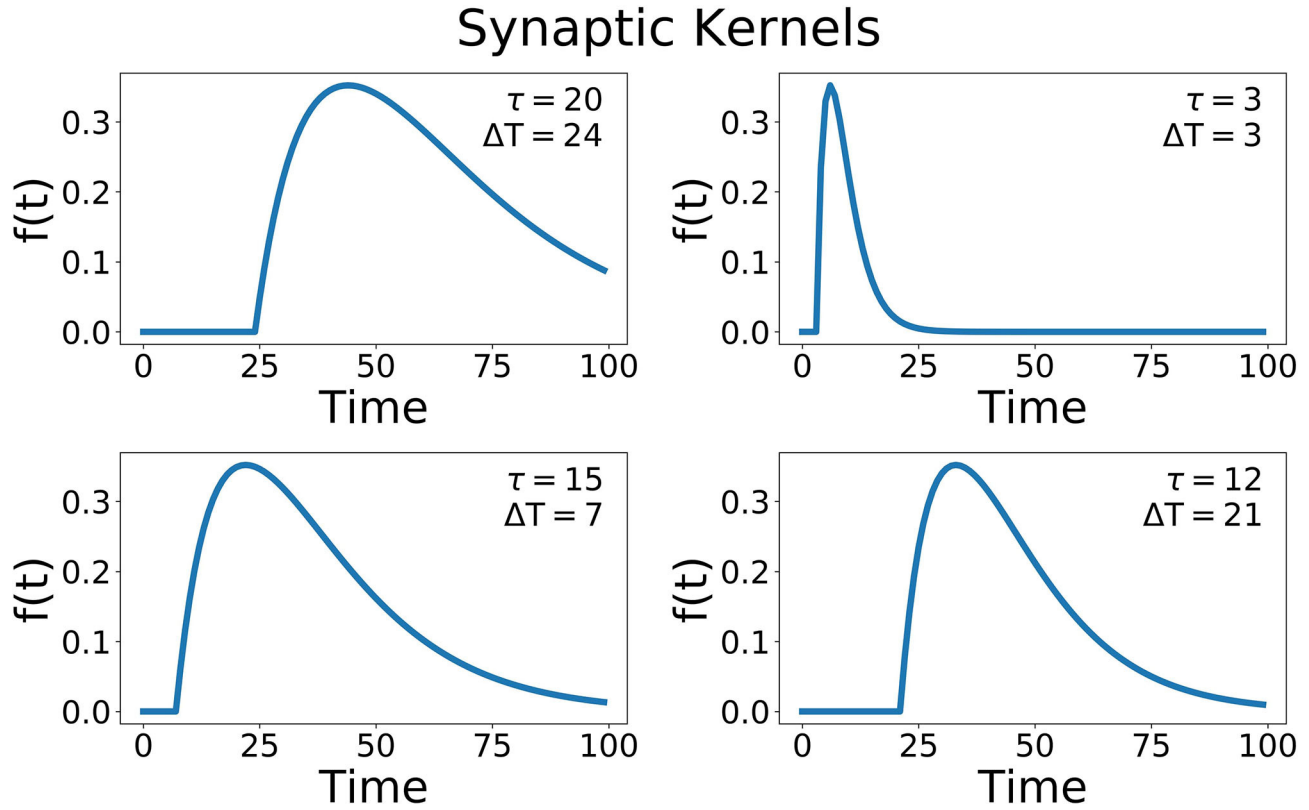


FIGURE 7 | Some example synaptic kernels. Two parameters are changed, the delay for the onset of the function (ΔT), and the width of the alpha function (τ). The x-axis corresponds to the variable t of this function.

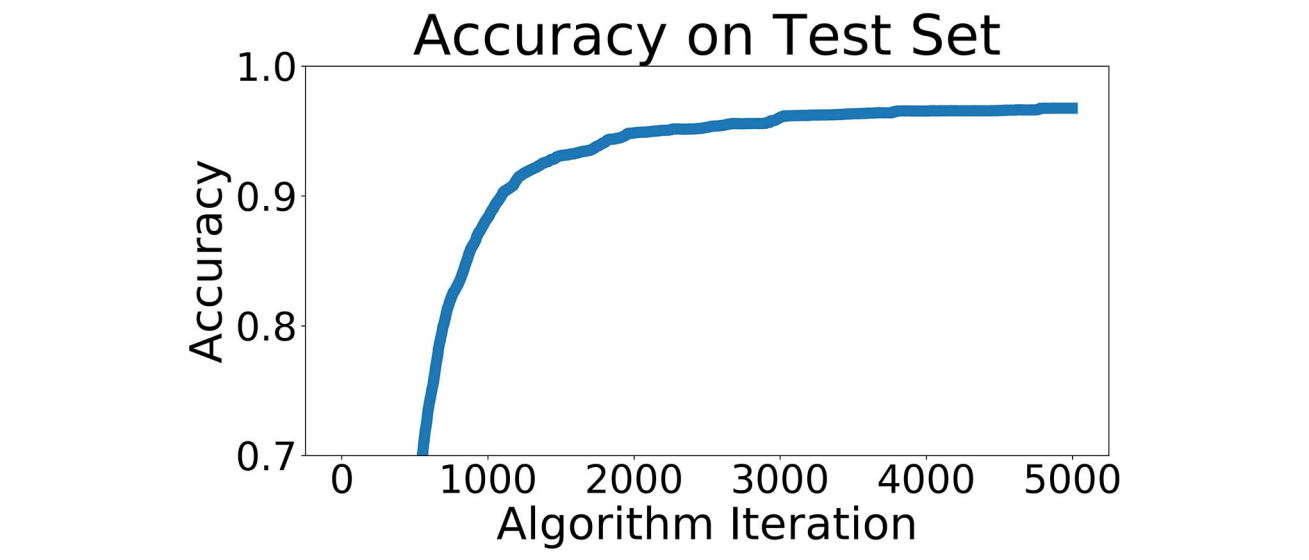


FIGURE 8 | Network accuracy as training progresses. Each algorithm iteration takes ∼0.5 s, and the network takes about 1 h to train.

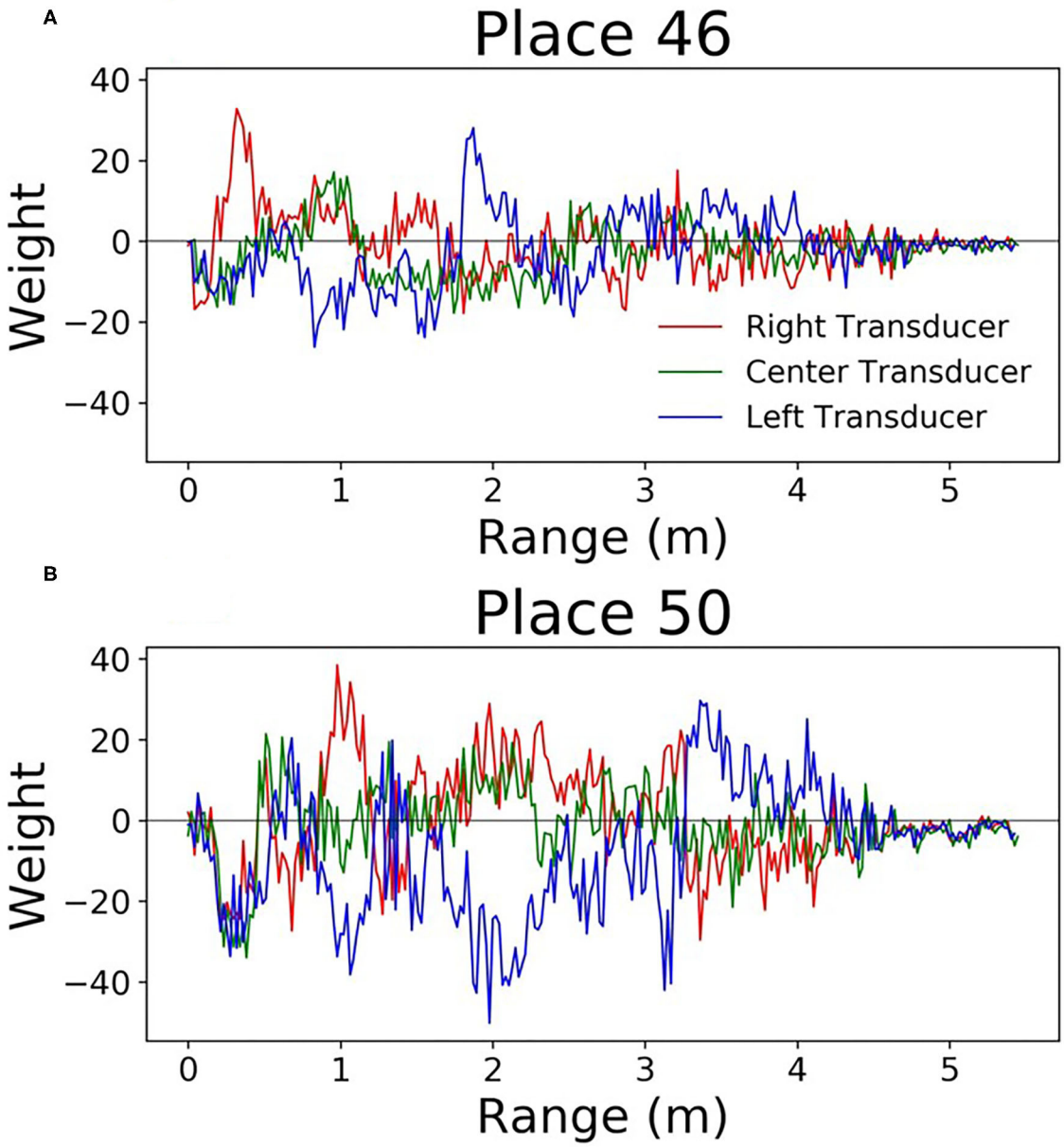


FIGURE 9 | Perceptive fields of the output neurons in the single layer network **(A,B)**. It's clear that in some spots these perceptive fields split the left and right signals. This gives the network the ability to discriminate direction.

RESULTS

Single Layer Network

This network was trained to predict which of the 66 locations a sonar pattern came from. The recorded sonar dataset was split into three parts, 80% training data, 10% testing data, and 10% validation data. The data was randomly shuffled across locations and positions within locations before being split into these three groups. Our accuracy of identifying the location of a particular pattern from the validation data set reached 97.5%. A graph of the accuracy across the training regimen is shown in **Figure 8**.

Since this network is very simple, it is easy to understand how the weights can be interpreted. Each output neuron has a weight corresponding to every input. These can be thought of as the perceptive field of this output neuron. By looking at which inputs cause the output to activate, we can get an idea of the sonar image preferred by each output neuron. **Figure 9** shows some example weights from the network. One noticeable pattern in these weights is the splitting that occurs between the right and left transducers; there are clear ranges where one will be positive and the other will be negative. Functionally, this is

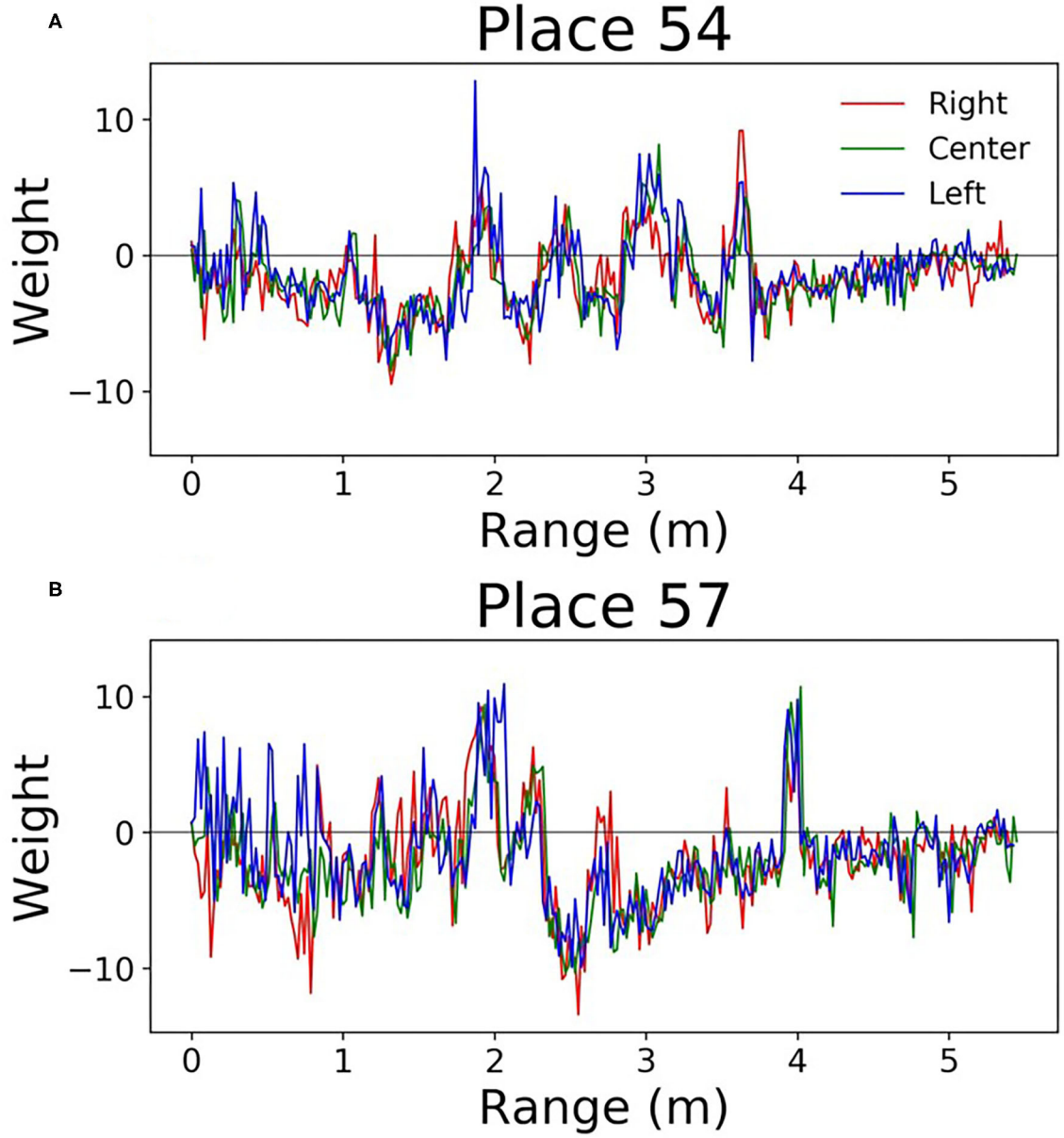


FIGURE 10 | Perceptive fields of the output neurons in the single layer network. These **(A,B)** are from the hallway data. These weights were of lower amplitude, and all transducers were correlated with one another.

the network learning to look for objects at a certain angular orientation. Another clear pattern that arose in the network weights; the weights from the hallway seemed to be synchronized across transducers (**Figure 10**). These weights were also lower in amplitude than those from inside the lab.

Figures 11A–D shows how the different view cells responded across the whole map. It is clear that the network learned very rigid boundaries where it was trained to do so. Although this demonstrates a successfully trained network, the sharp distinctions between neighboring locations is not what is seen in mammalian place cells.

SKIM

In the SKIM network trained with OPIUM, we achieved up to 93.5% accuracy on our dataset. The choice of time constants (τ, the alpha function widths) and delays (ΔT) for the synaptic kernels was very important. The time constants determine the temporal precision the network can observe; large time constants lead to less temporal precision. Long time constants provide tolerance to temporal jitter between patterns but result in a loss of temporal discrimination when needed. The time constants used for this network covered one to five time bins, with τ's randomly chosen between 0.5 and 1.5, keeping a relatively narrow

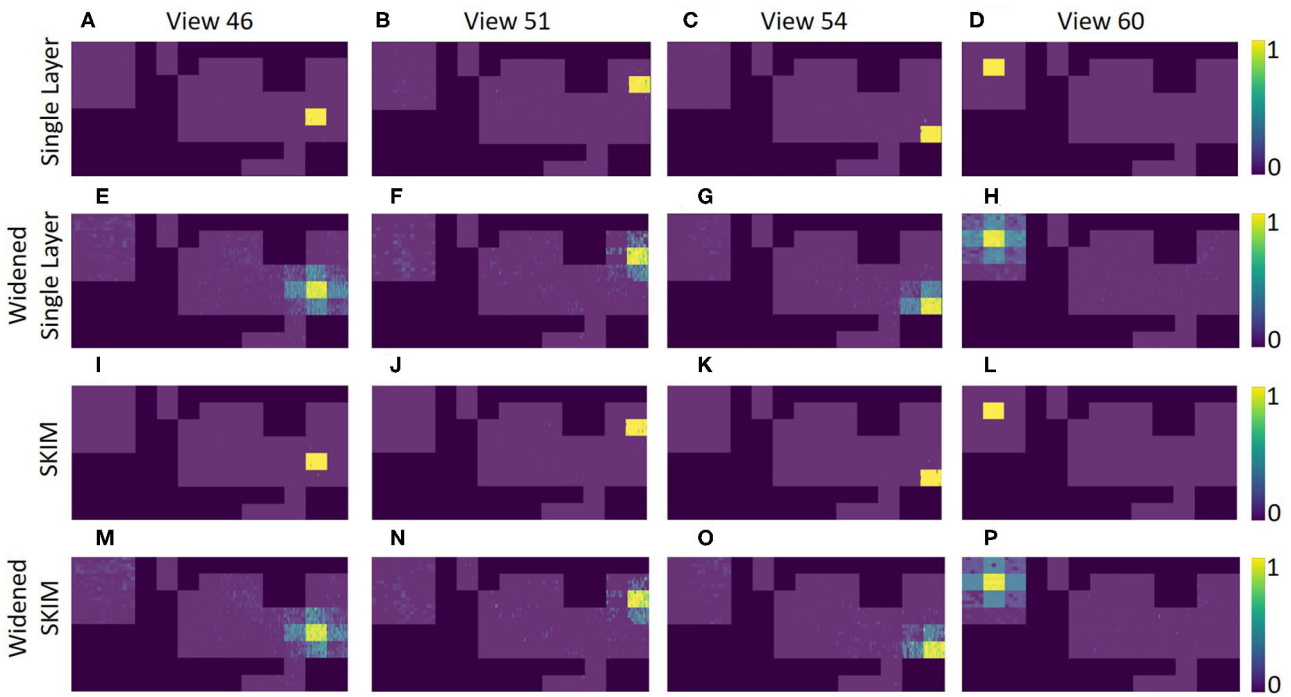


FIGURE 11 | An overhead view of the different echo view fields created by the two networks. This map is the same as shown in the top of **Figure 4**. Each plot represents a different echo view cell's activation across the entire map. The top plots **(A–D)** show the original network for four different views, and the plots **(E–H)** shows the single layer network with widened labels, resulting in neighboring views being activated. Plots **(I–P)** show the view activations for the SKIM network and the widened SKIM network. It is important to note that only areas in the training dataset are displayed. The 1 foot squares in between each of the locations have been omitted (shown explicitly in **Figure 4C**).

and precise response. The choice of delays determined which temporal part of the data is relevant (i.e., beginning, middle, end of the pulse). The delays were distributed randomly over the length of the sonar pulse to ensure that all the echoes had an equal probability of activating the network, with $\Delta T^{'}s$ randomly chosen between 0 and 255. The network was trained to deliver an output at the end of a sonar image ($t = 255$). Accuracy was determined by taking the output neuron with the highest activation at t = 255. **Figures 11I–L** shows how the SKIM view cells responded across the whole map. The response is very similar to the single layer network with rigid boundaries between views.

Recognition Outside of Training Data

Outside of the locations (squares) where data was collected, both networks does not predictably recognize that it is near a known location. The accuracy was high when in an area it was trained on, but recognition drops quickly even inches away. **Figures 12A–C** shows this for the single layer network; **Figures 12G–I** shows this for the SKIM network. To spread the activation of the network to neighboring areas outside the training area, network training was changed. Instead of an output neuron being trained to 1.0 in its corresponding location and all other neurons trained to 0.0, neighboring neurons were trained to respond to neighboring views. A Gaussian function was used, giving adjacent views an activation of 0.5 and diagonal views and activation of 0.38. After this round of training the accuracy of the single layer network dropped to 92.3%, while the accuracy of the SKIM network remained stable at 93.4%. **Figures 12D–F,J–L** show the results of this new training for the single layer network and SKIM network, respectively. The new activation pattern of the network is now spread through areas that were not explicitly trained on, and qualitatively looked more like biological place fields. **Figures 11E–H,M–P** also shows how these new view cells respond across the whole map. There is now more noticeable activation in areas that were not trained on. The cells have become much more broadly tuned. We call this new network the "widened" network, in contrast to the "original" network. The single layer network and the SKIM network responded very similarly in all the cases presented.

DISCUSSION

Functionality Test Along a Path

To demonstrate how this system might be used in practice, sonar data was recorded along a path consisting of points both inside and outside of the training data. The single layer network's response to this data shows how views can be recognized along the entirety of this path (**Figure 13**).

The widened network, which allows multiple view neurons to be active at once, creates a broader, more spatially-continuous response when compared with the original network. Less reliance

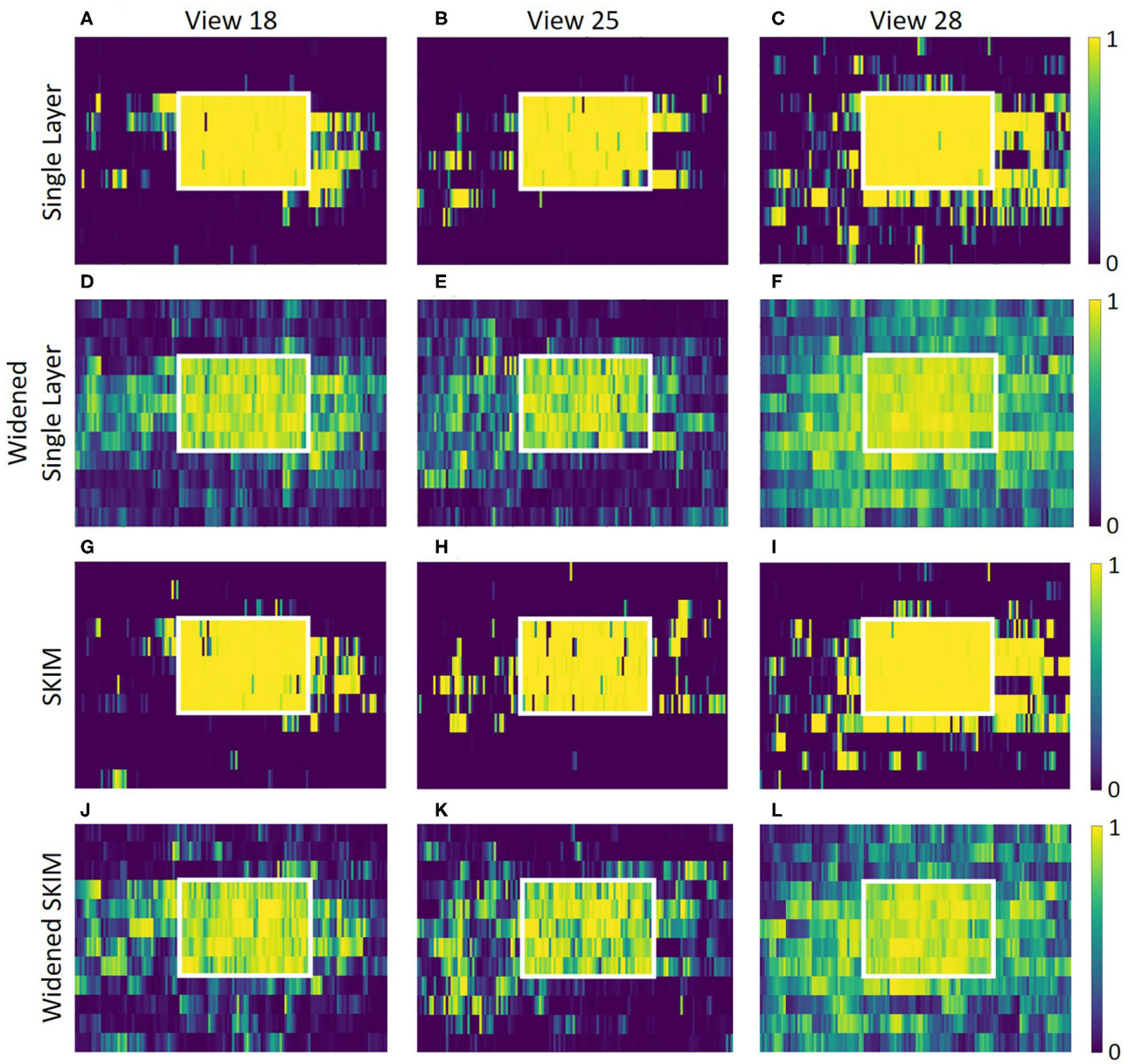


FIGURE 12 | Each graph presents an overhead view of a location. Inside the white square is where training data was recorded; outside the white square is an adjacent area that was not used for the training of the networks. Along the x axis, eleven adjacent pixels show the eleven angles for each of the 25 (5 × 5) spatial positions inside the white box. Pixels along the y axis are spaced evenly. These view neuron activation patterns are generated by the corresponding output neuron from the neural network. The top plots **(A–C)** show how the single layer network responds around these locations, showing sparse activation outside the trained square and very high activation inside the square. Plots **(D–F)** show the single layer network trained to respond to neighboring views. Plots **(G–L)** show the same information, but for the SKIM network and the widened SKIM network. The widened networks show a much more spread out activation in the non-trained area outside the square.

on a single view neuron activating provides a more stable and nuanced interpretation of location. In situations where the original network fails to activate the correct view neuron, the widened network is more likely to alleviate the situation by activation of other nearby view neurons.

Leaky integration was also used to help smooth out the network response over time; each activation is given an exponentially decreasing tail over time. With A_t as the activation for a position at time t, and L_t as the activation for a position after leaky integration is applied, the equation used is $\mathbf{L_t}=\alpha\mathbf{A_t}+(\mathbf{1}-\alpha)\,\mathbf{L_{t-1}}$. In this example, one view is about 10 movements wide. Using a leaky integration constant (α) of 5/9 allows for activation to be maintained at %10 of its original value 10 time steps in the future, allowing persistent activation while moving across a position at the cost of a slight lag. An equivalent way to calculate this would be to have each

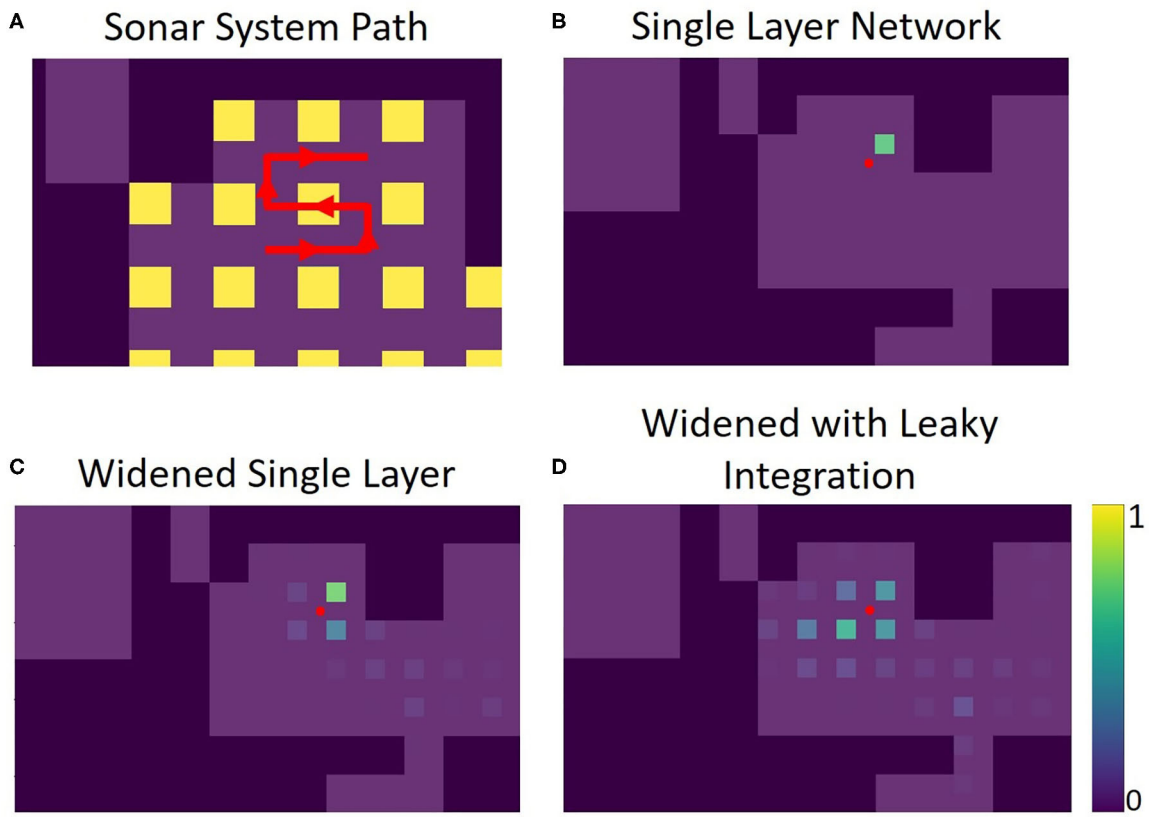


FIGURE 13 | Panel **(A)** shows the path the sonar system moves through, in red. There are 39 positions total along this path, each position 3 inches from the last. The portion of the path within the yellow squares is contained in the training set for the networks (5 of the path positions). The rest of the path was not used for the training of the networks. Panels **(B–D)** show echo view field responses on the path. The red dot represents the position of the sonar. The activations of the echo view cells are shown in their corresponding location, seen as colored squares on the plots. The single layer network was trained to have only one view cell active at a time. The widened network allows for more cells to be active at once, improving accuracy in between trained views. The leaky integration maintains a more stable activation due to its use of the past activations in the path.

activation exponentially decaying over time; the corresponding time constant would be 4.5. In some locations on the path, the sonar is not able to correctly recognize the view. For this example, integration over time gives the network more stability and accuracy. **Supplementary videos** show the activations of the original network, the widened network, and the leaky integration applied to the widened network similar to **Figure 13**, but over the entire path.

The widened network with leaky integration gives consistently accurate results over the whole path. The echo view fields activated are generally smooth over space and decaying activation can be seen multiple locations away. To evaluate the effectiveness of these echo view fields, we calculated the activity-weighted centroid at each point on the path, giving us an average point of each field to compare with the actual position of the sonar. The distance between the activity-weighted centroid and the actual position was used to calculate a mean error. Across 117 steps along three different paths, the original network's average error was 28.6 inches (72.6 cm), the widened network's average error was 18.6 inches (47.2 cm), and the widened network with leaky integration's average error was 16.3 inches (41.4 cm). This system successfully recognized locations that are not contained in the training set; the network can generalize and recognize many nearby views. When this fails, leaky integration allows past information to maintain a stable sense of place for the system.

Context/Previous Studies

These results complement previous studies that have used sonar to aid in place recognition. A large inspiration for our project was BatSLAM (Steckel and Peremans, 2013), a biomimetic sonar system that used odometry and sonar to map an area of their laboratory. Because odometry is quite inaccurate due to wheel slippage and other errors, such as compounding inaccuracies in estimating direction and position, sonar was used to provide error correction. Their system first drew paths of motion based solely on odometry. When the sonar-based recognition system recognized the current location from a prior visit, it updated the odometry system to match its memory and propagated the correction to earlier time steps for consistency. This was sufficient to correctly create a map of the area with little error. While this approach showed that sonar was able to aid place recognition, it did not do so in a biologically-plausible manner. Over the robot's path, 6,000 sonar measurements were taken, and 3,300 different places were established. While this system provides a

method to maintain an estimate of the robot's position, it does not seem to reflect what little is known about how biological memories of the environment. Memorizing 3,300 different places all within one environment is computationally and memory-intensive; it is not a biologically-plausible algorithm. While our study attempted to show that odometry is not needed for view recognition, incorporating odometric information can provide a strong framework for unsupervised mapping. For example, a new "place" can be created when a system, using odometry, estimates it is a certain distance from any other "place."

Another recent paper explored the idea of recognizing place with sonar in three different locations (Vanderelst et al., 2016). Using a very precise sonar sensor they measured the echo response at positions over a wide range of angles and along a linear, 10 m long path. They collected an enormous amount of data (over 20,000 echo traces) and evaluated whether the echoes varied smoothly over angle and distance as well as whether unique locations could be classified. Most of the data came from angular variation; large translational steps contrast the high angular resolution. They also found places that were difficult to distinguish between, mainly in open areas with few objects to sense, but concluded that sonar is enough to recognize most locations. When they were comparing different positions along a linear path, they compared the same precise angle (0.1 degree error) from the different positions. This is much more precise than an animal can hope to achieve, in reality both angle and position will be changing at the same time. We have shown in this study how sensitive an echo signature can be to changes in angle; we expect place recognition to be tolerant to moderate changes in the sensing direction. Our study can complement this one by providing a wider, two-dimensional range of positions for comparison as well as removing the need for very precise angular measurements.

In our study, all views were looking in the same direction. A network that could respond to views in different directions but at the same general location would be a step toward modeling a more general place cell. This could be modeled using an additional layer of a neural network. We have shown that different views can be separately recognized in a single layer network, another layer would be able to select which views correspond to the same place. This could be as simple as an "or" function that allows a view from any direction to activate the place cell.

Single Frequency vs. Broadband

One important aspect of the sonar currently used in our system that is not biologically-realistic is the use of a single frequency (40 kHz). Bats use a broadband sonar pulse that provides much richer echo signatures with spectral content that likely contributes to object characterization that is not possible with our sonar (Mogdans and Schnitzler, 1990). Even with this limitation, this study shows that place field generation is still possible knowing only object range (inferred by the peak sound pressure on the three transducer channels) and echo magnitude. Different objects with multiple close surfaces can also produce echoes with different durations. With a broadband sonar sensor, it may be possible to significantly improve the size and reliability of the place fields.

CONCLUSION

We have presented a robotic sonar system that uses ultrasonic transducers to mimic bat echolocation and have demonstrated two different networks that can recognize sonar views over a range of angles and offsets ("echo view fields"), with one network showing that this can be done in a biologically plausible manner. This view-based approach that does not require the identification of specific objects or explicit use of landmarks. The echo view cells produce "reasonable" responses outside of places where training data was collected and has the potential to be integrated into a larger system to model bat hippocampal place cells and spatial mapping.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

TH: created the hardware and software template for the sonar system. JI: created the sonar mount and modified the software for this project, and performed the data collection and analysis with advice and guidance from TH. Both authors discussed the results and contributed to the final manuscript.

FUNDING

This work was support by grants from the U.S. Office of Naval Research (N000141210339), the U.S. Air Force Office of Scientific Research (FA9550-14-1-0398, Center of Excellence), and the U.S. National Science Foundation (SMA1540916, Neuromorphic Engineering Workshop, and DGE-1632976, COMBINE program).

ACKNOWLEDGMENTS

We thanked Connor O'Ryan for his contributions to initial SKIM software testing and helped collecting some data, Chenxi Wen for technical advice and advice on early manuscripts, and Terrence Stewart for advice on machine learning implementation and software.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2020.567991/full#supplementary-material

Supplementary Video | Similar to **Figure 13**, these videos show echo view fields responses along the path. The red dot represents the position of the sonar. The activations of the echo view cells are shown in their corresponding location, seen as colored squares on the plots. Video 1 shows the single layer network, which was trained to have only one view cell active at a time. Video 2 shows the widened single layer network, which allows for more cells to be active at once, improving accuracy in between trained views. Video 3 shows the leaky integration applied to the widened network, which maintains a more stable activation due to its use of the past activations in the path.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro Abadi, C., et al. (2015). TensorFlow: large-scale machine learning on heterogeneous systems, 2015. *arXiv*. arXiv:1603.04467. Software Available online at: https://www.tensorflow.org/ (accessed October 18, 2020).

Bachelder, I. A., and Waxman, A. M. (2011). Mobile robot visual mapping and localization: a view-based neurocomputational architecture that emulates hippocampal place learning. *Neural Netw.* 7, 1083–1099. doi: 10.1016/S0893-6080(05)80160-1

Bisong, E. (2019). *"Google Colaboratory." Building Machine Learning and Deep Learning Models on Google Cloud Platform.* Berkeley, CA: Apress, 59–64. doi: 10.1007/978-1-4842-4470-8_7

Eliakim, I., Cohen, Z., Kosa, G., and Yovel, Y. (2018). A fully autonomous terrestrial bat-like acoustic robot. *PLoS Comput. Biol.* 14:e1006406. doi: 10.1371/journal.pcbi.1006406

Geva-Sagiv, M., Las, L., Yovel, Y., and Ulanovsky, N. (2015). Spatial cognition in bats and rats: from sensory acquisition to multiscale maps and navigation. *Nat. Rev. Neurosci.* 16, 94–108. doi: 10.1038/nrn3888

Huang, G.-B., Dian, H. W., and Yuan, L. (2011). Extreme learning machines: a survey. *Int. J. Mach. Learn. Cyber.* 2, 107–122. doi: 10.1007/s13042-011-0019-y

Jiang, T., Zang, W., Chao, C., and Shi, J. (2010). An energy consumption optimized clustering algorithm for radar sensor networks based on an ant colony algorithm. *EURASIP J. Wireless Commun. Netw.* 2010.1:627253. doi: 10.1155/2010/627253

Jung, M. W., Wiener, S. I., and McNaughton, B. L. (1994). Comparison of spatial firing characteristics of units in dorsal and ventral hippocampus of the rat. *J. Neurosci.*14, 7347–7356. doi: 10.1523/JNEUROSCI.14-12-07347.1994

Kingma, D. P., and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv*. arXiv preprint:1412.6980.

Koul, S., and Horiuchi, T. K. (2019). Waypoint path planning with synaptic-dependent spike latency. *IEEE Trans. Circuits Syst. I Regul. Pap.* 66, 1544–1557. doi: 10.1109/TCSI.2018.2882818

Mogdans, J., and Schnitzler, H. (1990). Range resolution and the possible use of spectral information in the echolocating bat, *E. ptesicusfuscus*. *J. Acous. Soc. Am.* 88, 754–757. doi: 10.1121/1.399724

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2017). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Trans. Biomed. Circuits Syst.* 12, 106–122. doi: 10.1109/TBCAS.2017.2759700

O'Keefe, J. (1976). Place units in the hippocampus of the freely moving rat. *Exp. Neurol.* 51, 78–109. doi: 10.1016/0014-4886(76)90055-8

Ollington, R., and Vamplew, P. (2004). "Learning place cells from sonar data," in AISAT2004: International Conference on Artificial Intelligence in Science and Technology (Hobart).

Steckel, J., and Peremans, H. (2013). BatSLAM: simultaneous localization and mapping using biomimetic sonar. *PLoS ONE* 8:0054076. doi: 10.1371/journal.pone.0054076

Strösslin, T., Sheynikhovich, D., Chavarriaga, R., and Gerstner, W. (2005). Robust self-localisation and navigation based on hippocampal place cellsral networks. *Neural Netw.* 18, 1125–1140. doi: 10.1016/j.neunet.2005.08.012

Tapson, J., and van Schaik, A. (2013). Learning the pseudoinverse solution to network weights. *Neural Netw.* 45, 94–100. doi: 10.1016/j.neunet.2013.02.008

Tapson, J. C., Cohen, G. K., Afshar, S., Stiefel, K. M., Buskila, Y., Wang, R. M., et al. (2013). Synthesis of neural networks for spatio-temporal spike pattern recognition and processing. *Front. Neurosci.* 7:153. doi: 10.3389/fnins.2013.00153

Ulanovsky, N., and Moss, C. F. (2007). Hippocampal cellular and network activity in freely moving echolocating bats. *Nat. Neurosci.* 10, 224–233. doi: 10.1038/nn1829

van Rossum, G. (1995). *Python Tutorial*, Technical Report CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI).

Vanderelst, D., Steckel, J., Boen, A., Peremans, H., and Holderied, M. W. (2016). Place recognition using batlike sonar. *Elife* 5:e14188. doi: 10.7554/eLife.14188.018

Wohlgemuth, M. J., Luo, J., and Moss, C. F. (2016). Three-dimensional auditory localization in the echolocating bat. *Curr. Opin. Neurobiol.* 41, 78–86. doi: 10.1016/j.conb.2016.08.002

Yartsev, M. M., and Ulanovsky, N. (2013). Representation of three-dimensional space in the hippocampus of flying bats. *Science* 340, 367–372. doi: 10.1126/science.1235338

Yartsev, M. M., Witter, M. P., and Ulanovsky, N. (2011). Grid cells without theta oscillations in the entorhinal cortex of bats. *Nature* 479, 103–107. doi: 10.1038/nature10583

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 13 November 2020
doi: 10.3389/fnbot.2020.568283

A Spike-Based Neuromorphic Architecture of Stereo Vision

Nicoletta Risi, Alessandro Aimar, Elisa Donati, Sergio Solinas and Giacomo Indiveri*

Institute of Neuroinformatics, University of Zurich, Eidgenössische Technische Hochschule Zurich, Zurich, Switzerland

The problem of finding stereo correspondences in binocular vision is solved effortlessly in nature and yet it is still a critical bottleneck for artificial machine vision systems. As temporal information is a crucial feature in this process, the advent of event-based vision sensors and dedicated event-based processors promises to offer an effective approach to solving the stereo matching problem. Indeed, event-based neuromorphic hardware provides an optimal substrate for fast, asynchronous computation, that can make explicit use of precise temporal coincidences. However, although several biologically-inspired solutions have already been proposed, the performance benefits of combining event-based sensing with asynchronous and parallel computation are yet to be explored. Here we present a hardware spike-based stereo-vision system that leverages the advantages of brain-inspired neuromorphic computing by interfacing two event-based vision sensors to an event-based mixed-signal analog/digital neuromorphic processor. We describe a prototype interface designed to enable the emulation of a stereo-vision system on neuromorphic hardware and we quantify the stereo matching performance with two datasets. Our results provide a path toward the realization of low-latency, end-to-end event-based, neuromorphic architectures for stereo vision.

Keywords: neuromorphic, event-based processing, event-based sensing, stereo vision, asynchronous computation

OPEN ACCESS

Edited by:
Christian Tetzlaff,
University of Göttingen, Germany

Reviewed by:
Jorg Conradt,
Royal Institute of Technology, Sweden
Akos Ferenc Kungl,
Heidelberg University, Germany

****Correspondence:***
Nicoletta Risi
nicoletta.risi@uzh.ch

Received: *31 May 2020*
Accepted: *09 October 2020*
Published: *13 November 2020*

Citation:
Risi N, Aimar A, Donati E, Solinas S and Indiveri G (2020) A Spike-Based Neuromorphic Architecture of Stereo Vision. Front. Neurorobot. 14:568283. doi: 10.3389/fnbot.2020.568283

1. INTRODUCTION

Biological and artificial binocular visual systems rely on stereo-vision processes to merge the visual information streams. This implies solving the stereo-matching problem, i.e., finding correspondent points in two slightly shifted views (Cumming and Parker, 1997). Typical applications in robotics that can benefit from stereo vision include navigation in unknown environments, object manipulation, and grasping. However, current machine-vision approaches still lag behind their biological counterpart mainly in terms of bandwidth and power consumption (Tippetts et al., 2016; Steffen et al., 2019). Classical methods are based on frame-based vision sensors. The main challenges of frame-based algorithms are spatial redundancy and temporal information loss due to the intrinsic nature of fixed-rate processing. This affects latency, throughput, and power consumption, making frame-based approaches difficult to integrate into mobile platforms.

Biological systems, on the other hand, seem to efficiently solve the stereo-matching problem by using space-variant and asynchronous space-time sampling (Steffen et al., 2019). Space-variant resolution refers to a non-uniform distribution of retinal photoreceptors, with higher density in the center (fovea) and a decreasing density toward the periphery. Asynchronous instead refers to event-driven, self-timed sensing and processing. Therefore, a massively parallel, asynchronous,

event-based chain, from sensing to processing, seems to be a promising solution for more robust and efficient architectures of stereo vision.

In this context, neuromorphic hardware has proven to be an effective substrate (Chicca et al., 2014; Indiveri et al., 2015). To date, the emerging field of event-based stereo vision has shown successful approaches that interface Spiking Neural Networks (SNNs) with neuromorphic event-based sensors, also referred to as "event cameras," in order to build real-time event-based visual processing systems (Mahowald, 1994a; Osswald et al., 2017). Inspired by the retinal ganglion cells, the neuromorphic vision sensors broadcast information, independently for all the pixels, only in response to significant changes in illumination, which results in a low-power, low-latency, event-driven, and sparse input stream (Lichtsteiner et al., 2008; Posch et al., 2010; Berner et al., 2013). Spiking neurons, in turn, can process signals using temporal information, and therefore, can take full advantage of an event-based input stream to solve the stereo-matching problem. However, although several biologically-inspired implementations of stereo vision (Mahowald, 1994b; Piatkowska et al., 2013, 2017; Dikov et al., 2017; Osswald et al., 2017; Kaiser et al., 2018) have extensively been explored, only a few solutions fully exploit the advantages of parallel computation, with an end-to-end neuromorphic architecture that can replace traditional Von Neumann architectures. In Dikov et al. (2017), the first scalable architecture of the Marr and Poggio cooperative network (Marr and Poggio, 1976, 1977, 1979) is implemented on the SpiNNaker platform (Furber et al., 2014). Despite the short latency (2 ms) of the network and the portable design, the reported power consumption of the neuromorphic implementation (90 W for a 3-board SpiNNaker machine) makes it difficult to integrate in mobile or autonomous applications. More recently, Andreopoulos et al. (2018) proposed the first fully end-to-end stereo pipeline, implemented on multiple TrueNorth processors (Sawada et al., 2016). The architecture achieves a 200× improvement, compared to Dikov et al. (2017), in terms of power per pixel per disparity map (0.058 mW/Pixel). Both solutions, however, emulate the cooperative stereo network on digital hardware. Inspired by biological neurons, analog neuromorphic circuits, by contrast, can potentially lead to more promising solutions for low-power, yet noisy, computation.

Following up on the work from Osswald et al. (2017), we present an end-to-end neuromorphic architecture of cooperative stereo vision implemented on mixed analog/digital neuromorphic hardware. Compared to the previous work, here we replaced the mixed-signal Very Large Scale Integration (VLSI) ROLLS chip (Qiao et al., 2015) with a scalable multi-core design (Moradi et al., 2018). Moreover, the proposed solution shifts the event-based computation directly on chip and provides a more robust, biologically-inspired coincidence detection mechanism. In the next section, we describe the digital interface between the sensing and the processing stage. Then, we present the neuromorphic implementation of the spiking network and we quantify the stereo matching performance with a synthetic dataset and an event camera dataset.

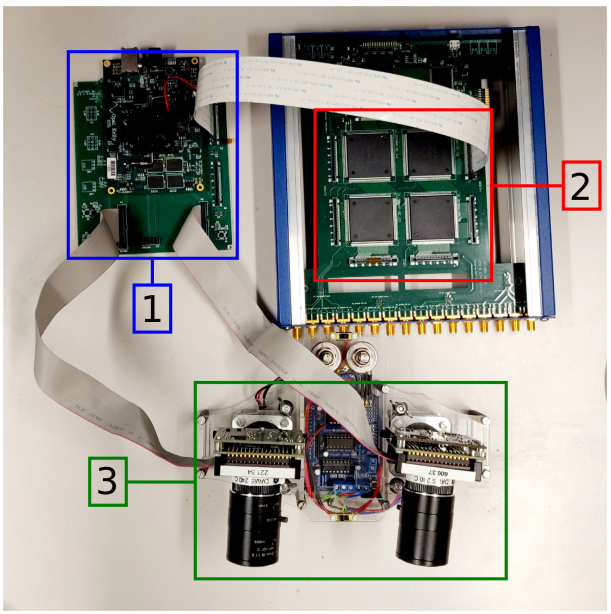


FIGURE 1 | The neuromorphic stereo-vision setup: OpalKelly XEM7360 [1], DYNAP [2], Stereo DAVIS240C [3].

2. METHODS

The stereo-vision architecture introduced here combines two event-based sensors, the Dynamic and Active Pixel Vision Sensor (DAVIS) (Berner et al., 2013), and three VLSI multi-core analog/digital Dynamic Neuromorphic Asynchronous Processors (DYNAPs) (Moradi et al., 2018) integrated in a 4-chip board. As a prototype, we designed the interface between sensing and processing on a dedicated Field Programmable Gate Array (FPGA) device (Xilinx Kintex-7 FPGA on the OpalKelly XEM7360).

2.1. Event-Based Sensing

As opposed to classical frame-based cameras, event-based sensor encodes information with lower latency and redundancy (Gallego et al., 2019). Inspired by the biological photoreceptors, the neuromorphic pixels operate independently and send out asynchronous events in response to significant changes in illumination using an event-based data protocol Address Event Representation (AER) (Deiss et al., 1998). The polarity of those events encodes increases (ON events) or decreases (OFF events) in illumination. Overall, this results in fast data acquisition with low latency and high temporal resolution. Compared to the original DVS (Lichtsteiner et al., 2008), the DAVIS sensor features a higher spatial resolution (240 × 180) and adds an APS (Active Pixel Sensor) readout.

In the proposed architecture, the two DAVIS sensors are mounted on a stereo-setup (see **Figure 1**) and are separated by a baseline distance of about 6 cm, which is similar to the pupillary distance of humans. Events are sent separately from both retinas to an FPGA using the AER protocol.

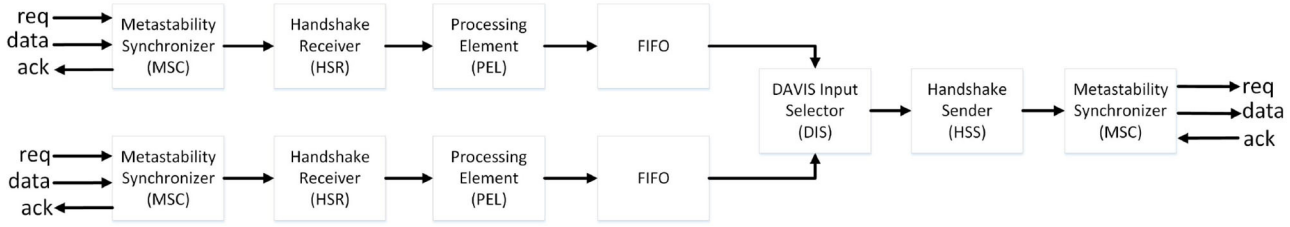


FIGURE 2 | Overview of the event-based digital interface.

2.2. Sensors-Processor FPGA Interface

Figure 2 shows the main modules of the event-based digital interface. The communication to/from the FPGA is based on a 4-phase handshake protocol, handled by the Handshake Receiver (HSR). Since the 4-phase handshake interfaces two different clock domains, metastable states of the input events could occur. This is handled by the Metastability Synchronizer (MSC) module, which uses a chain of two Flip-Flops to prevent metastability. A pre-processing element (PEL) reduces the input resolution to a 16×16 array to redirect the AER events to the destination core on the neuromorphic processor. The pre-processed events are thus forwarded to a small FIFO with eight entries, in charge of absorbing the pipeline stall due to the successive multiplexing stage. The DAVIS Input Selector (DIS) module multiplexes the data using a round-robin scheme and forwards them to the Handshake Sender (HSS), which handles the output handshake with the neuromorphic processor.

2.3. Event-Based Processing

The architecture computational substrate is a multi-core asynchronous mixed-signal neuromorphic processor fabricated using standard 0.18 µm 1P6M CMOS technology, the DYNAP (Moradi et al., 2018). Each core comprises 256 adaptive exponential integrate-and-fire (AEI&F) silicon neurons that emulate the biophysics of their biological counterpart, and four different dedicated analog circuits that mimic fast and slow excitatory/inhibitory synapse types (Brette and Gerstner, 2005). Each neuron has a Content Addressable Memory (CAM) block, containing 64 programmable entries allowing to customize the on-chip connectivity. A fully asynchronous inter-core and inter-chip routing architecture allows flexible connectivity with microsecond precision under heavy systems loads. Digital peripheral asynchronous input/output logic circuits are used to receive and transmit spikes via an AER communication protocol, analogous to the one used for the event-based input stream. As a result, the proposed implementation leads to a prototype for a fully asynchronous pipeline of event-based stereo vision.

2.4. The Spiking Neural Network Model

The SNN implemented on the DYNAP is adapted from the structure presented in Osswald et al. (2017). It consists of three neuronal populations: the retina, the coincidence detectors, and the disparity detectors (see **Figure 3**). Each coincidence and disparity neuron is assigned a triplet of coordinates, a horizontal cyclopean position ($x = x_R + x_L$), a vertical cyclopean position (y), and a disparity value ($d = x_R - x_L$), which determines the neuron representation of a location in the 3D space.

Each coincidence neuron receives excitatory inputs from a pair of retina cells tuned to its same spatial location (x_R or x_L), thereby encoding temporal coincidences among pairs of inter-ocular events. However, the temporal information is crucial but not sufficient to correctly solve the stereo ambiguity, which arises from matching features from different stimuli. For instance, two stimuli moving synchronously on a plane yield four clusters of activation in the coincidence detectors population: two correct matches along the direction of constant disparity, here referred to as True Targets (TT) and two wrong matches along the direction of constant cyclopean position, here referred to as False Targets (FT), which correspond to the erroneous perception of two stimuli moving in depth.

This ambiguity is reduced in the disparity population by means of two mechanisms of inhibition: recurrent inhibition (Type I) across disparity neurons tuned to the same line of sight (i.e., $x = x_L$ *or* $x = x_R$) and feed-forward inhibition (Type II) from coincidence neurons tuned to the same cyclopean position. Moreover, disparity neurons receive feed-forward lateral excitation from coincidence neurons tuned to the same disparity. This excitatory-inhibitory balance allows integrating the stimulus spatiotemporal features over time, thereby implementing the matching constraints of cooperative algorithms (Marr and Poggio, 1976; Mahowald, 1994b; Osswald et al., 2017). As a result, the SNN model can solve the stereo matching problem, with only TT represented in the disparity population.

2.5. Neuromorphic Hardware Implementation

The entire pipeline of visual information processing was designed to be a scalable neuromorphic architecture. In our proof-of-concept mixed-signal implementation of stereo vision, both coincidence and disparity detectors are implemented using silicon neurons. All neurons in the architecture are emulated by parallel physical circuits in real-time on the neuromorphic processor. In order to optimize the trade-off between the retina field of view and the computational resources on hardware, the input pixels from the event cameras are downscaled to two 2D arrays of 16×16 neurons on FPGA which, in turn, project to a 3D array of coincidence detectors. Therefore, the array has a

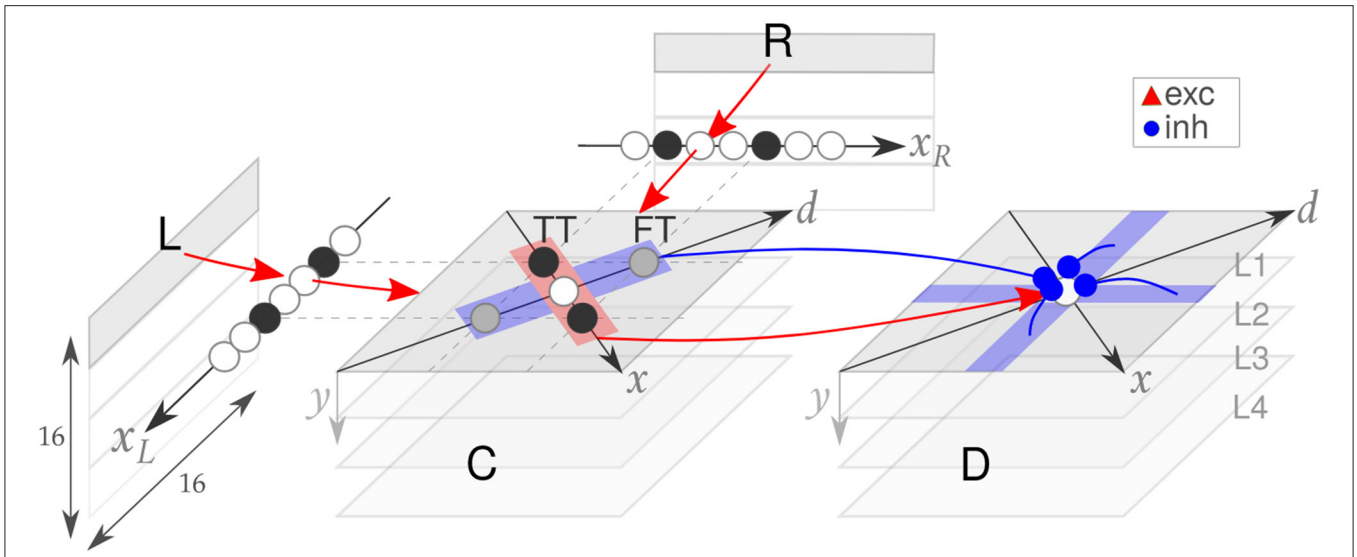


FIGURE 3 | The spiking neural network model. The input space of the retina (R, L) is downscaled and processed by four populations of coincidence (C) and disparity (D) neurons (first one highlighted in light gray). Excitatory (red) and inhibitory (blue) connections are shown. Adapted from Osswald et al. (2017).

width of 16 neurons, both in the x_R and x_L dimensions. The y dimension, instead, is further downscaled to four levels, hereafter referred to as network "layers" L1–4 (**Figure 3**). The same structure is implemented for the 3D array of disparity neurons. In total, the architecture comprises $N_n = 3,072$ silicon neurons and $N_s = 62,562$ silicon synapses (see **Supplementary Data 1.2** for the estimated power consumption of the network).

2.5.1. Coincidence Detection

Since coincidence detection is a key component of our model, we carefully emulated and further optimized the low-power mechanism exploited by biological brains. Specifically, temporal coincidences are detected by combining the mechanism of supra-linear, dendritic summation of synaptic events with slow and fast synaptic time constants. As in biological brains, AMPA synaptic currents can boost the effect of slow NMDA synapses when both synaptic inputs are close in time (González, 2011). Coincidence detectors are emulated on the chip exploiting the non-linear properties of the dedicated analog synapse circuit block, which mimics the biological NMDA voltage-gating dynamics. Each coincidence detector is connected to one of the corresponding input retina cells via the slow (NMDA-like) synapse and to the other one via the fast (AMPA-like) synapse circuit block. Only if both synapses are stimulated in rapid succession the coincidence detector neuron fires. A demonstration of coincidence detection emulated on-chip is shown in **Figure 4** (see **Supplementary Data 1.1** for a full characterization of the proposed coincidence detection building block). To reduce the effect of high-frequency homolateral excitation (Dikov et al., 2017), we included one inhibitory connection from each input neuron to the coincidence detectors population. By controlling the ratio between excitatory/inhibitory synaptic time constants, this helps to suppress incoming monocular events with a high input rate, which would otherwise boost the activation of coincidence detectors, leading to the erroneous perception of inter-ocular coincidences.

2.5.2. Disparity Detection

Lateral feed-forward inhibition was implemented with a separate population of coincidence neurons receiving excitatory connections from the coincidence detectors (**Supplementary Figure 3**). As a result, the effect of the lateral inhibition is delayed with respect to the feed-forward input from the population of excitatory coincidence detectors. This allows to boost the activity of neurons receiving excitatory inputs due to temporally correlated inter-ocular events and therefore helps to suppress false targets in the disparity population.

2.5.3. Network Calibration

As shown in Osswald et al. (2017), neurons in the emulated SNN model of cooperative stereo vision compute an approximation of the local covariance of the spatiotemporal visual information. As a result, neuronal and synaptic time constants are key parameters in the proposed architecture, and they were configured as follows. First, we measured the distribution of both monocular and interocular inter-spike-intervals of the input events. Then, the time constants of coincidence detectors were set according to the constraints in (**Supplementary Data equation S2**). Finally, the neuronal time constants of disparity detectors were set significantly larger than the time constants of coincidence detectors, i.e., within the timescale of hundreds of milliseconds.

2.6. Experiments

Prior to a full-scale implementation of the prototype architecture, we assessed the stereo matching performance by comparing the network output to an event-based ground truth. We included in our interface design another datapath that uses the OpalKelly USB3.0 to allow high-speed data transfer from the PC. This

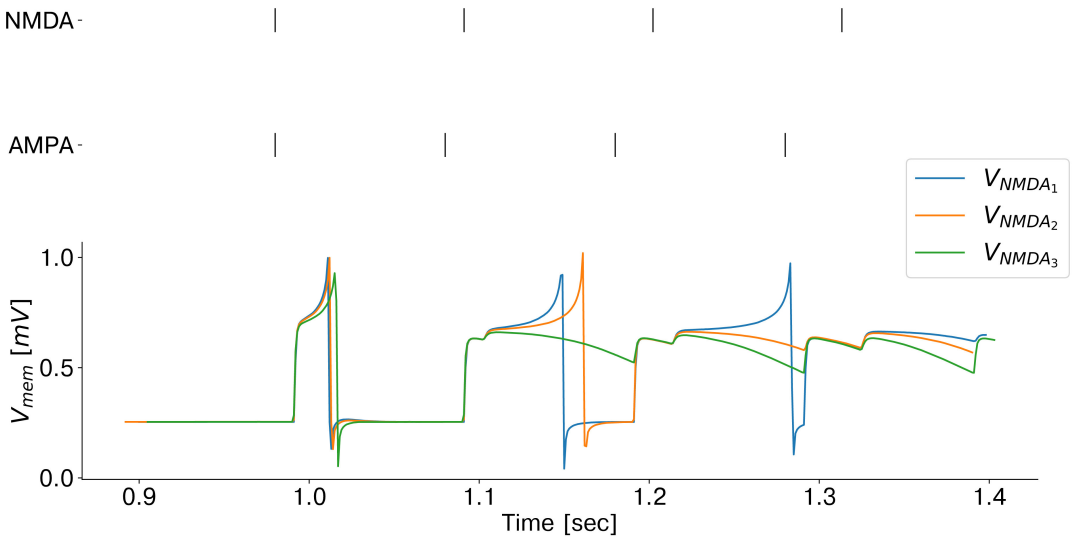


FIGURE 4 | Emulation of coincidence detection: recorded membrane potential of a coincidence detector as a function of the NMDA voltage-gating threshold (V_{NMDA}). As the threshold decreases, the silicon neuron responds to larger inter stimulus interval delays and therefore, the coincidence detector sensitivity increases.

allowed us to validate the network performance in two scenarios. First, we used a synthetic dataset to test the effectiveness of lateral inhibition with temporally correlated input events. Next, we tested the network performance with real events collected with the event cameras.

2.6.1. Stereo Matching With Synthetic Inputs

As a first step, we generated a synthetic dataset to mimic the output of two neuromorphic retinae recording the scenario of motion on a plane, and specifically two stimuli (dark edges) moving in opposite directions on different depth planes (**Figure 5B**). The spiking network model in Osswald et al. (2017) is designed to have individual coincidence detectors for each event polarity. However, since a full-scale implementation of the model is out of the scope of this work, we chose to focus our analysis on one event polarity. **Figure 5A** shows the reproduced activity in the input neurons, together with the expected output of the disparity population. The neural activity is depicted as a temporal image, with gray levels representing synchronous activation in time.

We define as "stimulus speed" the number of input neurons sequentially activated by the stimulus over time. Thus, we chose a speed of 20 input neurons/s for both stimuli, with each input neuron firing at 50 Hz when the stimulus moves to its corresponding location (**Supplementary Figure 7A**). Moreover, events were generated with vertical coordinates such that they would target only one out of four network layers.

Since the goal is to validate the effect of lateral inhibition, we explicitly constructed the input events with perfect temporal inter-ocular correlation. In this scenario, only if the network uses the lateral inhibition to integrate not only temporal but also spatial features of the stimuli, the ambiguity can be resolved.

2.6.2. Stereo Matching With Event Cameras Inputs

Real-time scenarios recorded with event cameras inevitably produce noisy events, mainly due to camera jitter and variable latency. Therefore, in order to validate the proposed approach for an end-to-end event-based architecture of stereo vision, it is essential to assess whether the network can still resolve the ambiguity of stereo correspondences with noisy inputs. To this end, we reproduced the scenario of motion on a plane simulated with synthetic data and recorded events from the event cameras. The experimental setup is illustrated in **Figure 6A**.

The software "Processing" (Reas and Fry, 2007) was used to simulate two dark edges moving on a white background at a constant speed on two different screens. The setup was calibrated using the MATLAB Stereo Camera Calibrator Toolbox with the grayscale images of the DAVIS240C. Upon estimating the camera extrinsics and intrinsics, one screen was placed around the camera vergence point and the second one between the vergence point and the stereo setup. In order to optimize the ratio between spatial resolution and the number of input neurons, a window of 96 × 96 pixels centered around the stimulus was applied to filter out information outside the region of interest, and the recorded events were further downscaled with a kernel of 6 × 6 pixels.

2.7. Stereo Matching Performance

2.7.1. Event-Based Ground Truth

In order to assess the stereo matching performance of the network, an event-based ground truth is required. While this is intrinsically available in the case of synthetic datasets, it is not as straightforward with a real dataset. For this scenario, we assumed as true matches the stereo correspondences detected with generalized time-based technique (Ieng et al., 2018) with spatial, temporal, and motion consistency used as matching

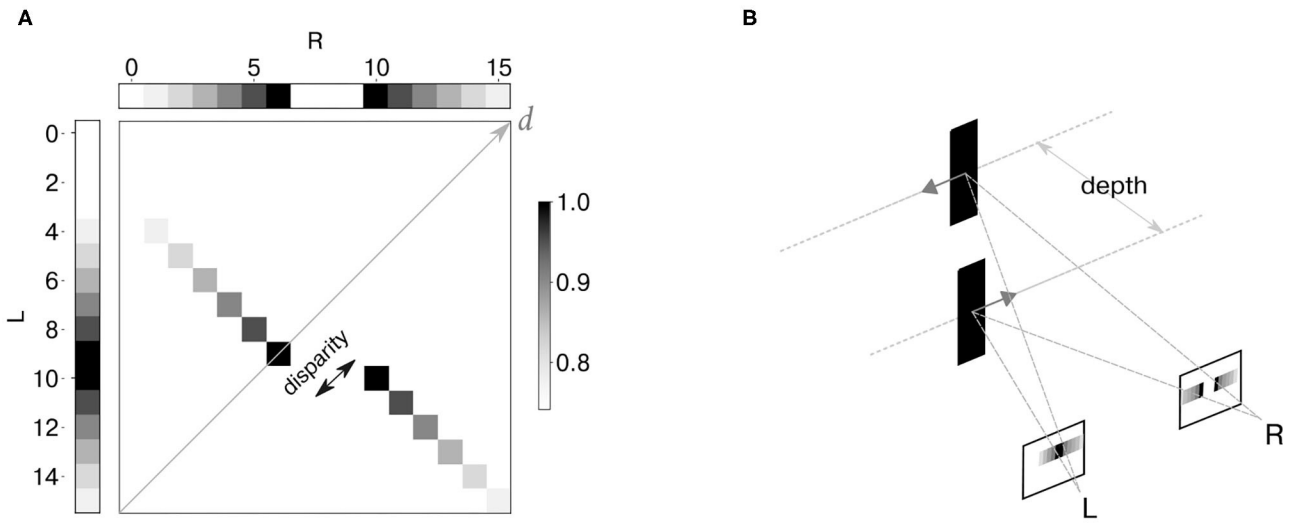


FIGURE 5 | Synthetic dataset: the neural activity is depicted as a temporal image, with gray levels representing synchronous activation in time **(A)**. Two spike trains were generated to simulate the activity of event cameras in response to two edges moving in opposite directions at constant depth levels **(B)**. Input binocular time series and expected activation over time of disparity neurons are shown as temporal images.

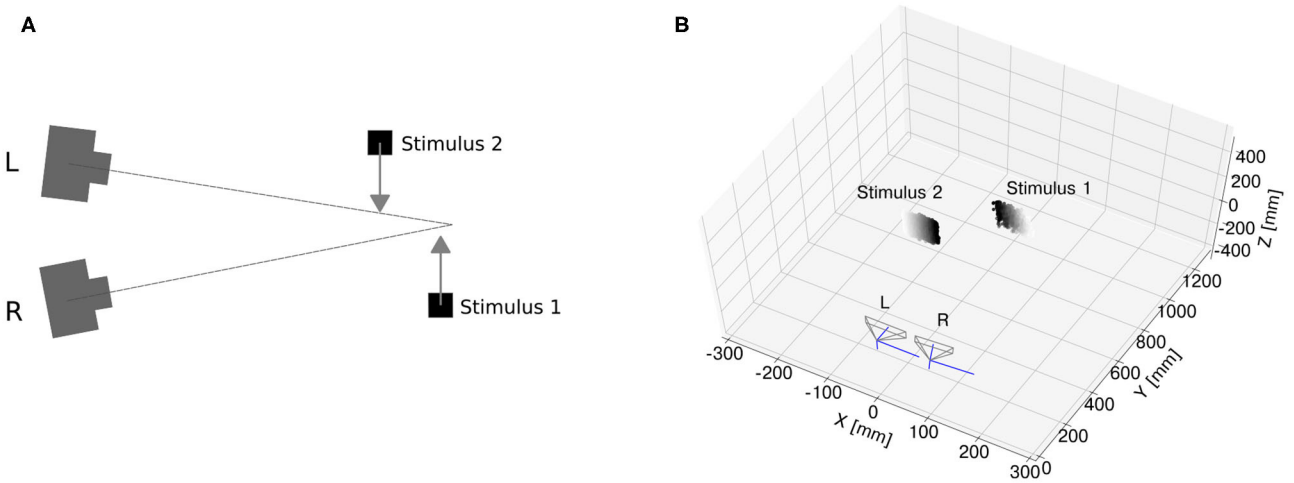


FIGURE 6 | Event camera dataset. Sketch of experimental setup **(A)** two monitors were used for the generation of two edges separated in depth and moving on a plane. After calibration, the monitor generating Stimulus 1 was placed closer to the region of the camera vergence point, while Stimulus 2 was placed closer to the stereo setup. Pointcloud reconstruction **(B)** with generalized time-based technique (time window ϵ = 2 ms, exponential decay kernel τ = 10 ms and a spatial kernel of 10 × 10 pixels).

constraints[1]. To increase the ground-truth accuracy, we fed the generalized time-based technique with one stimulus at a time so that there was no stereo ambiguity. Finally, detected stereo correspondences were labeled as true targets if yielding a correlation score larger than $c = 0.4$ (resulting pointcloud reconstruction shown in **Figure 6B**).

[1]As the DAVIS240C does not integrate the synchronous luminance information, the luminance consistency constraint could not be included in our analysis.

2.7.2. Accuracy

The stereo matching accuracy was measured with the following metrics proposed in Osswald et al. (2017).

1. Percentage of Correct Matches (PCM):

$$PCM_{C,D}(t_i) = \frac{TT_{C,D}(t_i)}{FT_{C,D(t_i)} + TT_{C,D}(t_i)} \quad (1)$$

with $TT_{C,D}(t_i)$, and $FT_{C,D}(t_i)$ being the normalized number of true targets and false targets recorded within a time window

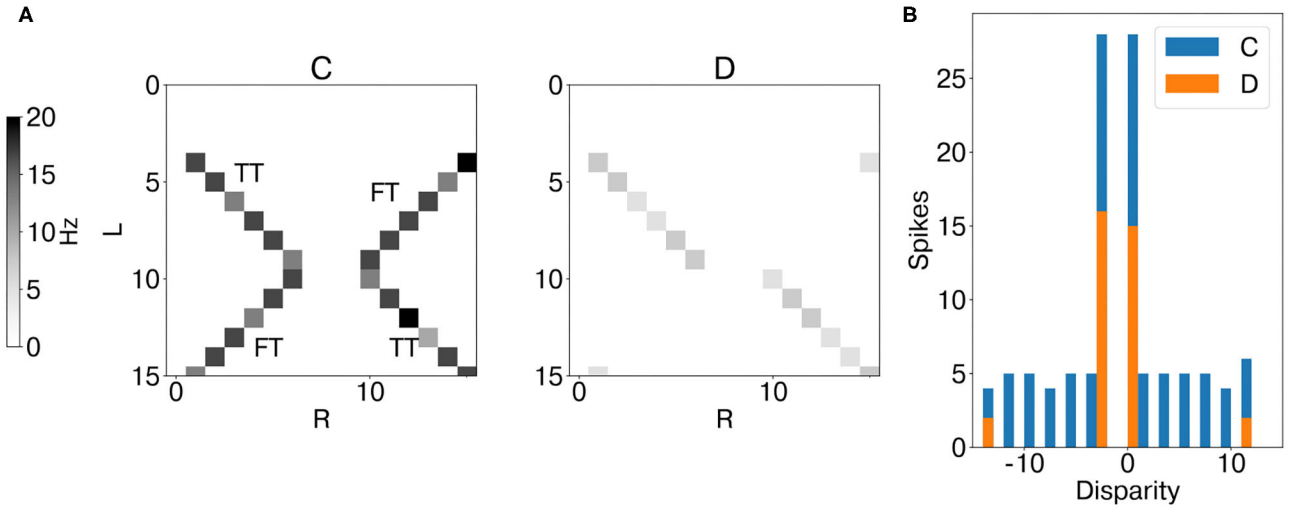


FIGURE 7 | Results of network emulation with the synthetic dataset. Mean firing rate of coincidence neurons (C, excitatory population) and disparity population (D) **(A)**. Histogram of encoded disparity values across the trial duration in both coincidence (blue) and disparity neurons (orange) **(B)**. As the activity clusters around the true disparity values (d = 0, d = −3), the disparity population successfully resolves the stereo ambiguity.

t_i, both for coincidence and disparity neurons. Spikes were labeled as true targets if the minimum euclidean distance in the 2D plane (x, d) between the recorded neuron id and the ground truth neuron ids was smaller than the threshold distance $D_{min} = 1$.

2. True Target Amplification (TTA) and False Target Amplification (FTA):

$$TTA = \frac{\sum_{t_i} TT_D(t_i)}{\sum_{t_i} TT_C(t_i)} \qquad FTA = \frac{\sum_{t_i} FT_D(t_i)}{\sum_{t_i} FT_C(t_i)} \tag{2}$$

which allow quantifying the disparity sensitivity (TTA) and the degree to which false targets are suppressed due to recurrent and lateral feed-forward inhibition (FTA).

3. RESULTS

3.1. Stereo Matching With Synthetic Inputs

Figure 7 shows the mean firing rate of coincidence (excitatory population) and disparity neurons during the whole trial. The coincidence detectors successfully detect the temporal matches, i.e., an action potential arises only when the input events from the retina cells are coincident in time. However, coincidence detectors still respond to false targets, i.e., coincident events arising from different stimuli. Indeed, in this scenario, binocular time series related to different stimuli are perfectly synchronized (**Supplementary Figure 8A**) and therefore not distinguishable from the true targets in the temporal domain (Mulansky and Kreuz, 2016). However, as they activate coincidence detectors along the dimension of constant cyclopean position, they also trigger the activation of the correspondent inhibitory coincidence detectors, leading to inhibition of disparity detectors tuned to the same cyclopean position (**Supplementary Figure 4**). This is not the case for true targets as binocular events due to the same stimulus target coincidence detectors along the dimension of constant disparity, which injects excitatory current into target disparity detectors tuned to the same disparity. As a result, disparity detectors integrate evidence of true disparities and effectively solve the stereo ambiguity.

This is well-depicted by the metrics of stereo matching performance. Compared to coincidence detectors, disparity neurons can successfully suppress false targets (FTA = 0.08), while still being responsive to true targets (TTA = 0.45). This leads to a PCM score of 0.88, compared to PCM = 0.57 for coincidence detectors.

As temporal information is the key feature for an event-based network, the stimulus speed is a crucial factor influencing the network performance. Indeed, as the number of input neurons sequentially activated by the stimulus decreases, the ratio TTA/FTA decreases, thereby affecting the stereo matching performance (**Figure 8**).

3.2. Stereo Matching With Event Cameras Inputs

Analogously to the analysis performed with synthetic data, we first measured the average instantaneous firing rate of coincidence and disparity neurons during one trial with data from the event cameras. Notably, binocular time series of non-correspondent stimuli are less correlated in real scenarios (**Supplementary Figure 8B**). Therefore, false and true targets become more separable from the temporal information already. This is why the activation of coincidence detectors responding to false targets is reduced compared to those responding to true targets (**Figure 9**). However, disparity detectors still achieve better performances in resolving the stereo ambiguity (**Figure 10**).

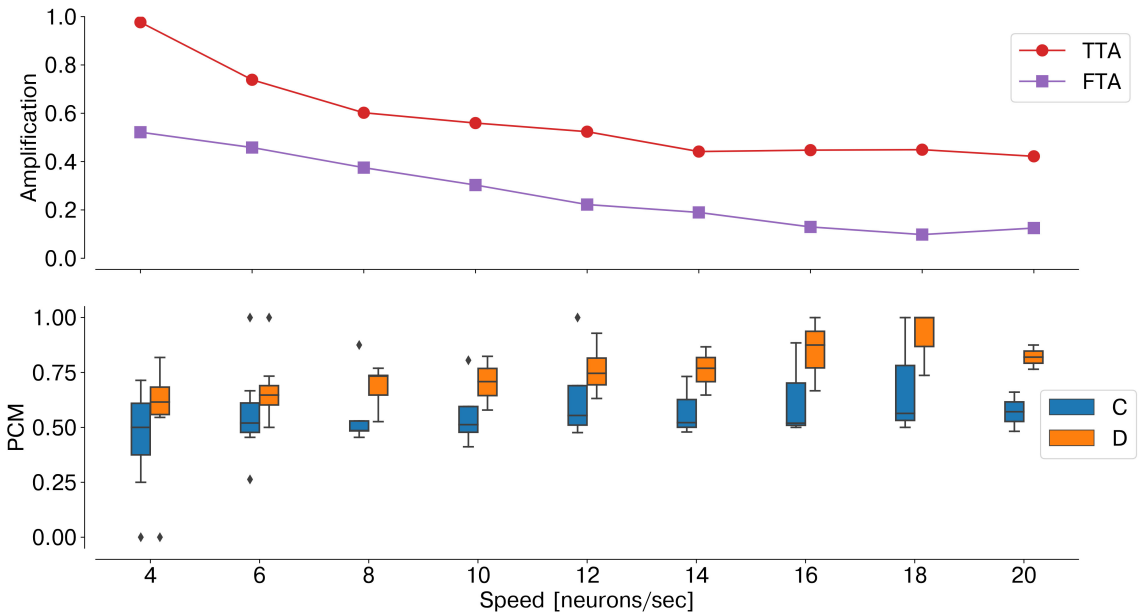


FIGURE 8 | Performance sensitivity to the stimulus speed: PCM (bottom graph, with median and interquartile range, measured across one trial over time windows $t_i = 300$ ms), TTA, and FTA (top graph). As the stimulus speed increases, the stereo matching performance increases (i.e., lower FTA).

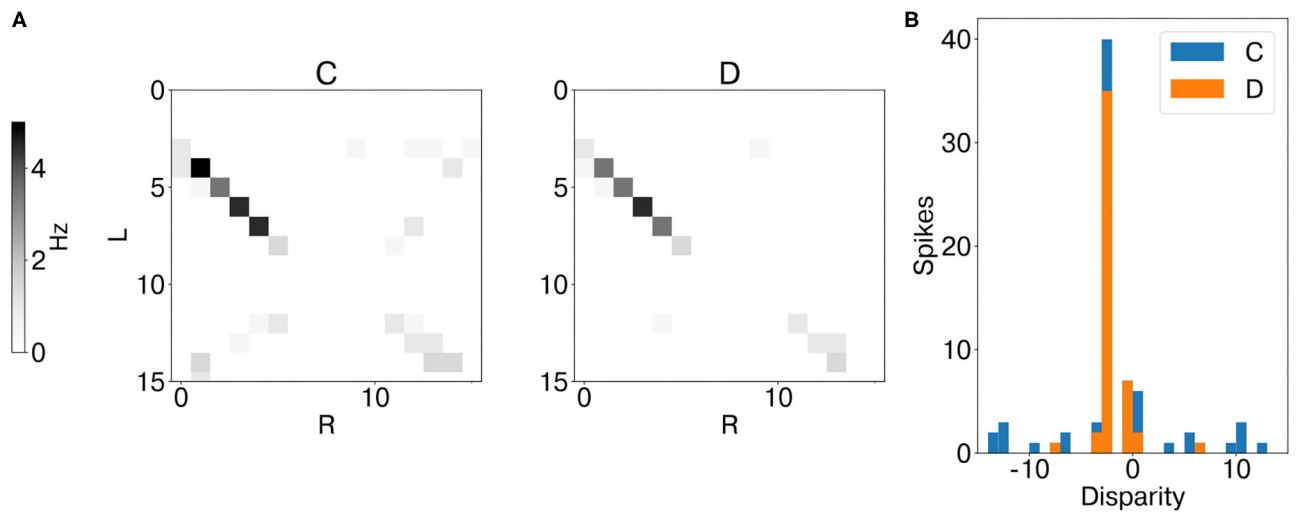


FIGURE 9 | Results of network emulation with event camera dataset from network layer L2. Mean firing rate of coincidence neurons (C, excitatory population) and disparity population (D) **(A)**. Histogram of encoded disparity value across the trial duration in both coincidence (blue) and disparity neurons (orange) **(B)**. As the activity clusters around the true disparity values, the disparity population successfully resolves the stereo ambiguity.

4. DISCUSSION

We have presented a prototype architecture for cooperative stereo vision implemented on a scalable neuromorphic architecture. Recovering the 3D structure of a scene is still computationally expensive for conventional computer vision approaches. Yet, biology shows several examples of stereo vision whereby space-variant and asynchronous space-time sampling are some of the key features involved. With parallel, sparse, and asynchronous computation, neuromorphic hardware promises to offer an optimal substrate for a low-latency implementation of 3D vision. However, only a few approaches developed so far fully exploit the advantages of analog asynchronous computation. Hereby we implemented a biologically-inspired, event-based network of stereo vision on a mixed analog-digital neuromorphic processor and we validated the stereo matching performances of the architecture.

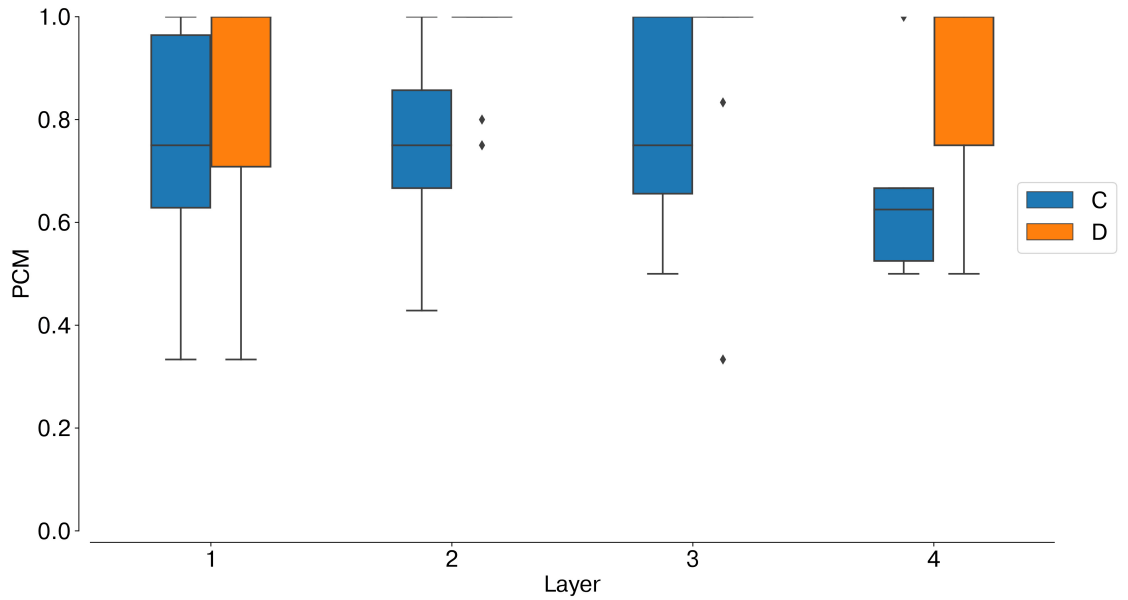


FIGURE 10 | Stereo matching accuracy: PCM, median and interquartile range. In all network layers (L1–4) the PCM of disparity neurons is larger than the PCM of coincidence neurons, showing that disparity detectors can still solve the stereo ambiguity with slower, and uncorrelated, real stimuli.

Our model is derived from the work of Osswald et al. (2017), which presents software simulations of the full-scale implementation. By solving the stereo-matching problem with leaky-integrate-and-fire neurons, the simulated spiking network proves an effective approach to fully exploiting the event-based visual sensors. However, the full potential of the model and its scalability can only be leveraged if the neurons operate in parallel. Here we validated the stereo-matching abilities of the network by implementing it on a massively parallel neuromorphic processor. Compared to the previous feasibility study based on the ROLLS chip (Osswald et al., 2017), the proposed solution shifts the coincidence detection mechanism, previously on FPGA, directly on analog silicon neurons. Exploiting the non-linear properties of a dedicated analog circuit, that mimics the biological NMDA voltage-gating dynamics, led to a robust coincidence detection mechanism that could ease the network sensitivity to device mismatch, which is a crucial feature of subthreshold mixed-signal neuromorphic processors. In this regard, we anticipate that quantifying the effect of device mismatch on coincidence detection will be a crucial step prior to a full-scale implementation of the network on-chip.

In order to validate the effectiveness of the neuromorphic substrate in solving the stereo correspondence problem, we assessed the network performances in two scenarios. First, with a synthetic dataset, we demonstrated the crucial role of the synaptic kernel of feed-forward lateral inhibition. To do so, we explicitly constructed the input binocular time series such that false targets would be temporally correlated and, therefore, only distinguishable from the true matches if disparity neurons integrated the stimulus spatiotemporal features. However, this is only possible when the temporal dynamics of the stimulus are comparable with the neuron synaptic time constants, as we showed in **Figure 8**. In other words, as the network exploits motion cues to solve the stereo matching problem, the network temporal sensitivity becomes intrinsically related to the network spatial resolution. Thus, the number of input neurons sequentially activated by the moving stimulus over time is a crucial factor: increasing the number of neurons sensitive to the input field of view would restore the network sensitivity to lower speed stimuli.

The second scenario with data from event cameras allowed us to test the network performance with noisy time series, whereby non-correspondent inter-ocular events are not perfectly correlated. Here the lateral inhibition fails due to lower speed stimuli (**Supplementary Figure 7B**). Yet the network can still achieve good stereo matching performances due to the recurrent inhibition, which triggers competition among disparity neurons tuned to the same line of sight. In this scenario, the feed-forward excitatory input from coincidence detectors responding to temporally correlated stimuli boosts the activation of disparity neurons responding to true targets, therefore successfully leading to false target suppression again.

Overall, both experiments validate our approach with stimulus motion yielding constant disparity. The future step is testing the network dynamics in the case of motion-in-depth, which naturally addresses the trade-off accuracy vs. speed. Indeed, coincidence detectors feature low-latency response to short inter-ocular time differences, thereby setting the network temporal resolution within the timescale of microseconds. Disparity detectors, by contrast, need to integrate the stimulus motion cues over time to resolve the stereo ambiguity, and therefore they require longer neuronal time constants (up to 100 ms). In fact, by receiving excitatory and inhibitory projections from coincidence and inhibitory

neurons, respectively, disparity neurons compare evidence of the current stimulus statistics against the integrated evidence of the stimulus spatiotemporal features. Measuring the network response in the case of motion in depth will allow investigating the effect of this excitatory/inhibitory balance on the stereo-matching performances. Moreover, since no synaptic plasticity is included in the architecture and given the event-based nature of the input stimulus, a prior assumption about the stimulus statistics is currently required to calibrate the network. Future implementations on the new generation of DYNAP chips will allow to incorporate mechanisms of short-term plasticity, thereby enabling an autonomous adaptive calibration procedure.

Although the architecture proposed is scalable by construction, implementing very large-scale systems based on such architecture, able to operate in real-time, requires adequate resources, and supporting neuromorphic processing hardware. The DYNAP processor used in this study comprises only 1,024 neurons, distributed among four cores of 256 neurons each. However, the routing scheme implemented on that device supports all-to-all connections of up to 16 by 16 chips providing already the ability to scale the system up to 256k neurons. This would, however, require very large printed circuit boards, or many boards interconnected among each other. The DYNAP chips proposed in Moradi et al. (2018) could be integrated into a system comprising a much higher number of cores [e.g., the IBM TrueNorth chip has 4,096 cores (Merolla et al., 2014), and the Intel Loihi chip has 128 cores (Davies et al., 2018)] without making any changes to the design. This would enable the construction of larger scale stereo-vision setups that would still be able to operate in real-time, given the parallel processing ability of the emulated neurons and synapses. We anticipate that designing an end-to-end asynchronous dedicated architecture of this type would allow to fully leverage the potential of sparse, event-based computation of SNN models of cooperative stereo-matching. An additional strategy that would enable the construction of large-scale stereo-vision setups would be to use more complex vision pre-processing stages, for example, implemented using convolutional networks and applying the same principles presented in this work to the features extracted by the convolutional network, rather than the raw pixel values. This would allow us to use a smaller feature space compared to the resolution of the vision sensor, and increase robustness to noise in the vision sensors. As discussed in Steffen et al. (2019), although there are many methods for event-based depth estimation, the lack of a comprehensive dataset or a standard testbed makes it difficult to compare them. Yet, some event-based datasets for stereo vision have been recently released (Andreopoulos et al., 2018; Zhu et al., 2018). Implementing the full-scale model on new generations of mixed analog/digital neuromorphic processors would allow comparing the architecture performances against already existing methods. In the long-term, the goal of the approach proposed is to enable on-chip estimation of depth on a per-event basis, with the highest resolution confined around the camera vergence point. Indeed, conventional approaches of event-based stereo vision constrain the search window for stereo matches along the epipolar lines, which results in the point of zero disparity to be shifted at infinity, and depth error increasing quadratically with depth. Instead, in this work, we took inspiration from the biological coarse and space-variant sampling and processed the raw events with large input search zones. In other words, here disparity detectors tuned to zero disparity respond to targets moving around the camera vergence point. While this naturally constrains the spatial (and therefore depth) resolution, it could set out an optimized solution with latency response and space-variant sampling. Combined with vergence control, this active perception strategy could lead to promising solutions for embedded neuromorphic architectures of stereo vision in humanoid robots (Gallego et al., 2019). Moreover, the need for compelling benchmarks that could show the advantages of spike-based computation in real-world scenarios is currently one of the major challenges for the neuromorphic research field (Davies, 2019). Our solution could show a valuable example of exploiting spike-timing to process real-time information in closed-loop systems, by emulating sparse, parallel computation of biological neurons in order to solve the stereo matching problem.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

NR did the research and wrote the manuscript. AA designed the digital interface. ED designed the interface between the neuromorphic chip and the OpalKelly board. ED, SS, and GI supervised the work. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported by the ERC Grant NeuroAgents (Grant. No. 724295).

ACKNOWLEDGMENTS

We authors would like to thank Marc Osswald for the fruitful discussions, Chenxi Wu for contributing to the AER interface design, and Dmitrii Zendrikov for helping with the process of calibrating the network parameters on the DYNAP. Additionally, we would like to thank the organizers of the Robust Artificial Intelligence for Neurorobotics Workshop, where preliminary results of this work were presented.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2020.568283/full#supplementary-material

REFERENCES

Andreopoulos, A., Kashyap, H. J., Nayak, T. K., Amir, A., and Flickner, M. D. (2018). "A low power, high throughput, fully event-based stereo system," in *The IEEE Conference on ComputerVision and Pattern Recognition (CVPR)* (Salt Lake City, UT), 7532–7542.

Berner, R., Brandli, C., Yang, M., Liu, S.-C., and Delbruck, T. (2013). "A 240 × 180 10 mW 12μs latency sparse-output vision sensor for mobile applications," in *2013 Symposium on VLSI Circuits* (Kyoto: IEEE), C186–C187.

Brette, R., and Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.* 94, 3637–3642. doi: 10.1152/jn.00686.2005

Chicca, E., Stefanini, F., Bartolozzi, C., and Indiveri, G. (2014). Neuromorphic electronic circuits for building autonomous cognitive systems. *Proc. IEEE* 102, 1367–1388. doi: 10.1109/JPROC.2014.2313954

Cumming, B. G., and Parker, A. J. (1997). Responses of primary visual cortical neurons to binocular disparity without depth perception. *Nature* 389, 280–283. doi: 10.1038/38487

Davies, M. (2019). Benchmarks for progress in neuromorphic computing. *Nat. Mach. Intell.* 1, 386–388. doi: 10.1038/s42256-019-0097-1

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Deiss, S., Douglas, R., and Whatley, A. (1998). "A pulse-coded communications infrastructure for neuromorphic systems," in *Pulsed Neural Networks, Chapter 6*, eds W. Maass and C. Bishop ((Cambridge, MA: MIT Press), 157–178.

Dikov, G., Firouzi, M., Röhrbein, F., Conradt, J., and Richter, C. (2017). "Spiking cooperative stereo-matching at 2 ms latency with neuromorphic hardware," in Conference on Biomimetic and Biohybrid Systems (Stanford, CA: Springer), 119–137.

Furber, S., Galluppi, F., Temple, S., and Plana, L. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2019). Event-based vision: a survey. *arXiv* 1904.08405. doi: 10.1109/TPAMI.2020.3008413

González, J. (2011). Distinguishing linear vs. non-linear integration in CA1 radial oblique dendrites: it's about time. *Front. Comput. Neurosci.* 5:44. doi: 10.3389/fncom.2011.00044

Ieng, S.-H., Carneiro, J., Osswald, M., and Benosman, R. (2018). Neuromorphic event-based generalized time-based stereovision. *Front. Neurosci.* 12:442. doi: 10.3389/fnins.2018.00442

Indiveri, G., Corradi, F., and Qiao, N. (2015). "Neuromorphic architectures for spiking deep neural networks," in *2015 IEEE International Electron Devices Meeting (IEDM)* (Washington, DC: IEEE), 4.2.1–4.2.14. doi: 10.1109/IEDM.2015.7409623

Kaiser, J., Weinland, J., Keller, P., Steffen, L., Tieck, J. C. V., Reichard, D., et al. (2018). "Microsaccades for neuromorphic stereo vision," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Rhodes). doi: 10.1007/978-3-030-01418-6_24

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128 × 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circuits* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Mahowald, M. (1994a). "Analog VLSI chip for stereocorrespondence," in *International Symposium on Circuits and Systems (ISCAS)* (London), Vol. 6, 347–350. doi: 10.1109/ISCAS.1994.409597

Mahowald, M. (1994b). *An Analog VLSI System for Stereoscopic Vision*. Boston, MA: Kluwer.

Marr, D., and Poggio, T. (1976). Cooperative computation of stereo disparity. *Science* 194, 283–287. doi: 10.1126/science.968482

Marr, D., and Poggio, T. (1977). *A Theory of Human Stereo Vision*. Technical report, Massachusetts Institute of Technology, Cambridge Artificial Intelligence Lab.

Marr, D., and Poggio, T. (1979). A computational theory of human stereo vision. *Proc. R. Soc. Lond. B Biol. Sci.* 204, 301–328. doi: 10.1098/rspb.1979.0029

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. (2018). A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs). *IEEE Trans. Biomed. Circuits Syst.* 12, 106–122. doi: 10.1109/TBCAS.2017.2759700

Mulansky, M., and Kreuz, T. (2016). Pyspike-a python library for analyzing spike train synchrony. *SoftwareX* 5, 183–189. doi: 10.1016/j.softx.2016.07.006

Osswald, M., Ieng, S.-H., Benosman, R., and Indiveri, G. (2017). A spiking neural network model of 3Dperception for event-based neuromorphic stereo vision systems. *Sci. Rep.* 7:40703. doi: 10.1038/srep44722

Piatkowska, E., Belbachir, A., and Gelautz, M. (2013). "Asynchronous stereo vision for event-driven dynamic stereo sensor using an adaptive cooperative approach," in *2013 IEEE International Conference on Computer Vision Workshops (ICCVW)* (Sydney, NSW), 45–50. doi: 10.1109/ICCVW.2013.13

Piatkowska, E., Kogler, J., Belbachir, N., and Gelautz, M. (2017). "Improved cooperative stereo matching for dynamic vision sensors with ground truth evaluation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (Honolulu, HI), 370–377. doi: 10.1109/CVPRW.2017.51

Posch, C., Matolin, D., and Wohlgenannt, R. (2010). "A QVGA 143 dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression," in *International Solid-State Circuits Conference Digest of Technical Papers, ISSCC 2010* (San Francisco, CA), 400–401. doi: 10.1109/ISSCC.2010.5433973

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Front. Neurosci.* 9, 1–17. doi: 10.3389/fnins.2015.00141

Reas, C., and Fry, B. (2007). *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press.

Sawada, J., Akopyan, F., Cassidy, A. S., Taba, B., Debole, M. V., Datta, P., et al. (2016). "Truenorth ecosystem for brain-inspired computing: scalable systems, software, and applications," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, UT: IEEE), 130–141. doi: 10.1109/SC.2016.11

Steffen, L., Reichard, D., Weinland, J., Kaiser, J., Roennau, A., and Dillmann, R. (2019). Neuromorphic stereo vision: a survey of bio-inspired sensors and algorithms. *Front. Neurorobot.* 13:28. doi: 10.3389/fnbot.2019.00028

Tippetts, B., Lee, D. J., Lillywhite, K., and Archibald, J. (2016). Review of stereo vision algorithms and their suitability for resource-limited systems. *J. Real Time Image Process.* 11, 5–25. doi: 10.1007/s11554-012-0313-2

Zhu, A. Z., Thakur, D., Özaslan, T., Pfrommer, B., Kumar, V., and Daniilidis, K. (2018). The multivehicle stereo event camera dataset: an event camera dataset for 3D perception. *IEEE Robot. Autom. Lett.* 3, 2032–2039. doi: 10.1109/LRA.2018.2800793

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 13 November 2020
doi: 10.3389/fnbot.2020.577804

Extending the Functional Subnetwork Approach to a Generalized Linear Integrate-and-Fire Neuron Model

Nicholas S. Szczecinski[1]*, *Roger D. Quinn*[2] and *Alexander J. Hunt*[3]

[1] Department of Mechanical and Aerospace Engineering, West Virginia University, Morgantown, WV, United States,
[2] Department of Mechanical and Aerospace Engineering, Case Western Reserve University, Cleveland, OH, United States,
[3] Department of Mechanical and Materials Engineering, Portland State University, Portland, OR, United States

Engineering neural networks to perform specific tasks often represents a monumental challenge in determining network architecture and parameter values. In this work, we extend our previously-developed method for tuning networks of non-spiking neurons, the "Functional subnetwork approach" (FSA), to the tuning of networks composed of spiking neurons. This extension enables the direct assembly and tuning of networks of spiking neurons and synapses based on the network's intended function, without the use of global optimization or machine learning. To extend the FSA, we show that the dynamics of a generalized linear integrate and fire (GLIF) neuron model have fundamental similarities to those of a non-spiking leaky integrator neuron model. We derive analytical expressions that show functional parallels between: (1) A spiking neuron's steady-state spiking frequency and a non-spiking neuron's steady-state voltage in response to an applied current; (2) a spiking neuron's transient spiking frequency and a non-spiking neuron's transient voltage in response to an applied current; and (3) a spiking synapse's average conductance during steady spiking and a non-spiking synapse's conductance. The models become more similar as additional spiking neurons are added to each population "node" in the network. We apply the FSA to model a neuromuscular reflex pathway two different ways: Via non-spiking components and then via spiking components. These results provide a concrete example of how a single non-spiking neuron may model the average spiking frequency of a population of spiking neurons. The resulting model also demonstrates that by using the FSA, models can be constructed that incorporate both spiking and non-spiking units. This work facilitates the construction of large networks of spiking neurons and synapses that perform specific functions, for example, those implemented with neuromorphic computing hardware, by providing an analytical method for directly tuning their parameters without time-consuming optimization or learning.

Keywords: spiking neuron, generalized integrate and fire models, non-spiking neuron, neurorobotics, functional subnetwork approach, synthetic nervous system

Edited by:
Joe Hays,
United States Naval Research Laboratory, United States

Reviewed by:
Vikas Bhandawat,
Drexel University, United States
Erik Christopher Johnson,
Johns Hopkins University, United States

***Correspondence:**
Nicholas S. Szczecinski
nicholas.szczecinski@mail.wvu.edu

Received: *30 June 2020*
Accepted: *08 October 2020*
Published: *13 November 2020*

Citation:
Szczecinski NS, Quinn RD and Hunt AJ (2020) Extending the Functional Subnetwork Approach to a Generalized Linear Integrate-and-Fire Neuron Model. Front. Neurorobot. 14:577804. doi: 10.3389/fnbot.2020.577804

1. INTRODUCTION

Neuromorphic computing hardware is becoming more widely available (Khan et al., 2008; Pfeil et al., 2013; Benjamin et al., 2014; Gehlhaar, 2014; Merolla et al., 2014; Ionica and Gregg, 2015; Davies et al., 2018). Such chips have non-traditional architecture, with highly-parallel processing and specialized circuits that mimic neural and synaptic dynamics. These chips mimic the communication of spiking neural networks, whose discrete communication events (i.e., spikes) reduce the communication overhead relative to continuous networks. Many canonical brain networks have been tested with these chips including decorrelation networks, winner-take-all networks, and balanced random networks (Pfeil et al., 2013), as well as other networks that perform complex computations, such as multi-object recognition (Merolla et al., 2014) and keyword-matching (Blouw et al., 2018), using <100 mW of power in the process.

Neuromorphic hardware is advancing both computational neuroscience (Eliasmith and Anderson, 2002; Eliasmith et al., 2012) and artificial intelligence (Pfeil et al., 2013; Benjamin et al., 2014; Merolla et al., 2014), and soon will play a critical role in robotics. Animals' mobility shows that neuron-based control is effective, and several groups have already developed neural-inspired controllers that could benefit from the low power and parallel computing of neuromorphic hardware (Ayers et al., 2010; Floreano et al., 2014; Dasgupta et al., 2015; Hunt et al., 2017; Szczecinski and Quinn, 2017b; Dürr et al., 2019). However, to apply these neuromorphic chips to robotics, these controllers must be converted into a chip's specific neural model, which may not be trivial. All chips use a variant of the integrate-and-fire model (Brunel and van Rossum, 2007). Toward this goal, we have developed methods for applying our functional subnetwork approach (FSA) for designing non-spiking recurrent neural networks (Szczecinski et al., 2017b) to the specific generalized integrate-and-fire (GLIF) model used by Intel's Loihi chip (Mihalaş and Niebur, 2009; Davies et al., 2018). We will show that these models (i.e., non-spiking and GLIF) have several parallels that enable a network designer to map between them. The details of this comparison are listed at the end of the Introduction.

Setting parameter values in dynamic neural networks can be extremely difficult. Even when network architecture is created directly from animal architecture, parameters cannot be practically measured across all neurons and synapses, and there may be thousands of parameters that dynamically interact. Therefore, these parameter values must be set by the modeler such that the network produces the desired behavior. Some methods and tools have been developed to assist neural designers when mapping network behavior to a desired output, for example, the Neuroengineering Framework and its browser-based design program, Nengo (Eliasmith and Anderson, 2002; Maass and Markram, 2004; Bekolay et al., 2014). These methods seek to build populations of neurons, whose average activity (i.e., spiking frequency) encodes a value of interest. Each population can then interact with others to perform specific operations, such as arithmetic or calculus. This technique is very powerful, enabling the construction of brain-scale networks (Eliasmith, 2013). However, one drawback is that within each population, the connectivity is random, and may not provide insight into how biological networks are structured at small scales. This method is also not ideal for modeling networks with relatively few neurons, such as those that have been described in the locomotion networks of animals (e.g., Bueschges et al., 1994; Sauer et al., 1996; Berg et al., 2015).

As an alternative to this technique we have developed methods for explicitly computing neural and synaptic parameter values for non-spiking dynamical neural networks that perform arithmetic and calculus (Szczecinski et al., 2017b). Such networks are also called "recurrent neural networks," because each neuron's instantaneous state is a function of its own history, producing a form of self-feedback. While such recurrent dynamics can make it difficult to tune networks, such continuous dynamical neural models enable direct analysis of a network's eigenvalues, equilibrium points, and therefore, individual neuron behavior in response to specific inputs (Szczecinski et al., 2017a,b). Such analysis can be difficult to perform on spiking networks, but is particularly important in robotics, in which engineers seek to guarantee a robot's stability and the controller's robustness to parameter changes or sensor noise. The resulting networks are sparse and based on known anatomy, similar to related robotic controllers composed of analog very large scale integration (VLSI) circuits or efficient, discrete-time neuron models (Ayers and Crisman, 1993; Ayers et al., 2010).

Such non-spiking, continuous-valued models theoretically have the same activation dynamics as the average spiking frequency of a population of spiking neurons, all of whom receive the same (noisy) inputs (Wilson and Cowan, 1972). The current manuscript explores this assertion by identifying relationships between the parameter values in the non-spiking model used in our previous work (Szczecinski et al., 2017a,b) and those in a GLIF model (Mihalaş and Niebur, 2009; Davies et al., 2018). Applying the functional subnetwork approach to spiking networks has three primary benefits: First, it enables rapid and direct assembly of spiking networks that have predictable performance; second, it enhances neural robot controllers with richer dynamics than recurrent neural networks; and third, it is a tool for implementing neural controllers on neuromorphic hardware for robots.

In this manuscript, we demonstrate the first of these benefits by extending our previously-developed design tools for non-spiking models (Szczecinski et al., 2017b) to spiking models. We present three "parallels" between the non-spiking and spiking models that enable the extension of our non-spiking network design techniques to spiking networks. This extension includes reducing the impact of non-linear relationships within a network. We derive three parallels between these models:

P1. The steady-state spiking frequency of a spiking neuron is parallel to the steady-state depolarization of a non-spiking neuron because each is proportional to the current applied to the neuron. We refer to both of these quantities as the "activation" of the neuron. The activation of each model can be related to the other via model parameters, and specific

parameter values increase the similarity between spiking frequency and non-spiking depolarization.

P2. The instantaneous spiking frequency of a spiking neuron is parallel to the instantaneous depolarization of a non-spiking neuron. Each exhibits a transient response when stimulated. The decay rate of each model can be related to the other via model parameters, and specific parameter values increase the similarity between the spiking frequency time constant and the non-spiking membrane time constant.

P3. The time-averaged conductance of a spiking synapse is parallel to the conductance of a non-spiking synapse because each is proportional to the activation of the pre-synaptic neuron. Both spiking and non-spiking synapses can be designed to implement a given "gain" value, i.e., the ratio between the post-synaptic (i.e., receiving) and pre-synaptic (i.e., sending) neurons' activations. Specific parameter values increase the similarity between the time-averaged spiking synapse conductance and the non-spiking synapse conductance.

This manuscript is organized as follows. The methods in section 2 present the non-spiking and GLIF models and compute fundamental quantities for each, including equilibrium points and useful relationships between parameter values and variables. We use these expressions to extend our FSA for designing non-spiking networks to spiking models. The results in section 3 demonstrate parallels P1-P3 and leverage them into a sequential process for designing a spiking pathway. In section 4, the results from section 3 are applied to a neuromuscular model of a stretch reflex, and the resulting motion of the models is compared. Finally, the discussion in section 5 summarizes the work, explains how neurobiologists and roboticists can apply this work to their research, and proposes future work. To aid the reader, variable names are defined in **Table 1**.

TABLE 1 | List of variables and descriptions.

Variable	Description
NON-SPIKING	
$\bar{U}$	Membrane voltage, state variable
$\bar{C}_{mem}$	Membrane capacitance, constant parameter
G_{mem}	Membrane conductance/leak conductance, constant parameter
I_{app}	Applied current, input variable
$\bar{G}_{max}$	Maximum non-spiking synaptic conductance, constant parameter
$\bar{G}_s$	Instantaneous non-spiking synaptic conductance, piecewise linear function of the pre-synaptic neuron's voltage
$\bar{E}_s$	Non-spiking synaptic reversal potential, constant parameter
SPIKING	
U	Membrane voltage, state variable
θ	Spiking threshold, state variable
C_{mem}	Membrane capacitance, constant parameter
G_{mem}	Membrane conductance/leak conductance, constant parameter
I_{app}	Applied current, input variable
θ_0	Initial spiking threshold, constant parameter
τ_θ	Spiking threshold time constant, constant parameter
m	Proportionality constant that determines the change in θ relative to U, constant parameter
G_{max}	Maximum spiking synaptic conductance, constant parameter
G_s	Instantaneous synaptic conductance, state variable
τ_s	Synaptic time constant, constant parameter
E_s	Spiking synaptic reversal potential, constant parameter

2. METHODS

In this section, we present both the non-spiking model and the spiking model. For each, we compute parameter values necessary to demonstrate parallels P1–P3. Then we briefly summarize the philosophy behind the FSA.

2.1. Non-spiking Neuron and Synapse Models

The non-spiking model is a leaky integrator, or recurrent neural model (Beer and Gallagher, 1992). Such a model describes the subthreshold dynamics of a neuron with the differential equation

$$\bar{C}_{mem} \cdot \frac{d\bar{U}}{dt} = -G_{mem} \cdot \bar{U} + \sum_{i=1}^{n} \bar{G}_{s,i} \cdot (E_{s,i} - \bar{U}) + I_{app} + I_{bias}, \quad (1)$$

where $\bar{U}$ is the non-spiking neuron voltage above its rest potential (referred to as "membrane depolarization" throughout, see Szczecinski et al., 2017b for more detail), $\bar{C}_{mem}$ is the capacitance of the cell membrane, G_{mem} is the leak conductance, $\bar{G}_{s,i}$ is the instantaneous conductance of the ith incoming non-spiking (i.e., graded) synapse, $E_{s,i}$ is the reversal potential of the i^{th} incoming synapse relative to the neuron's rest potential, I_{app} is the applied current that encodes information (e.g., muscle stretch, as shown in **Figure 1**), and I_{bias} is the constant offset current. The synaptic conductance $\bar{G}_s$ is a piecewise linear function of the pre-synaptic neuron voltage,

$$\bar{G}_s = \bar{G}_{max,i} \cdot \begin{cases} 0, & \text{if } \bar{U}_{pre} \le 0, \\ \frac{\bar{U}_{pre}}{R}, & \text{if } 0 < \bar{U}_{pre} < R, \\ 1, & \text{if } \bar{U}_{pre} \ge R. \end{cases} \quad (2)$$

$\bar{G}_{max}$ is the maximum synaptic conductance, and R is the maximum membrane depolarization of neurons in the network (Szczecinski et al., 2017b). Parameters with a bar (e.g., $\bar{U}$) are those that relate to the non-spiking model, and will be mapped to analogous parameters in the spiking model in section 2.2. The results in section 3 explain how to map from these values into their spiking model counterparts.

The steady-state membrane depolarization, $\bar{U}_\infty$, is calculated by solving Equation (1) when $d\bar{U}/dt = 0$,

$$\bar{U}_\infty = \frac{\sum_{i=1}^{n} \bar{G}_{s,i} \cdot E_{s,i} + I_{app} + I_{bias}}{\sum_{i=1}^{n} \bar{G}_{s,i} + G_{mem}}. \quad (3)$$

Note that the equilibrium voltage is effectively the average of incoming synaptic potentials E_s, weighted by their conductances $\bar{G}_s$. If a neuron receives an applied current but has no incoming

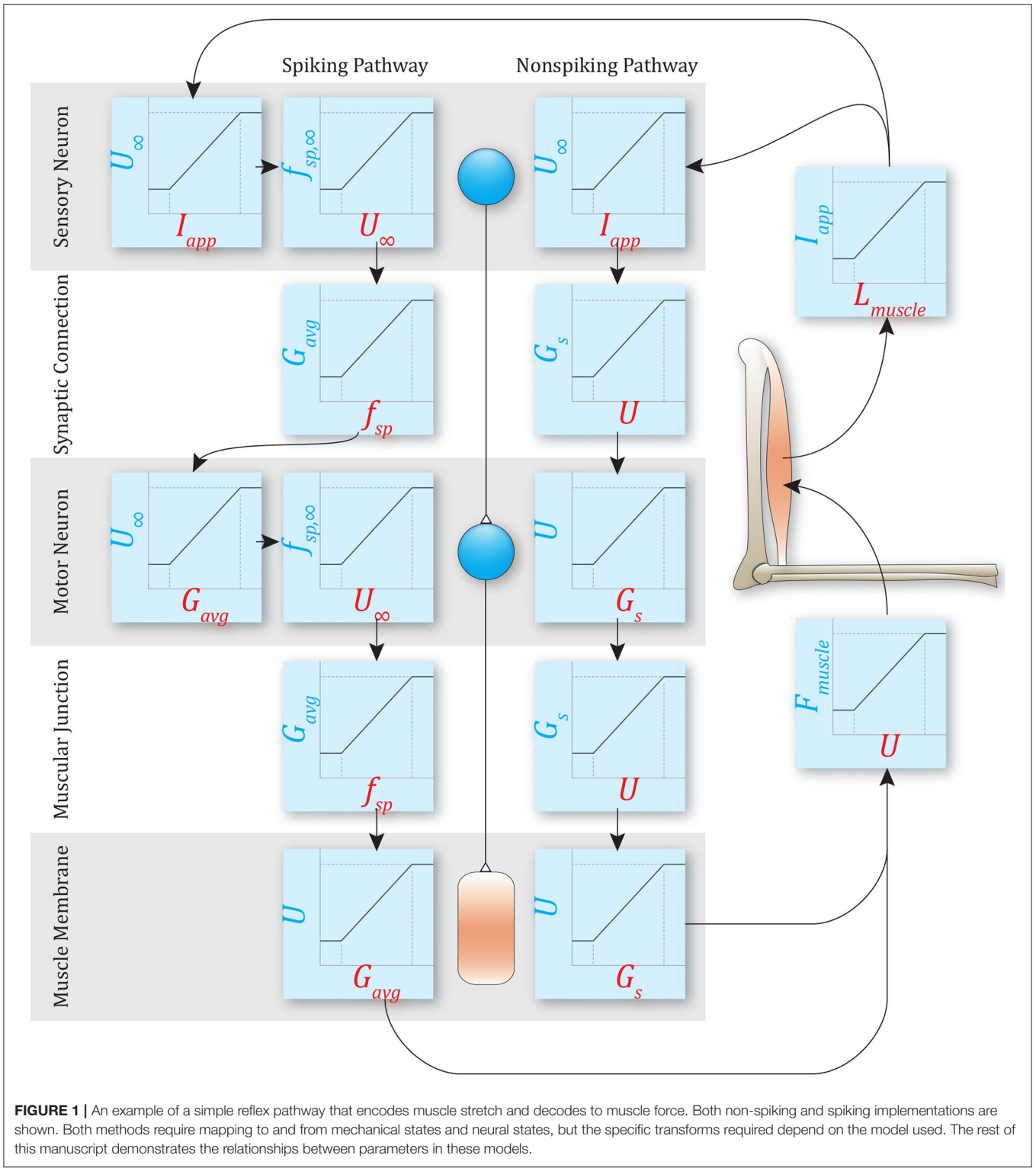


FIGURE 1 | An example of a simple reflex pathway that encodes muscle stretch and decodes to muscle force. Both non-spiking and spiking implementations are shown. Both methods require mapping to and from mechanical states and neural states, but the specific transforms required depend on the model used. The rest of this manuscript demonstrates the relationships between parameters in these models.

synaptic connections, this expression simplifies to:

$$\bar{U}_\infty = \frac{I_{app} + I_{bias}}{G_{mem}}. \tag{4}$$

If $I_{bias} = 0$, then $\bar{U}$ is directly proportional to I_{app}. This expression will be used to demonstrate parallel P1 between the models.

How does the non-spiking model's response evolve over time? The response to a tonic current that is turned on at time

$t = 0$ is

$$\bar{U}(t) = \bar{U}_\infty \cdot (1 - e^{-t/\tau_{mem}}), \tag{5}$$

where U_∞ comes from Equation (3), and

$$\tau_{mem} = \frac{C_{mem}}{\sum_{i=1}^{n} G_{s,i} + G_{mem}}. \tag{6}$$

The transient response of $\bar{U}$ decays with time constant τ_{mem}, which will be compared to the spiking model's transient spiking frequency, demonstrating parallel P2 between the models.

The effect of a synaptic input on the post-synaptic neuron can be quantified by the "gain" of the synapse. We define the gain $k_{syn} = \bar{U}_{\infty,post}/\bar{U}_{\infty,pre}$, the ratio between the post-synaptic and pre-synaptic steady-state voltage. Substituting the expression for synaptic conductance in Equation (2) into Equation (3) relates the synapse's parameter values to its gain (Szczecinski et al., 2017b):

$$\bar{G}_{max} = \frac{k_{syn} \cdot R}{E_s - k_{syn} \cdot R}. \tag{7}$$

This expression will be extended to design spiking synapses, once we demonstrate parallel P3 between the models.

2.2. Spiking Neuron and Synapse Model

The spiking model used in this work is a generalized leaky integrate and fire (GLIF) model (Mihalaş and Niebur, 2009; Davies et al., 2018). The dynamics of the membrane voltage are identical to Equation (1) (Equation 8), except that U is reset to 0 after crossing the spiking threshold, θ (Equation 10). In addition, the threshold θ is itself a dynamical variable that evolves according to Equation (9). The dynamics of U and θ interact to produce a diverse set of spiking responses (Mihalaş and Niebur, 2009).

The spiking model's dynamics are:

$$C_{mem} \cdot \frac{dU}{dt} = -G_{mem} \cdot U + \sum_{i=1}^{n} G_{s,i} \cdot (E_{s,i} - U) + I_{app} + I_{bias} \tag{8}$$

$$\tau_\theta \cdot \frac{d\theta}{dt} = -\theta + \theta_0 + m \cdot U \tag{9}$$

$$\text{if } U \geq \theta, 0 \leftarrow U, \tag{10}$$

where the variable names and functions are the same as in Equation (1), with the exception of the synaptic conductance $G_{s,i}$ (described below); and the threshold variable θ, whose equation includes the threshold time constant τ_θ, the initial threshold θ_0, and the proportionality constant that specifies how θ changes in response to changes in U, called m.

The spiking synapse conductance G_s is reset to its maximum value G_{max} when the pre-synaptic neuron spikes, and decays to 0 with a time constant τ_s,

$$\tau_s \cdot \frac{dG_s}{dt} = -G_s. \tag{11}$$

The steady-state threshold value θ_∞ is calculated by solving Equation (9) when $d\theta/dt = 0$,

$$\theta_\infty = \theta_0 + m \cdot U_\infty, \tag{12}$$

where U_∞ is the neuron's steady-state membrane depolarization (as in Equation 3), if the spiking mechanism (Equation 10) is disabled. We will refer to U_∞ as the "target voltage." When the spiking mechanism is enabled, then steady-state spiking occurs if $U_\infty > \theta_\infty$, which we show below.

The steady-state spiking frequency of a neuron f_{sp} is the inverse of the time required for the neuron's membrane potential in Equation (5) to cross the threshold $\theta(t)$,

$$f_{sp} = \frac{-1}{\tau_{mem} \cdot \ln\left(1 - \frac{\theta}{U_\infty}\right)}. \tag{13}$$

Equation (13) indicates that if the target voltage is below the threshold (i.e., $U_\infty < \theta$), then the argument of ln() becomes <0 and the frequency is undefined because no spikes can occur. Increasing the target voltage U_∞ (e.g., via synaptic inputs or applied current) or decreasing the threshold θ increases f_{sp} by decreasing the amount of time needed for $U(t)$ to reach θ. The spiking frequency can also be increased by reducing τ_{mem}, which similarly reduces the time needed for $U(t)$ to reach θ.

If the spiking threshold *is not* dependent on the membrane voltage (i.e., $m = 0$ in Equation 10), then $\theta(t) = \theta_0$, and Equation (13) is used to calculate the spiking frequency explicitly, given the target voltage U_∞. According to Equation (3), U_∞ is a function of I_{app}, so Equation (13) provides the $I_{app} \rightarrow f_{sp}$ mapping necessary to demonstrate parallel P1 when $m = 0$.

The spiking threshold may depend on the membrane potential ($m \neq 0$ in Equation 10), for example, when modeling neurons that exhibit class 2 excitability, frequency adaptation, or other non-constant spiking frequency responses (Mihalaş and Niebur, 2009). In such cases, Equation (13) is still used to calculate the spiking frequency. However, since θ is a dynamical variable, the value of θ at the instant that a spike occurs, called θ^* must be found. Additionally, θ^* may change from spike to spike, so in fact one must solve for θ^*_∞, the instantaneous spiking threshold for each spike during steady-state spiking. This amounts to solving the following implicit equation, which is derived in the **Supplementary Materials S1.1.1** and **S1.1.2**:

$$f(\theta^*_\infty) = 0 = \begin{cases} (\theta_\infty - \theta^*_\infty) \cdot \frac{\theta^*_\infty}{U_\infty} + m \cdot U_\infty \cdot \left(1 - \frac{\theta^*_\infty}{U_\infty}\right) \cdot \ln\left(1 - \frac{\theta^*_\infty}{U_\infty}\right), \\ \quad \text{if } \tau_{mem} = \tau_\theta; (\theta_\infty - \theta^*_\infty) \cdot \left(1 - (1 - \frac{\theta^*_\infty}{U_\infty})^{\tau_{mem}/\tau_\theta}\right) + \\ \qquad \frac{m \cdot U_\infty \cdot \tau_{mem}}{\tau_\theta - \tau_{mem}} \cdot \left((1 - \frac{\theta^*_\infty}{U_\infty}) - (1 - \frac{\theta^*_\infty}{U_\infty})^{\tau_{mem}/\tau_\theta}\right), \text{ else.} \end{cases} \tag{14}$$

Equation (14) implicitly describes the relationship between the target voltage U_∞ and the threshold at the instant a spike occurs θ^*_∞, and must be solved numerically. Once θ^*_∞ is found, the steady-state spiking frequency is calculated by substituting $\theta = \theta^*_\infty$ in Equation (13).

A more easily computed but less accurate explicit function approximation of θ^*_∞ based on U_∞ is shown below in Equation (16). This approximation follows from observing that when f_{sp} is large, two key phenomena arise: First, there is less time between spikes for θ to fluctuate, so its average value is a good estimate of its instantaneous value; second, the average voltage U_{avg} approaches $\theta^*_\infty/2$ (**Supplementary Materials S1.1.3**), enabling the usage of Equation (12) to calculate

$$\theta^*_\infty = \theta_0 + m \cdot \frac{\theta^*_\infty}{2}. \tag{15}$$

Rearranging for θ^*_∞ yields

$$\theta^*_\infty = \frac{\theta_0}{1 - m/2} = B \cdot \theta_0, \tag{16}$$

where $B = 1/(1-m/2)$. The approximation in Equation (16) has the advantage of being explicit. However, it is only accurate when $1/f_{sp} << \tau_\theta$. Its utility will be explored in the results (section 3).

2.2.1. Average Synaptic Conductance

The average synaptic conductance G_{avg} is computed by solving for $G_s(t)$ and calculating its average value over a duration of one interspike period, $T_{sp} = 1/f_{sp}$. The synaptic conductance evolves according to Equation (11). Because a pre-synaptic spike resets the conductance to G_{max}, the conductance after a spike at time $t = 0$ is simply

$$G_s(t) = G_{max} \cdot e^{-t/\tau_s}. \tag{17}$$

The average conductance G_{avg} given a steady-state pre-synaptic interspike period T_{sp} can be calculated by integrating Equation (17) over the interval $[0, T_{sp}]$ and dividing by T_{sp}. Performing this integral,

$$G_{avg} = G_{max} \cdot \tau_s \cdot f_{sp} \cdot (1 - e^{-1/f_{sp} \cdot \tau_s}). \tag{18}$$

The average conductance G_{avg} is directly proportional to the pre-synaptic spiking frequency f_{sp} except for the influence of the exponential term, which we define as

$$\delta = e^{-1/f_{sp} \cdot \tau_s} \tag{19}$$

This equation will be necessary to tune synaptic connections in the network.

2.3. Network Construction

To control motion, the nervous system must map from mechanical quantities (e.g., muscle stretch) to neural quantities (e.g., sensory neuron voltage, spiking frequency, etc.), perform computations, and then map neural output back into mechanical quantities (e.g., muscle force) (Eliasmith and Anderson, 2003; Szczecinski et al., 2017b; Hilts et al., 2019). Thus, the network "encodes" the mechanical state of the animal or robot, performs control computations, and then "decodes" the required actuator forces. For example, **Figure 1** illustrates a simple stretch reflex, implemented as both non-spiking and spiking pathways. The type of encoding and decoding that take place in each pathway depends on whether it is spiking or non-spiking, but the computation itself should not differ.

In this manuscript, each non-spiking neuron has a maximum expected membrane depolarization R, and each spiking neuron has a maximum expected spiking frequency F_{max}. The minimum value in each case is 0. Specifying an expected range of activity for each type of network simplifies network tuning in two ways. First, it enables a clear comparison between non-spiking and spiking implementations of the same network. The activation of analogous nodes in two networks should be equal once normalized to the range of activity, e.g., $\bar{U}/R = f_{sp}/F_{max}$. When all of a network's nodes have the same activity scale, more specific and precise computations can be designed within the network (Szczecinski et al., 2017b). Second, it enables synaptic connections between serial nodes in the same network to be parameterized by the gain k_{syn}, whether non-spiking or spiking models are used. The gain can be formulated as either $k_{syn} = \bar{U}_{post}/\bar{U}_{pre}$ or $k_{syn} = f_{post}/f_{pre}$. Normalizing network activity in this way will enable us to apply the same design constraints derived in Szczecinski et al. (2017b) to the spiking model.

2.4. Simulation

All simulations were implemented in Matlab (The Mathworks, Natick, MA). Neurons were initialized with a random initial depolarization $U(0) \in [0, \theta_0]$. This added some variation to the simulation. All units were scaled to ms, mV, nA, nF, and μS (MΩ). Dynamics were simulated using the forward Euler method, with time step $\Delta t = \min(\tau_{mem}) \cdot 10^{-4}$ ms.

3. RESULTS

3.1. Comparison of Non-spiking and Spiking Neuron Activation

The steady-state spiking frequency f_{sp} is approximately a linear function of the applied current I_{app}. Equation (4) shows that the non-spiking model's voltage $\bar{U}$ is directly proportional to I_{app}. However, Equation (13) shows that the spiking frequency of the spiking model f_{sp} is a transcendental function of the neuron's applied current I_{app}. Despite its transcendental nature, it can be bounded by parallel lines,

$$\frac{I_{app}}{G_{mem} \cdot \tau_{mem} \cdot \theta^*_\infty} - \frac{1}{2 \cdot \tau_{mem}} \le f_{sp}(I_{app}, \theta^*_\infty) \le \frac{I_{app}}{G_{mem} \cdot \tau_{mem} \cdot \theta^*_\infty} + \frac{1}{2 \cdot \tau_{mem}}, \tag{20}$$

provided that

$$I_{bias} = \frac{G_m \cdot \theta^*_\infty}{2}. \tag{21}$$

This inequality is derived in **Supplementary Materials (S1.1.4)**. For any I_{app}, the precise value of θ^*_∞ is calculated numerically with Equation (14), or approximated with the explicit function in Equation (16). Then, Equation (20) provides affine bounds on the spiking frequency, with a range of $\pm 1/(2 \cdot \tau_{mem})$. **Figures 2A,B**

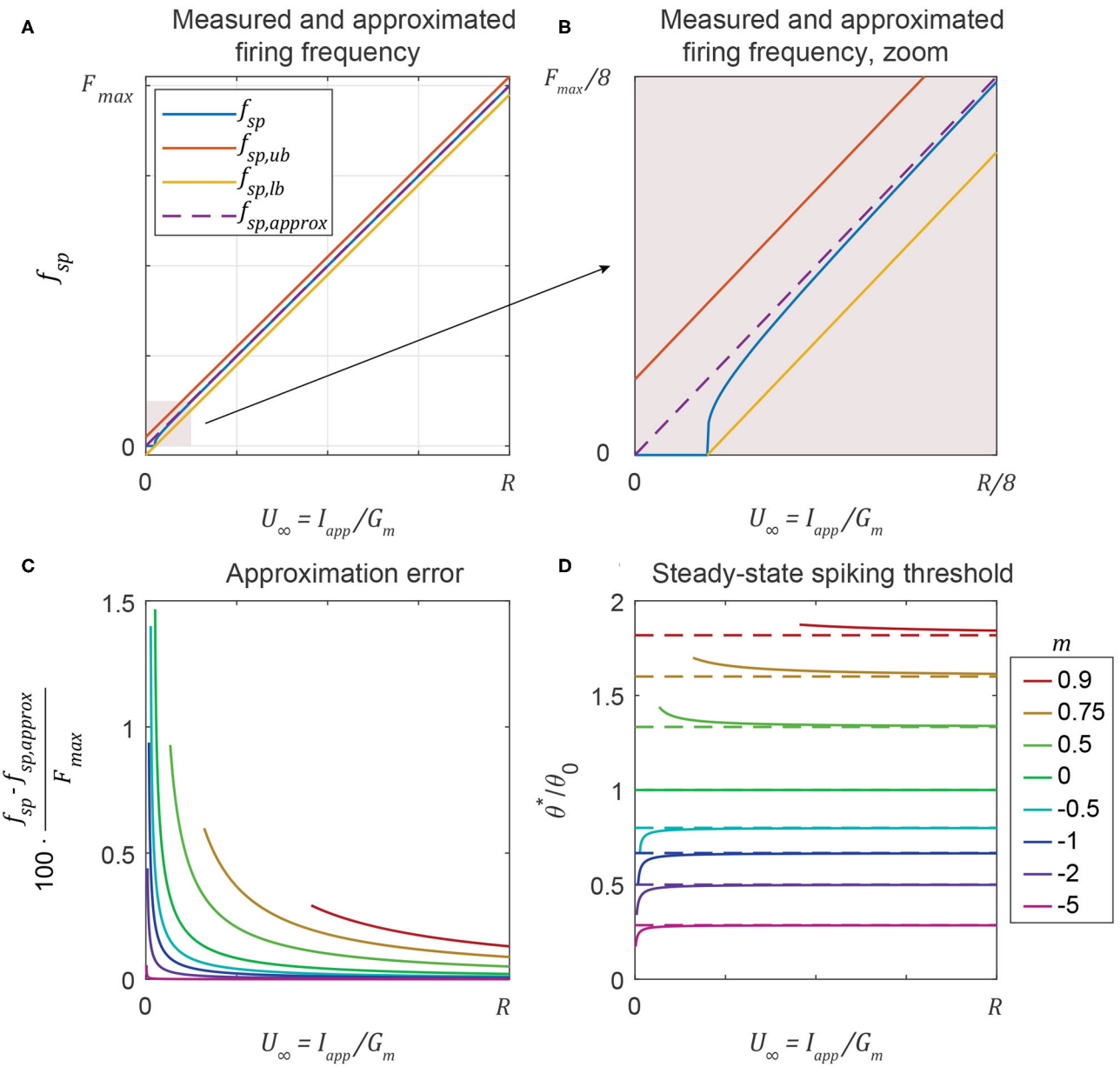


FIGURE 2 | (A) A neuron's spiking frequency f_{sp} is an approximately linear function of the steady-state membrane voltage U_∞, staying between the upper bound $f_{sp,ub} = (U_\infty + \theta^*_\infty/2)/(\tau_{mem} \cdot \theta^*_\infty)$ and the lower bound $f_{sp,lb} = (U_\infty - \theta^*_\infty/2)/(\tau_{mem} \cdot \theta^*_\infty)$ (Equation 20). **(B)** Zooming in shows that the linear approximation is poor at low frequencies. **(C)** The error between the measured and approximated spiking frequency is a function of both U_∞ and m, but stays below 2% in all cases. **(D)** Even if the spiking threshold can change over time ($m \neq 0$), neurons tend to spike at the same threshold no matter the value of U_∞. This threshold approaches the explicit approximation in Equation (16) (dashed lines) as U_∞ increases.

plot the three terms in Equation (20) vs. I_{app}/G_{mem}. **Figure 2A** shows that as I_{app} increases, f_{sp} is an approximately linear function of I_{app}, and approaches the mean of the bounds from Equation (20). The resulting approximation is

$$f_{sp,approx}(I_{app}, \theta^*_\infty) = \frac{I_{app}}{G_{mem} \cdot \tau_{mem} \cdot \theta^*_\infty}. \tag{22}$$

Figure 2B shows that the linear approximation in Equation (22) breaks down for very small values of I_{app}, but **Figure 2C** shows that the error asymptotically approaches 0 as I_{app} increases, no matter the value of m. **Figure 2D** shows that for any value of m, θ^*_∞ approaches the value in Equation (16) as I_{app} (and f_{sp}) increases.

Equation (22) reveals the parallel between the non-spiking activation $\bar{U}$ and the spiking activation f_{sp}: Both are directly proportional to I_{app} (provided the bias current is tuned according to Equation 21). To strengthen this parallel, τ_{mem} can be tuned such that each activation value equals its maximum value when the input I_{app}

equals its maximum value, $G_{mem} \cdot R$. The resulting constraint is

$$\tau_{mem} = \frac{R}{F_{max} \cdot \theta^*_\infty}. \tag{23}$$

This condition implies that given the same input current I_{app}, the non-spiking and spiking activations can be mapped to each other through the relationship

$$f_{sp,approx}(I_{app}, \theta^*_\infty) = \frac{F_{max}}{R} \cdot \bar{U}_\infty, \tag{24}$$

The results show that the spiking frequency is parallel to the membrane depolarization of the non-spiking model. Each activation state is a linear function of the applied current, and can be related to each other by the factor F_{max}/R. This supports parallel P1 from the Introduction: Non-spiking neuron voltage is analogous to the spiking neuron's spiking frequency.

3.2. Comparison of Non-spiking and Spiking Activation Transient Responses

The transient spiking threshold θ^* can be tuned to approximate the transient response of the non-spiking membrane depolarization $\bar{U}$. Knowing the steady-state spiking threshold at the time of spiking, θ^*_∞, one can calculate the evolution of θ^* from spike to spike. This evolution describes the transient response of the neuron's spiking frequency in response to a step-input current. The following expression is derived in Appendix 1.4:

$$\theta^*(t) = \theta^*_\infty + (\theta_0 - \theta^*_\infty) \cdot e^{-t/\tau_{\theta^*}}, \tag{25}$$

where θ^*_∞ is calculated via Equation (14) or Equation (16) and $\tau_{\theta^*} = \tau_\theta \cdot B$ (Equation 46).

Equation 25 describes how the spiking threshold, and thus the spiking frequency, evolves from spike time to spike time. This is analogous to the non-spiking neuron membrane transient response from Equation (5), whose time constant is defined in Equation (6). The time constants in a non-spiking functional subnetwork (e.g., as designed according to Szczecinski et al., 2017b) are mapped to the threshold's time constant in a spiking neuron network with the equation

$$\tau_\theta = \bar{\tau}_{mem} \cdot \left(1 - \frac{m}{2}\right). \tag{26}$$

Figure 3A plots the response of θ and f_{sp} when $m = -5$, compared to the response of a non-spiking neuron voltage $\bar{U}$, scaled by F_{max}/R (Equation 24). **Figures 3Ai–iii** show that Equation (25) accurately predicts the spiking threshold at each spike time, even as different values of τ_θ are used. **Figures 3Aiv–vi** show that when $m = -5$, f_{sp} evolves smoothly, because θ^* evolves quickly and with a large amplitude.

Figure 3B plots the transient responses when $m = -0.5$. θ^* is predicted accurately in **Figures 3Bi–iii**. However, the spiking frequency discontinuously leaps from 0 to a non-zero value as shown in **Figures 3Biv–vi**. Since θ has the same transient time constant as $\bar{U}$, the transient firing frequency has the same decay rate as the non-spiking neuron voltage. However, the amplitude does not map directly.

Finally, **Figures 3Ci–iii** show that this analysis applies even when $m > 0$, that is, when the threshold *increases* as U increases. **Figure 3C** does not include plots of f_{sp} or $\bar{U}$ because when $m > 0$, f_{sp} *decreases* over time, a behavior that the non-spiking model cannot reproduce.

The data in **Figure 3A** show that when $m << 0$, the firing frequency f_{sp} evolves smoothly at the same rate as $\bar{U}$ scaled by F_{max}/R, when subjected to the same current input. This result supports parallel P2 from the Introduction: The transient spiking threshold θ^* mimics the transient membrane depolarization $\bar{U}$ as long as $\tau_{\theta^*} = \tau_\theta \cdot B = \bar{\tau}_{mem}$.

3.3. Comparison of Non-spiking and Spiking Synaptic Conductance

The average spiking synaptic conductance G_{avg} can be tuned to approximate the non-spiking synaptic conductance $\bar{G}_s$. G_{avg} is approximately proportional to f_{sp} when τ_s is sufficiently small. Decreasing τ_s improves this approximation by reducing the impact of δ, the exponential term in Equation (18). **Figure 4** illustrates δ graphically. After selecting a value for δ, Equation (19) can be solved to compute the upper limit of τ_s,

$$\tau_s \leq \frac{-1}{F_{max} \cdot \ln \delta}. \tag{27}$$

For example, if $\delta = 0.01$ (1% deviation from a linear relationship) and $F_{max} = 0.1$ kHz, then $\tau_s \leq 2.17$ ms. Equation 27 is necessary to ensure that the average synaptic conductance G_{avg} is an approximately linear function of the pre-synaptic neuron's firing frequency f_{sp}.

Figure 4 plots G_{avg} vs. f_{sp} for various parameter value combinations. δ is different in each column. In each case, τ_s is calculated using Equation (27). When δ is small ($\leq$ 1%), the average spiking synaptic conductance is almost directly proportional to the spiking frequency of the pre-synaptic neuron. This is analogous to the non-spiking synaptic conductance, which is proportional to the membrane depolarization of the pre-synaptic neuron (Equation 2). This result supports parallel P3 from the Introduction: The average spiking synaptic conductance G_{avg} is proportional to the pre-synaptic neuron's activation. However, G_{avg} is an emergent property of the synapse; the designer can only set the value of G_{max}. How should G_{max} be set? Once the desired value for G_{avg} is known (see section 3.4), G_{max} is determined by rearranging Equation (18),

$$G_{max} = \frac{G_{avg}}{\tau_s \cdot F_{max} \cdot (1 - \delta)}. \tag{28}$$

What if one wishes to set G_{max} to achieve a particular functional gain, that is, ratio of firing frequencies between the post-synaptic and pre-synaptic neurons (i.e., $f_{sp,post}/f_{sp,pre}$)? In this case, one sets G_{avg} in Equation (28) equal to the equivalent non-spiking synaptic conductance in Equation (7), which is expressed

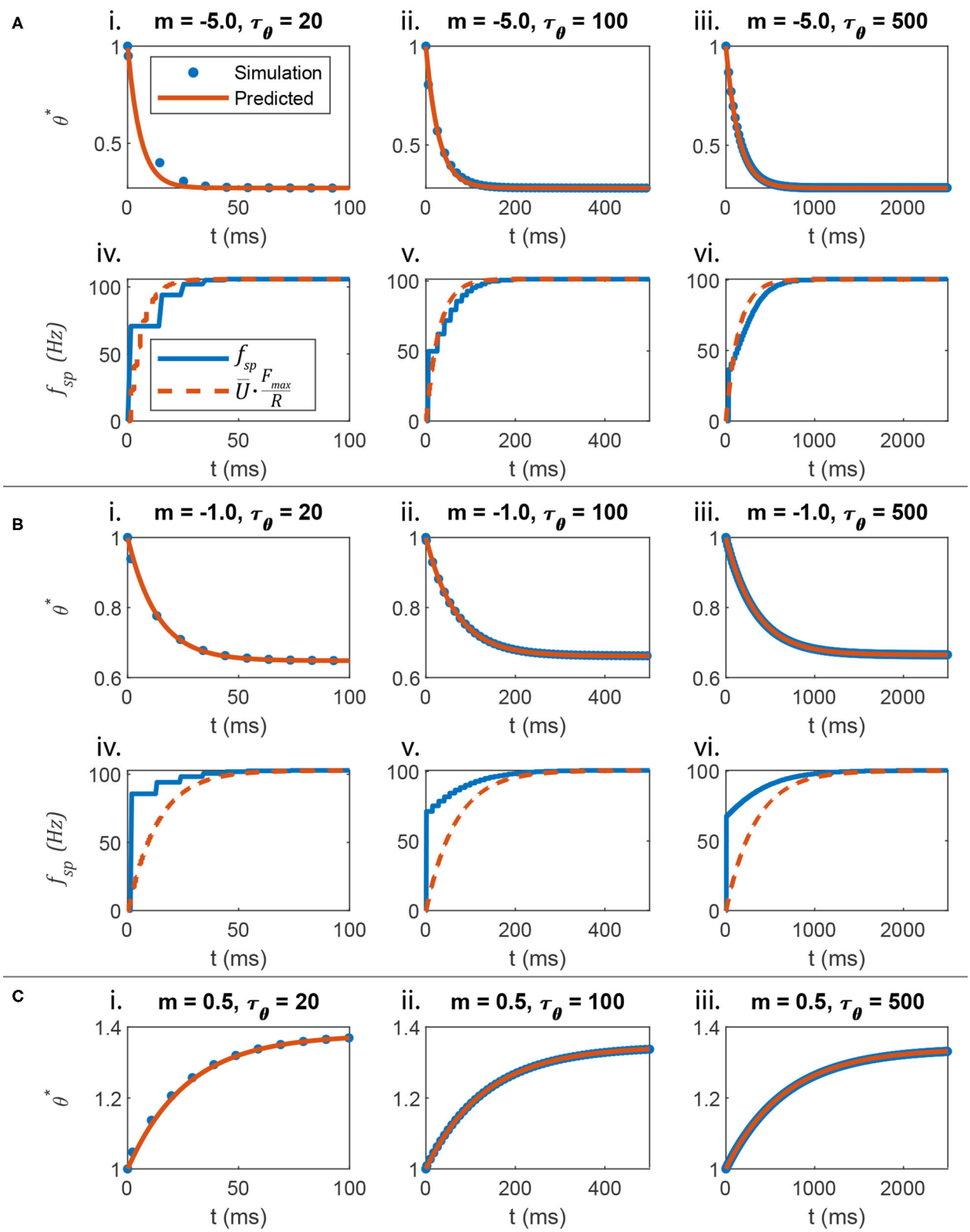


FIGURE 3 | The evolution of the spiking threshold when a spike occurs can be predicted. **(A)** When the threshold strongly hyperpolarizes in response to membrane voltage depolarization, the transient spiking frequency evolves much like the non-spiking neuron's membrane voltage. **(B)** When the threshold weakly hyperpolarizes in

(Continued)

FIGURE 3 | response to membrane voltage depolarization, the transient spiking frequency evolves at the same rate as a non-spiking neuron's membrane voltage, but the amplitude matches poorly. This is because the spiking neuron's membrane must depolarize strongly before firing even one spike, leading to a large leap in frequency from 0 to a non-zero value. **(C)** This analysis also applies if the spiking threshold depolarizes in response to membrane voltage depolarization (i.e., $m > 0$). However, the comparison between f_{sp} and U is not shown because in this case, f_{sp} begins large and decreases over time, rather than beginning small and increasing over time, like $\bar{U}$.

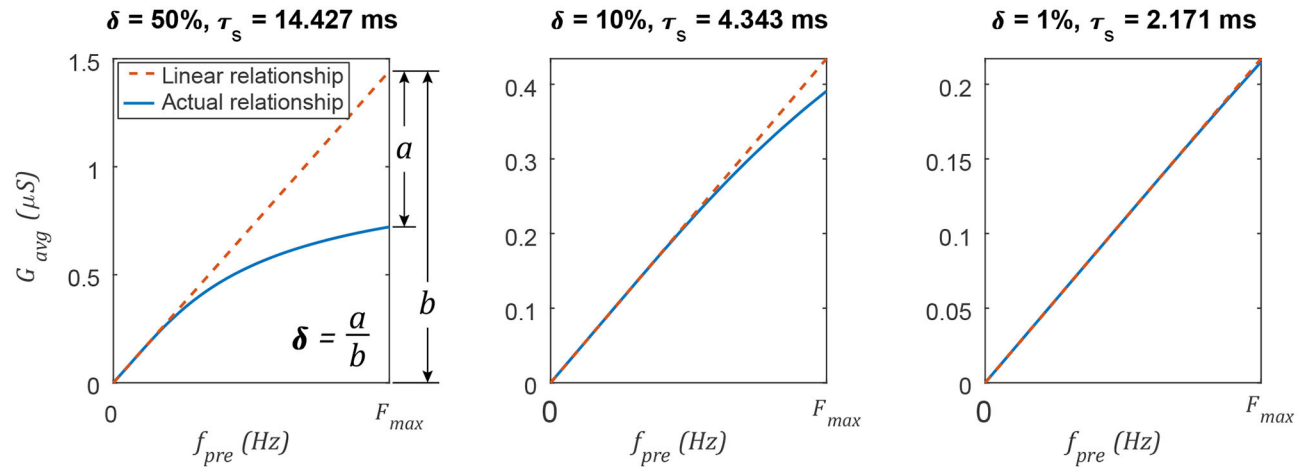


FIGURE 4 | Decreasing the synaptic time constant τ_s increases the linearity of the average synaptic conductance G_{avg} as a function of the pre-synaptic neuron's spiking frequency f_{pre}. Changing the network's maximum spiking frequency F_{max} will change the specific value of τ_s, but the linearity δ varies in the same way.

in terms of k_{syn}, the functional synaptic gain. As a further simplification, let us assume $\delta \approx 0$, in which case:

$$G_{max} = \frac{k_{syn} \cdot R}{(E - k_{syn} \cdot R) \cdot \tau_s \cdot F_{max}}. \tag{29}$$

This equation enables the direct design of synaptic conductance based on synaptic gain.

3.4. Extending the Functional Subnetwork Approach for Designing Non-spiking Networks to Spiking Networks

Having established analogous parameters between non-spiking and spiking networks, we can now adapt the functional subnetwork approach (FSA) (Szczecinski et al., 2017b) to spiking networks. The goal of the FSA is to determine the parameter values for individual neurons and synapses based on network-wide parameter values (e.g., R and F_{max}) and the intended function of a portion of a network. In the non-spiking framework we previously developed, one could construct networks that could add two (or N) input signals using three (or $N+1$) neurons; subtract two signals using three neurons; multiply two signals using four neurons; divide two signals using three neurons; differentiate a signal using four neurons; and integrate a signal over time using two neurons. The accuracy of the FSA has a practical limit, especially when parameter values are constrained to biologically plausible bounds. However, the FSA enables the rapid, direct assembly of networks that can perform sophisticated tasks, such as entraining rhythmic output to periodic inputs (Nourse et al., 2018) or learning the appropriate force to apply to the environment (Szczecinski and Quinn, 2017a), right "out of the box." Such networks also serve as good starting configurations for subsequent optimization (Pickard et al., 2020). As such, we find the FSA to be useful for designing computational models and robot controllers.

How does one use the FSA to select parameter values based on network function? As an example, consider a pathway in which one neuron's activation may represent the average of two other neurons' activation. To accomplish this, the synapses connecting these neurons should each have gain $k_{syn} = 1/2$, such that $f_3 = 1/2 \cdot (f_1 + f_2)$. For another example, one neuron's activation may represent the difference between two other neurons' activation if the synapses have equal and opposite gain, e.g., $k_{syn,1} = 1$, $k_{syn,2} = -1$ (and the second synapse has a negative reversal potential). The FSA ensures that each neuron encodes the intended quantities and performs the intended operation by tuning parameter values in an algorithmic way.

Table 2 contains the FSA for spiking networks. The designer first sets network-wide values for the spiking neuron parameters F_{max}, θ_0, R, either based on biological data or arbitrarily. Applying the same values across the entire network ensures proper encoding and decoding (e.g., **Figure 1**). Next, the designer sets m and τ_θ based on the desired transients in the system. Then I_{bias} and τ_{mem} are calculated so that $f_{sp} = F_{max}$ when $I_{app}/G_m = R$. After the neural parameters, the synaptic parameters are tuned. The synaptic decay constant τ_s is calculated to ensure the average synaptic conductance is proportional to the pre-synaptic neuron's spiking frequency. Finally, the maximum synaptic conductance G_{max} is calculated to achieve the intended average synaptic conductance.

TABLE 2 | Expanded functional subnetwork approach for designing spiking pathways.

Step	Goal	Param.	Eq. No.	Equation
1.	Set network-wide activation parameters.	F_{max}, R, and θ_0	24	$f_{sp,approx} = \frac{F_{max}}{R} \cdot \bar{U}_\infty$
2.	For each neuron, set transient response type based on function in the network.	m		For $\frac{\mathrm{d}}{\mathrm{dt}} f_{sp} > 0$, set $m < 0$. For $\frac{\mathrm{d}}{\mathrm{dt}} f_{sp} = 0$, set $m = 0$. For $\frac{\mathrm{d}}{\mathrm{dt}} f_{sp} < 0$, set $m > 0$.
3.	For each neuron, set the duration of transient firing based on its function in the network.	τ_θ	26	$\tau_\theta = \bar{\tau}_{mem} \cdot (1 - \frac{m}{2})$
4.	For each neuron, calculate bias current.	I_{bias}	21	$I_{bias} = \frac{G_{mem} \cdot \theta_0}{2-m}$
5.	For each neuron, ensure $f_{sp} \in [0, F_{max}]$.	τ_{mem}	23	$\tau_{mem} = \frac{R}{F_{max}} \cdot \frac{1-\frac{m}{2}}{\theta_0}$
6.	For each synapse, limit its non-linearity.	τ_s	27	$\tau_s \leq \frac{-1}{F_{max} \cdot \ln \delta}$
7.	For each synapse, set synaptic gain k_{syn} based on function in the network.	G_{max}	29	$G_{max} = \frac{k_{syn} \cdot R}{(E_s - k_{syn} \cdot R) \cdot \tau_s \cdot F_{max}}$

As an example, let us design a pathway wherein the post-synaptic neuron's spiking frequency mirrors its pre-synaptic neuron's spiking frequency. Following **Table 2**, the steps are:

1. Arbitrarily, but in the range of biological systems, pick the maximum spiking frequency $F_{max} = 0.1$ kHz, the maximum membrane depolarization $R = 20$ mV, and the initial firing threshold $\theta_0 = 1$ mV. These are arbitrary quantities, but may be tuned to match a specific pathway if data is available.
2. Pick $m = 0$ so θ is constant, and each neuron has no transient spiking frequency.
3. Since $m = 0$, there is no transient, and this step is skipped.
4. Calculate $I_{bias} = 0.5$ nA, so that $f_{sp} = 0$ when $I_{app} = 0$ and $f_{sp} = F_{max}$ when $I_{app} = R$.
5. Using the values from steps 1 to 4, calculate $\tau_{mem} = 200$ ms. The neuron properties are now set.
6. Set $\delta = 1\%$ such that G_{avg} is nearly directly proportional to f_{sp} of the pre-synaptic neuron, with a maximum deviation of 1%. This condition is met if $\tau_s = 2.17$ ms.
7. Using non-spiking network design rules from Szczecinski et al. (2017b), design a signal transmission pathway with a gain of $k_{syn} = 1$. For a value of $E_s = 160$, $G_{max} = 0.658$ μS.

Figure 5 shows the behavior of the signal transmission pathway designed above. The left column shows the pre-synaptic neuron activity, and the right column shows the post-synaptic neuron's response. The membrane depolarization U is plotted in the upper row for each trial, and the spiking frequency f_{sp} is plotted in the lower row. In each case, the post-synaptic neuron's spiking frequency is effectively the same as the pre-synaptic neuron's. The key result is that the process we outline above produces parameter values for the construction of neural systems to produce a particular behavior without optimization.

Let us perform another example, in which $m < 0$, to mimic the behavior of a non-spiking neuron whose membrane time constant is $\bar{\tau}_{mem} = 500$ ms. In this case, we can also validate that the transient spiking frequency is consistent with the non-spiking model's transient membrane depolarization.

1. Pick $F_{max} = 0.1$ kHz, $R = 20$ mV, and $\theta_0 = 1$ mV.
2. Pick $m = -5$, such that the spiking frequency has a transient response.
3. If $\bar{\tau}_{mem} = 500$ ms, then $\tau_\theta = 1,750$ ms.
4. Calculate $I_{bias} = 0.143$ nA.
5. Using the values from steps 1 to 4, calculate $\tau_{mem} = 700$ ms. Steps 4 and 5 ensure that $f_{sp} = 0$ when $I_{app} = 0$ and $f_{sp} = F_{max}$ when $I_{app} = R$. Neuron properties are now set.
6. Same as in the previous example, $\tau_s = 2.17$ ms.
7. Same as in the previous example, $G_{max} = 0.658$ μS.

Figure 6 shows the same type of data as **Figure 5**. The spiking frequency plots enable us to compare the smoothed transient spiking frequency to the membrane depolarization of the analogous non-spiking network. The spiking model's smoothed transient response decays at the same rate as the non-spiking model's, although some differences in the responses are visible. First, the spiking neuron's smoothed transient does not exhibit the same exponential-shaped rise as the non-spiking neuron's transient. Second, the spiking neuron's transient response to a spiking input exhibits fluctuations because the spiking threshold is continuously adapting to the instantaneous membrane voltage. In the next subsection, we show that adding more neurons can eliminate this problem.

3.5. Spiking Pathways May Introduce Unwanted Artifacts

We have shown that we can apply our non-spiking signal pathway design rules to spiking networks by treating many values as their average over time. However, there are some unintended artifacts of this approach that reduce performance. The first is an intermittent drop in a post-synaptic neuron's spiking frequency (**Figure 7A**). The way the system is tuned, the post-synaptic neuron should fire every time that the pre-synaptic neuron fires. Occasionally, however, the post-synaptic spike time is delayed relative to the synaptic current. This manifests as a temporary drop in the instantaneous spiking frequency. The prediction of the average spiking frequency is intermittently incorrect as a result.

The second artifact is a periodic instantaneous spiking frequency (about the predicted spiking frequency, **Figure 7B**). This occurs when $m < 0$, and thus the spiking threshold θ decreases when the neuron is depolarized, making it easier for the neuron to spike. Particularly at low stimulus frequencies, the neuron may exhibit a periodic spiking frequency (**Figure 7B**). However, one can see that the prediction of the average spiking

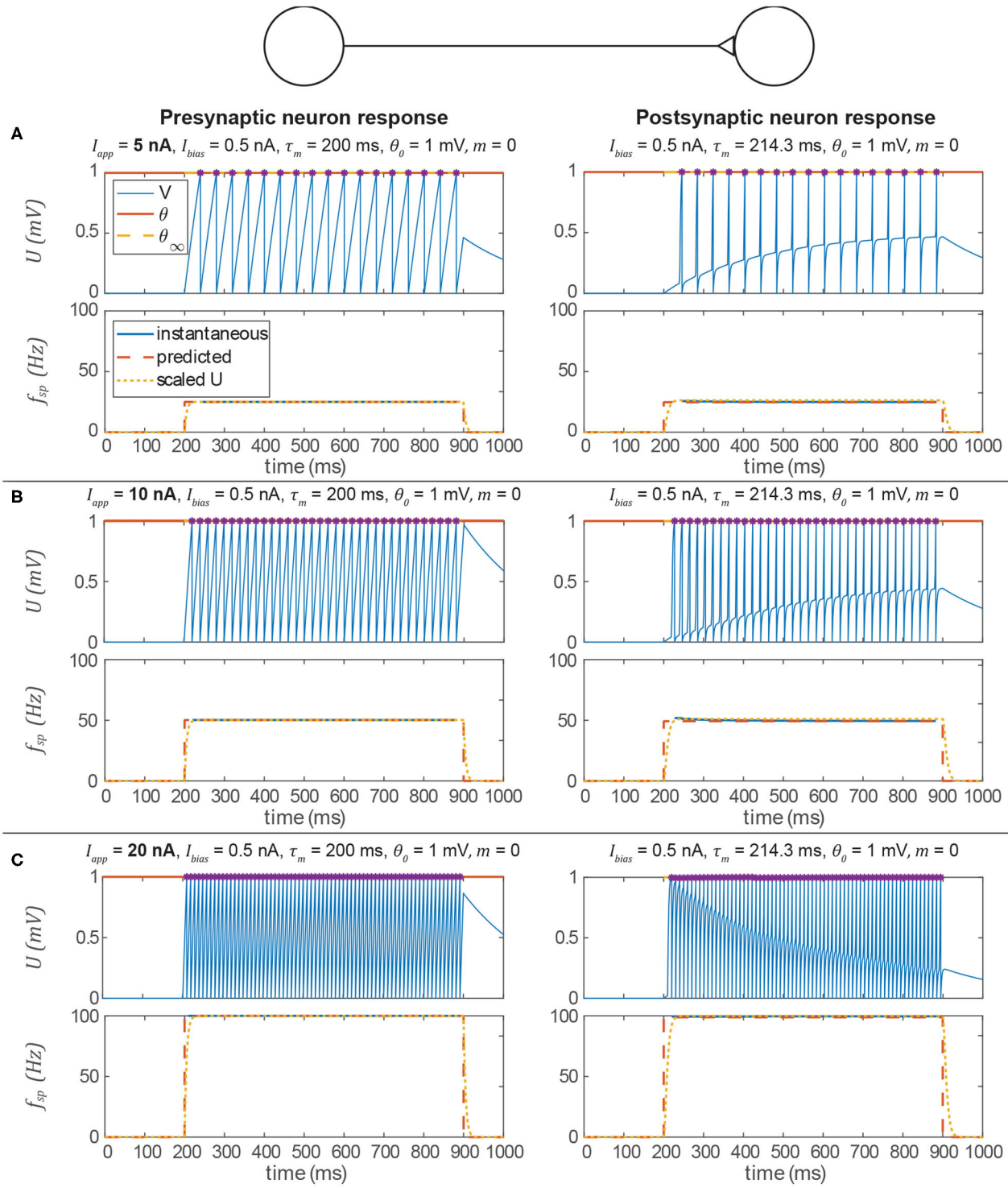


FIGURE 5 | Data from three simulations testing a signal transmission pathway where $k_{syn} = 1$ with different applied currents (I_{app}) to the pre-synaptic neuron. In each section, the upper row is the membrane voltage, and the lower row is the instantaneous spiking frequency plotted vs. time. In every case, the post-synaptic neuron's spiking frequency is the same as the pre-synaptic neuron's spiking frequency and is consistent with the prediction from the analysis above. In addition, it is approximately equal to $\bar{U}$ for the equivalent non-spiking network. Spikes are indicated by violet asterisks (no "cosmetic spikes" are plotted). **(A)** Response to a 5 nA current applied to the presynaptic neuron. **(B)** Response to a 10 nA current applied to the presynaptic neuron. **(C)** Response to a 20 nA current applied to the presynaptic neuron.

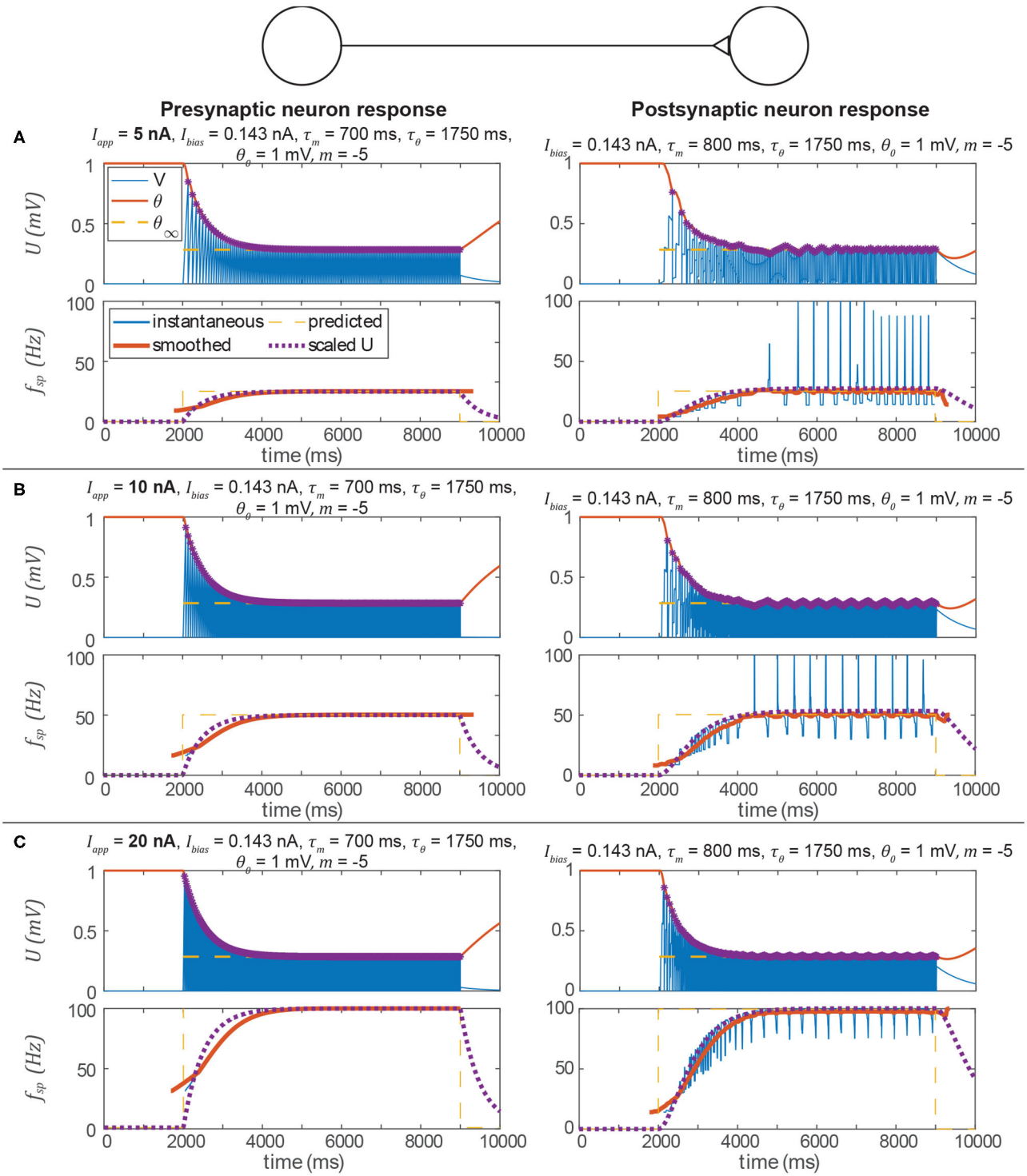


FIGURE 6 | Data from three simulations testing the signal transmission pathway. In each section, the left column is the pre-synaptic neuron's activity, the right column is the post-synaptic neuron's activity, the upper row is the membrane voltage, and the lower row is the instantaneous spiking frequency (inverse of the time between two spikes) plotted vs. time. In every case, the post-synaptic neuron's spiking frequency is the same as the pre-synaptic neuron's spiking frequency, and consistent with the prediction from the analysis above. In addition, the transient spiking frequency of each neuron follows the transient membrane voltage of the analogous non-spiking pathway. Spikes are indicated by violet asterisks (no "cosmetic spikes" are plotted). **(A)** Response to a 5 nA current applied to the presynaptic neuron. Note that the presynaptic neuron's low spiking frequency causes large fluctuations in the postsynaptic neuron's spiking frequency. **(B)** Response to a 10 nA current applied to the presynaptic neuron. Due to the higher spiking frequency, fluctuations in the postsynaptic neuron's spiking frequency appear less severe than in A. **(C)** Response to a 20 nA current applied to the presynaptic neuron. Fluctuations in the postsynaptic neuron's spiking frequency are not severe.

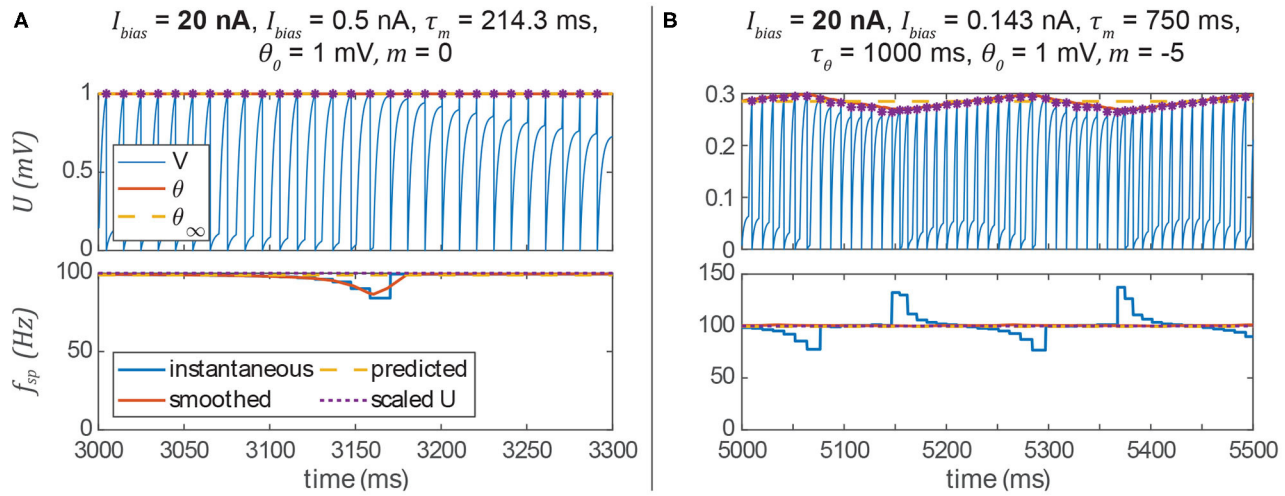


FIGURE 7 | (A) If a pre-synaptic spike does not elicit a spike in the post-synaptic neuron, then its spiking frequency will intermittently drop. **(B)** The spiking frequency of a post-synaptic neuron whose threshold hyperpolarizes in response to membrane voltage depolarization may oscillate at low frequencies. In both columns, spikes are indicated by violet asterisks (no cosmetic "spikes" are plotted).

frequency remains accurate. In the following section, we show that both of these unwanted artifacts can be eliminated by adding more neurons and synapses to the network.

3.6. A Spiking Pathway's Regularity and Accuracy Depends on the Number of Neurons in the Network

Adding more neurons to both the pre-synaptic and post-synaptic populations in a pathway helps mitigate the issues described in the previous subsection. To demonstrate this, we performed the same spiking frequency tests as before by connecting two populations of neurons together through a single pathway composed of multiple synapses. For these tests, each population has N neurons, and every neuron in each population is connected to every neuron in the second population, requiring N^2 synapses. This connectivity pattern is illustrated in **Figure 8**.

Figure 9 displays the post-synaptic population's mean spiking frequency as N increases. The same parameter values were used as in **Figure 5**, but with G_{max} randomly distributed across each neuron's N incoming synapses (uniform random distribution). For each value of N, 30 simulations were run. The raw spiking frequency over time is plotted for each. The spiking frequency fluctuates with smaller magnitude when the pathway contains more neurons. This is because each incoming synapse has a smaller maximum conductance, producing a total synaptic conductance that fluctuates less over time than when fewer synapses are present. **Figure 9B** plots the maximum and minimum spiking frequency of any one neuron in the population, normalized to the mean spiking frequency of the population. As N increases, each individual neuron's spiking frequency approaches the mean of the population. **Figure 9C** plots the mean spiking frequency of the population for each trial. As N increases, the population's spiking frequency more closely matches the intended value.

Adding more neurons to a pathway also reduces the random fluctuations in the transient response of a spiking pathway in which $m \neq 0$. **Figure 10** plots data similar to that in **Figure 9**, but for the pathway shown in **Figure 6**. Much like in **Figure 9**, adding more neurons to the signal transmission pathway makes the post-synaptic neurons fire more regularly, and with less fluctuation during the transient. As more neurons are added, the transient response more closely matches that of the equivalent non-spiking network (in black).

4. APPLICATION TO A NEUROMECHANICAL SYSTEM

The similarities between non-spiking models and a population of spiking models apply to neuromechanical control models. **Figures 11**, **12** show data from a simulation of the muscle stretch reflex illustrated in **Figure 1**. In each case, the extensor muscle was activated, causing the flexor to stretch. The system's behavior was simulated in four scenarios: (1) open loop, that is, the flexor does not activate, although it can develop passive tension; (2) closed loop, with a pathway containing a single spiking neuron at each level that mediates a stretch reflex to activate the flexor and resist the stretch imposed by the extensor; (3) closed loop, with a pathway built from populations of 10 spiking neurons per node, connected as illustrated in **Figure 8**, and (4) closed loop, with a pathway built from non-spiking neurons and synapses. Noise was added to the model via reset noise (Gerstner, 2000). The impact of such reset noise on a neuron's encoding properties is calculated in **Supplementary Materials (S1.1.6)**, and the model's formulation and parameter values are listed in **Supplementary Materials (S1.2.1)**.

Figure 11 compares the system performance when the spiking neurons have constant spiking threshold θ, i.e., $m = 0$. In this case, there is effectively no transient spiking frequency when

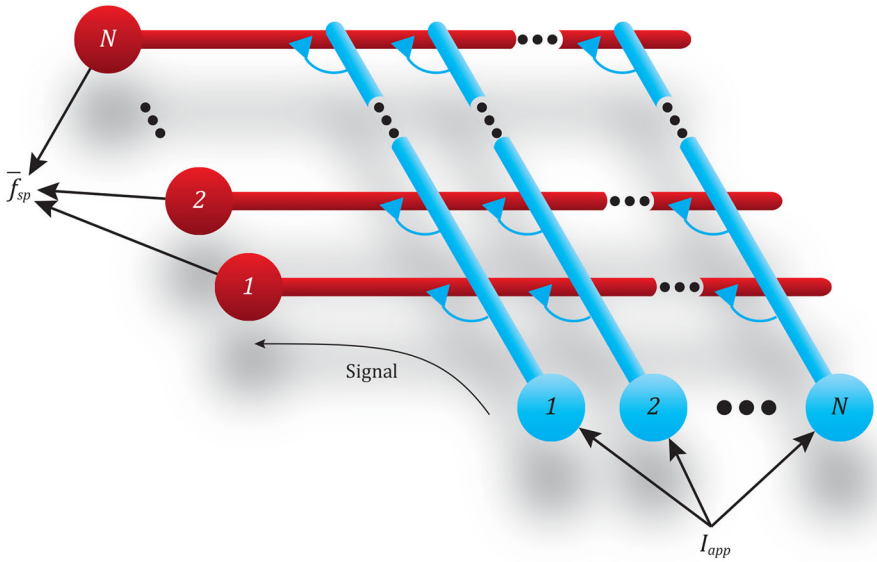


FIGURE 8 | Diagram showing the all-to-all connectivity scheme used between populations of spiking neurons. The *N* input neurons (blue) all receive the same applied current I_{app}, causing spikes that transmit this signal via N^2 synapses to the *N* output neurons (red).

a stimulus is applied based on the length of the flexor muscle (**Figure 1**). In case 1, the open loop case, contracting the extensor results in significant joint extension, ≈0.7 radian. In all three closed loop cases, the joint extends much less in response to the muscle force due to the stretch reflex. After the major motions have died out, the muscle force, and therefore the joint angle, fluctuates the most in case 2, less in case 3, and not at all in case 4. All of the key response characteristics, including the angle overshoot, the settling time, and the final angle are the same in cases 2–4, suggesting that these systems are interchangeable models of such a stretch reflex.

Figure 12 compares the system performance when the spiking neurons do not have constant spiking threshold θ, i.e., $m = -5$. Tuning the spiking and non-spiking networks to exhibit similar transient behavior (e.g., in section 3.4) will result in much longer membrane time constants for the non-spiking system. In this case one would expect delayed activation of the nodes along the reflex pathway, causing a delayed muscle membrane depolarization and resulting in more overshoot and a longer rise time than when $m = 0$. Indeed this occurs in each case 2–4 relative to the experiments in **Figure 11**.

However, the non-spiking system in case 4 exhibits damped oscillation after case 3 has effectively come to rest. The spiking pathway does not exhibit such oscillations because the transient spiking frequency is due to the threshold θ changing from its *initial* value θ_0 to the steady state value at which spikes occur θ^*_∞. **Figure 2D** illustrates this point, showing that once a neuron begins spiking, the value of θ^* is largely constant, even as the input I_{app} (and thus the spiking frequency f_{sp}) increases. This is because the average voltage U_{avg} is relatively insensitive to changes in spiking frequency (**Figure S1**), causing only small changes in θ^* and thus f_{sp}. Despite this mismatch in transient responses, the FSA enables us to rapidly construct models that contain both spiking and non-spiking elements.

5. DISCUSSION

5.1. Summary

The analysis and numerical results in this manuscript show how continuous, non-spiking leaky-integrator neural dynamics can approximate the dynamics of a population of identical GLIF spiking neurons with randomized interconnections. The parallels in encoding and information transfer manifest through three analogs: (1) A spiking population's mean spiking frequency is analogous to the membrane voltage of a leaky integrator, (2) the transient spiking frequency of a spiking neuron can mimic a non-spiking neuron's transient voltage, and (3) a spiking synapse's average conductance is proportional to the pre-synaptic neuron's spiking frequency, analogous to how a non-spiking synapse's conductance is proportional to the pre-synaptic neuron's membrane depolarization. Since the dynamics are approximately similar, a network built from either type can be used to encode and transfer information in an equivalent manner. Therefore, networks of either type can have similar overall behavior, and either type might effectively be used to model and understand the nervous system.

The parallels between non-spiking neural models and models of populations of spiking neurons have been known for quite some time (Wilson and Cowan, 1972). However, the process and tools needed to set parameters within networks of these models to achieve desired and/or equivalent behavior have been lacking. In an attempt to apply the classical analysis from (Wilson and Cowan, 1972) to the practical application of programming neuromorphic hardware, we extended our functional subnetwork approach (FSA) to tuning networks of GLIF neurons and synapses. This extension enables one to tune control systems built from either non-spiking nodes or populations of spiking neurons. We presented a step-by-step

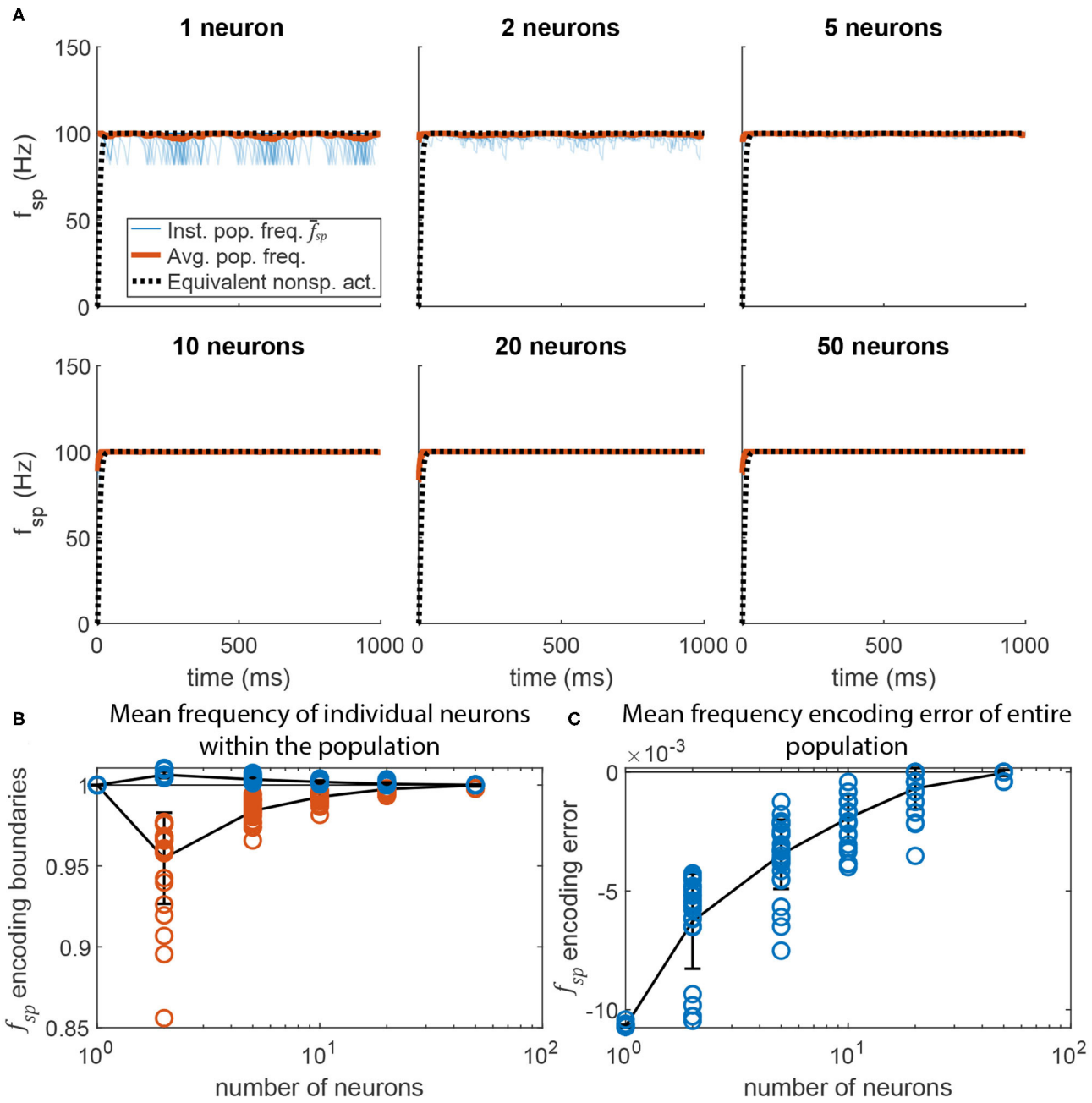


FIGURE 9 | Adding more neurons to the pre-synaptic and post-synaptic populations increases encoding accuracy and reduces activation variation. **(A)** Average spiking frequency of the post-synaptic population $\bar{f}_{sp}$ over time when the spiking threshold θ is constant (i.e., $m = 0$) for all neurons. Thirty individual trials (blue), the mean over time (red), and the equivalent non-spiking pathway (black dotted) are plotted. **(B)** The highest (blue circles) and lowest (red circles) mean frequency *of a single neuron* in the population. Line plots the mean of each group, error bars are ±1 standard deviation. **(C)** The error between the mean spiking frequency of the entire population and the intended spiking frequency for each trial (blue circles). Line plots the mean, error bars are ±1 standard deviation.

method for tuning practical networks of populations of spiking neurons and synapses. We provided examples showing how increasing the number of neurons makes data transmission more ideal (i.e., match the expected population average activity). Finally, we provided a practical example of how such a method can be used to tune a neuromechanical model for control, and how the non-spiking and spiking implementations compare and contrast.

Despite the similarities between the models' activation in response to inputs (section 3.1) and how this activation maps to synaptic conductivity (section 3.3), we observe differences in the models' transient responses. A spiking neuron's smoothed

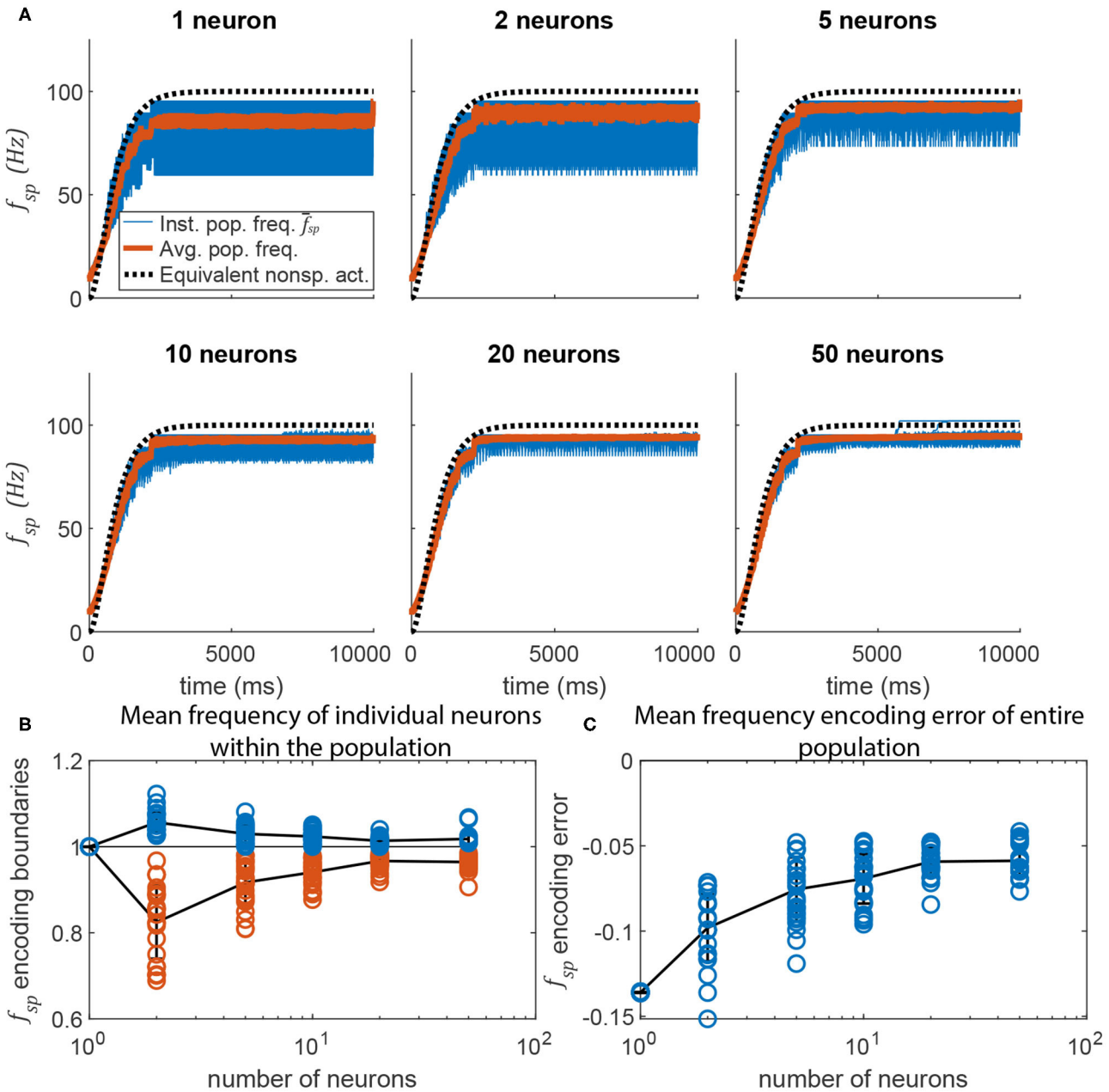


FIGURE 10 | Adding more neurons to the pre-synaptic and post-synaptic populations increases encoding accuracy and reduces activation variation. **(A)** Average spiking frequency of the post-synaptic population $\bar{f}_{sp}$ over time when the spiking threshold θ is variable (i.e., $m = -5$) for all neurons. Thirty individual trials (blue), the mean over time (red), and the equivalent non-spiking pathway (black dotted) are plotted. **(B)** The highest (blue circles) and lowest (red circles) mean frequency of a single neuron in the population. Line plots the mean, error bars are ± 1 standard deviation. **(C)** The error between the mean spiking frequency of the entire population and the intended spiking frequency for each trial is plotted (blue circles). Line plots the mean, error bars are ± 1 standard deviation.

(i.e., time-averaged) transient spiking frequency is a good match for a non-spiking neuron's transient membrane voltage if the spiking neuron is not initially spiking (**Figures 6**, **10**). This is because the spiking threshold must change from the initial value θ_0 to the steady state value when a spike occurs, θ^*_∞. However, the spike-time threshold θ^*_∞ is insensitive to a neuron's input current (and therefore the neuron's spiking frequency), meaning that the amplitude of the transient is very small once a neuron is spiking. This is not true for the non-spiking model, whose transient amplitude depends strongly on the input current. Therefore, the response properties of networks tuned to have long, exaggerated transient responses are a good match initially, but not once a neuron is already spiking (**Figure 12**). In all other

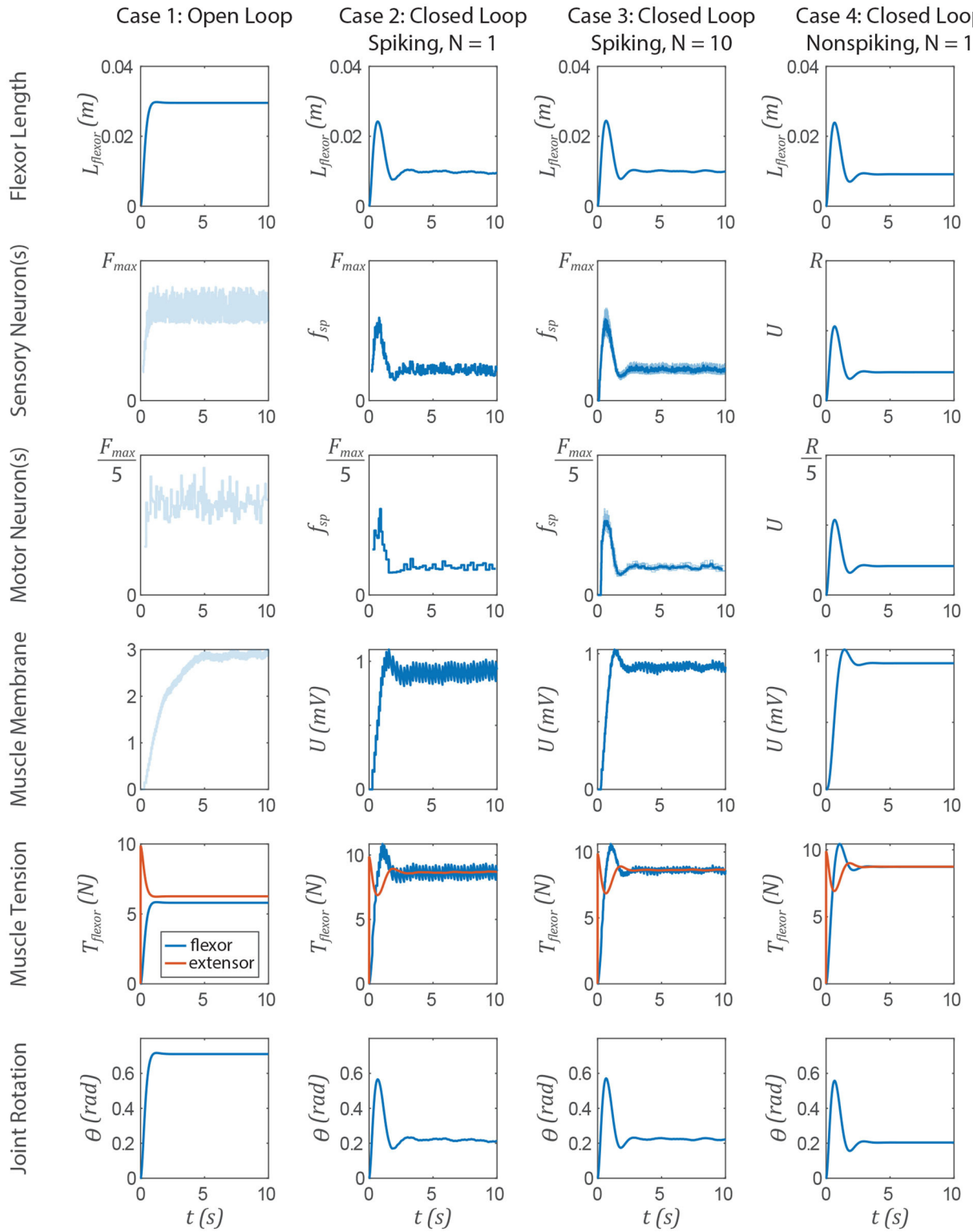


FIGURE 11 | Data from a simulation of the system in **Figure 1**. Each column shows data from a different configuration of the system, from left to right: Case 1, open loop; case 2, spiking reflex pathway, $N = 1$; case 3, spiking reflex pathway, $N = 10$ (f_{sp} for every neuron is shown in light blue, population mean f_{sp} is plotted in dark blue); case 4, non-spiking reflex pathway. All corresponding axes are scaled the same. Each row plots data from a different stage of the reflex loop as depicted in **Figure 1**. In this figure, all spiking neurons have a constant spiking threshold θ (i.e., $m = 0$).

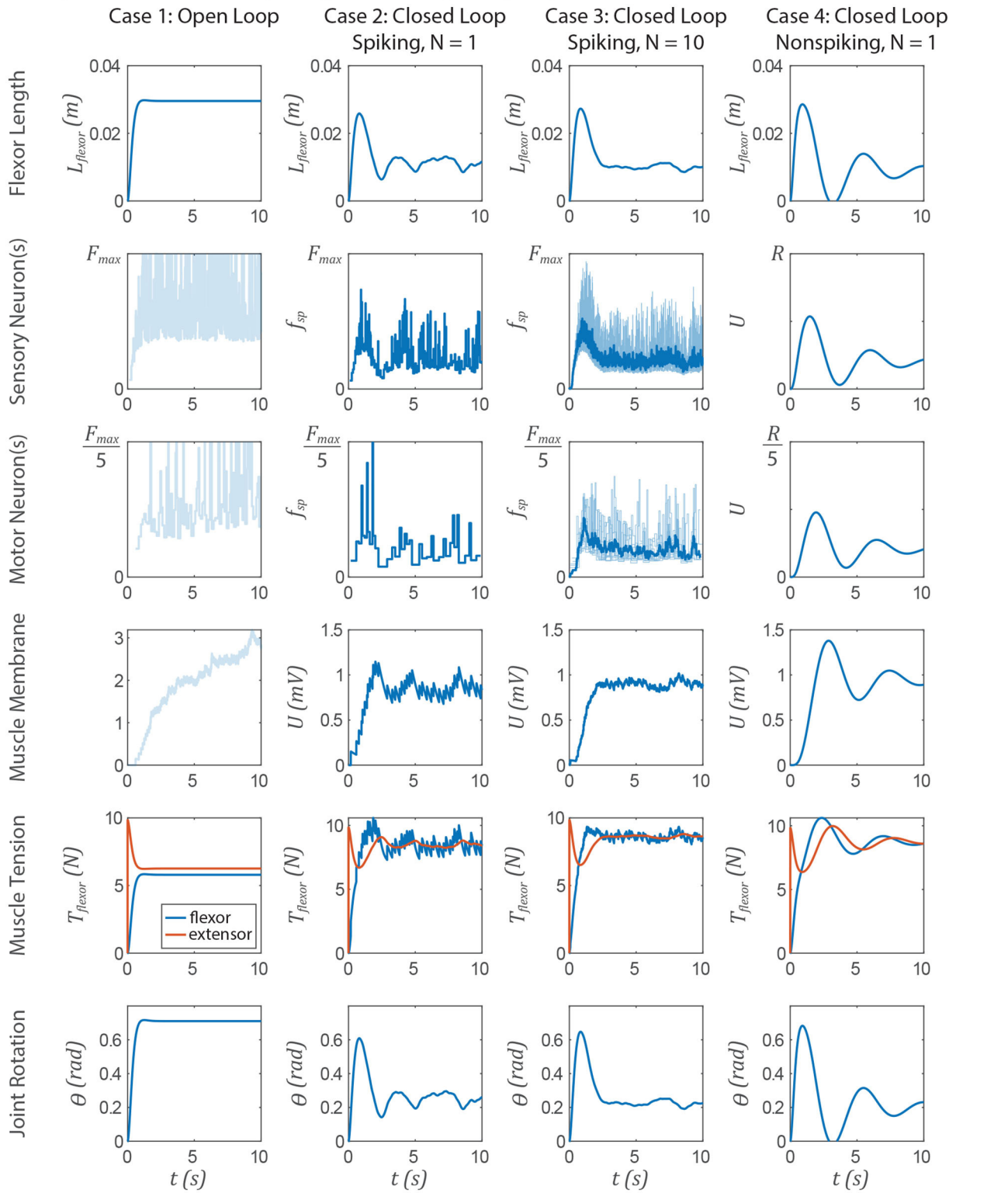


FIGURE 12 | Data from a simulation of the system in **Figure 1**. Each column shows data from a different configuration of the system, from left to right: Case 1, open loop; case 2, spiking reflex pathway, $N = 1$; case 3, spiking reflex pathway, $N = 10$ (f_{sp} for every neuron is shown in light blue, population mean f_{sp} is plotted in dark blue); case 4, non-spiking reflex pathway. All corresponding axes are scaled the same. Each row plots data from a different stage of the reflex loop as depicted in **Figure 1**. In this figure, all spiking neurons have a variable spiking threshold θ (i.e., $m = -5$).

respects, however, these models can be tuned to produce the same responses.

5.2. Expanding These Methods

The analysis in this manuscript primarily provides a framework for designing rate-coding networks, based on their steady-state spiking frequency. We have already shown how steady-state analysis contributes to designing arithmetic and dynamic (i.e., differentiation and integration over time) networks (Szczecinski et al., 2017b). However, not all neural computation is rate-coding, meaning that additional techniques are needed to expand this work to engineer direct encoding of other signals to produce more sophisticated neural behaviors. For instance, a non-spiking model captures class II excitability, in which a neuron's spiking frequency exponentially approaches a steady-state spiking frequency when subjected to a step input (Mihalaş and Niebur, 2009). However, the GLIF spiking neuron model used in this work can also exhibit class I excitability in which there is no transient spiking frequency, a behavior that a non-spiking neuron cannot replicate. Such a response might be useful in a reflex pathway, in which delayed sensory feedback may destabilize the system. In addition, a spiking neuron can exhibit phasic excitability, in which its spiking frequency starts high, but decays to 0 during a step input (Mihalaş and Niebur, 2009, see also **Figure 3C**). In the future, we plan to investigate whether such phasic responses could be used to replace the differentiation network from the non-spiking FSA approach (Szczecinski et al., 2017b) with a single neuron, a technique we have used in the past, but did not characterize thoroughly (Szczecinski et al., 2014). Exploiting single-neuron properties in this way could enable designers to pack more computational capability into chips with limited (albeit large) network sizes, such as Loihi (Davies et al., 2018).

However, creating small networks that seek to exploit single-neuron properties may reduce the accuracy of encoding, decoding, and other operations within the network. Alternative systems, such as the Neuroengineering Framework (NEF) (Eliasmith and Anderson, 2002), rely on nodes (ensembles) with many neurons (on the order of 10–1,000) to accurately encode information into the network. What makes NEF so powerful is the relatively hands-off design process, wherein the user specifies the intended function and number of neurons per node, and then NEF learns the intra-node parameter values necessary to perform that function (Eliasmith et al., 2012; Bekolay et al., 2014). This approach is much less onerous to the designer than the FSA, which requires explicit tuning of parameters for encoding, decoding, and other functions. We anticipate that these two approaches may complement one another, wherein the direct network tuning accomplished by the FSA could be used to initialize tuning within the NEF. In our experience, the FSA can be used to select initial network parameters that aid subsequent parameter optimization (Pickard et al., 2020). As a next step, we plan to compare the accuracy and efficiency of networks tuned via the FSA with those tuned via the NEF. We expect that the NEF may achieve arbitrarily high accuracy, but possibly at a computational cost. The FSA could be used to initialize learning networks in a less random way, requiring fewer neurons per node and less time to train.

5.3. When to Use Spiking or Non-spiking Neurons

A question that follows from this work is that if non-spiking and spiking neuron dynamics have many parallels, how does a modeler choose to use one or the other type? We believe that both types are useful in different contexts, depending on the knowledge available about the system to be modeled, the research question addressed by the model, and the real-world application of the network (e.g., in robotics).

A natural choice is to model spiking neurons in the nervous system with spiking models, and non-spiking neurons with non-spiking models. However, biomechanical constraints determine whether animals use spiking or non-spiking neurons. Specifically, action potentials can be transmitted over long distances, whereas graded (i.e., non-spiking) signals tend to dissipate over even short distances. This may be why many non-spiking neurons have been identified in insects and other small animals, particularly for integrating sensory information (Burrows et al., 1988; Sauer et al., 1996); they have less room in their bodies to house networks of many spiking neurons, and their small bodies do not require them to transmit information over long distances. No matter why animals have spiking or non-spiking neurons, a computer model does not share these biochemical constraints, so it is worth deciding how to model networks based on the computational hardware available.

One purely technological motivation to use spiking models is for model implementation on neuromorphic hardware. Chips, such as Loihi (Davies et al., 2018), SpiNNaker (Khan et al., 2008), TrueNorth (Merolla et al., 2014), and others (Pfeil et al., 2013; Gehlhaar, 2014; Ionica and Gregg, 2015) use non-traditional architecture to simulate hundreds of thousands of spiking neurons and hundreds of millions of synapses in real-time while using on the order of one watt of power. Neuromorphic computers tend to use spiking models because they have less communication overhead than non-spiking networks. For spiking networks, communication can be binary (1 bit per spike) and only must occur after a spike occurs, at a maximum of 200–300 Hz (Gerstner et al., 1997; Carter and Bean, 2010) but more often below 1–2 Hz (Kerr et al., 2005). In contrast, non-spiking synapses need to be updated during every simulation step, because they depend on the pre-synaptic neuron's continuous membrane voltage. Such a requirement significantly increases overhead relative to spiking models.

Spiking neurons and synapses are also attractive because they enable the use of spike timing dependent plasticity (STDP) learning techniques (Gerstner et al., 1996; Markram et al., 1997, 2012), a powerful class of machine learning tools. These methods have been applied to many stimulus-recognition tasks, such as hearing, speech, and vision. They measure the coincidence of incoming spikes to increase or decrease the strength of synapses in the network, and thus rely on spiking models. These networks are typically classified as "self-organizing," meaning that they initially have no structure, but develop their own structure as

connections are pruned due to disuse. However, many parts of the nervous system have exquisite structure, and lend themselves to being directly modeled, structure, and all. Thus, we believe that the methods in this manuscript may serve to produce baseline parameter values for a highly structured network, which may then use spike-based mechanisms to tune itself over time, a technique like that utilized in Nengo (Bekolay et al., 2014). In such cases, populations of spiking neurons would be preferable to non-spiking population models, but could be initialized using the tuning rules presented in this manuscript.

Additionally, spiking neurons and synapses may be beneficial because even a simple model like that used in this work can produce a wide variety of behaviors and responses (Mihalaş and Niebur, 2009). For example, setting $m > 0$ can produce spike frequency adaptation and phasic spiking, which are known to be critical for filtering sensory feedback in locomotory systems (Mamiya et al., 2018; Zill et al., 2018). Such rate-sensitive responses can be produced by small networks of non-spiking neurons and synapses (Szczecinski et al., 2017b), but force the modeler to use more neurons than may be necessary. Therefore, if the modeler knows that single neurons in their model system generate more complex responses than class I or II excitability, then spiking models should be utilized. However, if the neuron responses in the network are simple, then the model could be implemented as a network of non-spiking neurons instead.

We believe non-spiking networks may be particularly beneficial if a model is not meant to run on specialized neuromorphic hardware. Simulating the membrane voltage of each spiking neuron in a population requires storing and updating orders of magnitude more states than simply using a non-spiking node to represent the mean activity of the population. In addition, throughout this study, we observed that the timing of spikes was sensitive to the simulation step used (i.e., spikes cannot happen between time steps), and simulations only closely matched our analytic predictions as the time step became very small. We also tested adaptive stepping algorithms (Matlab's ode45 and ode15s solvers); however, the discontinuous nature of spikes required the use of event functions that halt simulation whenever a spike occurs, complicating the code. In general, we found that it took longer to simulate the dynamics of a spiking network relative to those of a non-spiking network. These reasons motivate implementing networks on traditional computers with non-spiking models.

Finally, non-spiking neurons may contribute to models by representing neuromodulatory neurons that cause large-scale changes to network behavior. In Szczecinski et al. (2017b), we not only designed "signal transmission" pathways as in this work, but also "signal modulation" pathways, in which one neuron could modify the gain of another neuron's response to its synaptic inputs. When we tested signal modulation pathways built from spiking neurons and synapses, the results were poor (data not shown). Due to the discrete nature of spikes, modulation was inconsistent, leading to unpredictably varying firing frequencies from the "modulated" neuron. However, it may be advantageous to instead construct networks in which signals are transmitted via spiking neurons, but modulated via non-spiking neurons. These non-spiking neurons in effect model neuromodulatory neurons, whose voltage reflects the concentration of a neuromodulator in the network, and whose non-spiking synapses represent the activation of receptors sensitive to that particular neuromodulator. Such pathways may change the resting potential, membrane conductance, and time constant of other neurons throughout the network (for a review, see Miles and Sillar, 2011). Such non-spiking neurons would have long time constants, enabling them to modify network performance on much longer timescales than that of a single spike. Such neurons could receive either descending input from the brain, or ascending input from local sensory neurons. The result would be a model that can modulate its control system based on exteroception and interoception, and exhibit truly adaptive, context-dependent behavior.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

NS conceived of the study. NS, RQ, and AH planned the results to be collected. NS collected results and wrote the manuscript. RQ and AH edited the manuscript. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported by NSF: NeuroNex Grant #2015317 to RQ, AH, and NS funds research into creating computational and robotic models of animal motor control, NSF: RI Grant #1704436 to RQ funds research into tuning dynamical neural controllers for legged robots, and NSF: US-German Collaboration Grant #1608111 to RQ funds research into tuning dynamical neural controllers for legged robots.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2020.577804/full#supplementary-material

REFERENCES

Ayers, J., and Crisman, J. (1993). "Lobster walking as a model for an omnidirectional robotic ambulation architecture," in *Proceedings of the Workshop on "Locomotion Control in Legged Invertebrates" on Biological Neural Networks in Invertebrate Neuroethology and Robotics* (San Diego, CA: Academic Press Professional, Inc.), 287–316.

Ayers, J., Rulkov, N., Knudsen, D., Kim, Y. B., Volkovskii, A., and Selverston, A. (2010). Controlling underwater robots with electronic nervous systems. *Appl. Bionics Biomech*. 7, 57–67. doi: 10.1155/2010/578604

Beer, R. D., and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adapt. Behav*. 1, 91–122. doi: 10.1177/105971239200100105

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., et al. (2014). Nengo: a Python tool for building large-scale functional brain models. *Front. Neuroinform*. 7:48. doi: 10.3389/fninf.2013.00048

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J. M., et al. (2014). Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. *Proc. IEEE* 102, 699–716. doi: 10.1109/JPROC.2014.2313565

Berg, E. M., Hooper, S. L., Schmidt, J., and Büschges, A. (2015). A leg-local neural mechanism mediates the decision to search in stick insects. *Curr. Biol*. 25, 2012–2017. doi: 10.1016/j.cub.2015.06.017

Blouw, P., Choo, X., Hunsberger, E., and Eliasmith, C. (2018). *Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware*.

Brunel, N., and van Rossum, M. C. W. (2007). Quantitative investigations of electrical nerve excitation treated as polarization. *Biol. Cybernet*. 97, 341–349. doi: 10.1007/s00422-007-0189-6

Bueschges, A., Kittmann, R., and Schmitz, J. (1994). Identified nonspiking interneurons in leg reflexes and during walking in the stick insect. *J. Compar. Physiol. A* 174, 685–700. doi: 10.1007/BF00192718

Burrows, M., Laurent, G., and Field, L. (1988). Proprioceptive inputs to nonspiking local interneurons contribute to local reflexes of a locust hindleg. *J. Neurosci*. 8, 3085–3093. doi: 10.1523/JNEUROSCI.08-08-03085.1988

Carter, B. C., and Bean, B. P. (2010). Incomplete inactivation and rapid recovery of voltage-dependent sodium channels during high-frequency firing in cerebellar purkinje neurons. *J. Neurophysiol*. 105, 860–871. doi: 10.1152/jn.01056.2010

Dasgupta, S., Goldschmidt, D., Wörgötter, F., and Manoonpong, P. (2015). Distributed recurrent neural forward models with synaptic adaptation and CPG-based control for complex behaviors of walking robots. *Front. Neurorobot*. 9:10. doi: 10.3389/fnbot.2015.00010

Davies, M., Srinivasa, N., Lin, T. H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

Dürr, V., Arena, P. P., Cruse, H., Dallmann, C. J., Drimus, A., Hoinville, T., et al. (2019). Integrative biomimetics of autonomous hexapedal locomotion. *Front. Neurorobot*. 13:88. doi: 10.3389/fnbot.2019.00088

Eliasmith, C. (2013). *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford: Oxford University Press.

Eliasmith, C., and Anderson, C. H. (2002). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA; London: MIT Press.

Eliasmith, C., and Anderson, C. H. (2003). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems. Computational Neuroscience. A Bradford Book*. Cambridge, MA: MIT Press

Eliasmith, C., Stewart, T. C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., et al. (2012). A large-scale model of the functioning brain. *Science* 338, 1202–1205. doi: 10.1126/science.1225266

Floreano, D., Ijspeert, A., and Schaal, S. (2014). Robotics and neuroscience. *Curr. Biol*. 24, R910–R920. doi: 10.1016/j.cub.2014.07.058

Gehlhaar, J. (2014). Neuromorphic processing: a new frontier in scaling computer architecture. *ACM SIGPLAN Not*. 49, 317–318. doi: 10.1145/2644865.2564710

Gerstner, W. (2000). Population dynamics of spiking neurons: fast transients, asynchronous states, and locking. *Neural Comput*. 12, 43–89. doi: 10.1162/089976600300015899

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature* 383, 76–78. doi: 10.1038/383076a0

Gerstner, W., Kreiter, A. K., Markram, H., and Herz, A. V. M. (1997). Neural codes: firing rates and beyond. *Proc. Natl. Acad. Sci. U.S.A*. 94, 12740–12741. doi: 10.1073/pnas.94.24.12740

Hilts, W. W., Szczecinski, N. S., Quinn, R. D., and Hunt, A. J. (2019). A dynamic neural network designed using analytical methods produces dynamic control properties similar to an analogous classical controller. *IEEE Control Syst. Lett*. 3, 320–325. doi: 10.1109/LCSYS.2018.2871126

Hunt, A., Szczecinski, N., and Quinn, R. (2017). Development and training of a neural controller for hind leg walking in a dog robot. *Front. Neurorobot*. 11:18. doi: 10.3389/fnbot.2017.00018

Ionica, M. H., and Gregg, D. (2015). The movidius myriad architecture's potential for scientific computing. *IEEE Micro* 35, 6–14. doi: 10.1109/MM.2015.4

Kerr, J. N. D., Greenberg, D., and Helmchen, F. (2005). Imaging input and output of neocortical networks *in vivo*. *Proc. Natl. Acad. Sci. U.S.A*. 102, 14063–14068. doi: 10.1073/pnas.0506029102

Khan, M. M., Lester, D. R., Plana, L. A., Rast, A., Jin, X., Painkras, E., et al. (2008). "Spinnaker: mapping neural networks onto a massively-parallel chip multiprocessor," in *IEEE International Joint Conference on Neural Networks, 2008. IJCNN 2008 (IEEE World Congress on Computational Intelligence)* (Hong Kong: IEEE), 2849–2856. doi: 10.1109/IJCNN.2008.4634199

Maass, W., and Markram, H. (2004). On the computational power of circuits of spiking neurons. *J. Comput. Syst. Sci*. 69, 593–616. doi: 10.1016/j.jcss.2004.04.001

Mamiya, A., Gurung, P., and Tuthill, J. C. (2018). Neural coding of leg proprioception in Drosophila. *Neuron* 100, 636–650. doi: 10.1016/j.neuron.2018.09.009

Markram, H., Gerstner, W., and Sjöström, P. J. (2012). Spike-timing-dependent plasticity: a comprehensive overview. *Front. Synap. Neurosci*. 4, 2010–2012. doi: 10.3389/fnsyn.2012.00002

Markram, H., Luebke, J., Frotscher, M., and Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science* 275, 213–215. doi: 10.1126/science.275.5297.213

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 668–673. doi: 10.1126/science.1254642

Mihalaş, Ş., and Niebur, E. (2009). A generalized linear integrate-and-fire neural model produces diverse spiking behaviors. *Neural Comput*. 21, 704–718. doi: 10.1162/neco.2008.12-07-680

Miles, G. B., and Sillar, K. T. (2011). Neuromodulation of vertebrate locomotor control networks. *Physiology* 26, 393–411. doi: 10.1152/physiol.00013.2011

Nourse, W., Quinn, R. D., and Szczecinski, N. S. (2018). "An adaptive frequency central pattern generator for synthetic nervous systems," in *Conference on Biomimetic and Biohybrid Systems* (Paris: Springer), 361–364. doi: 10.1007/978-3-319-95972-6_38

Pfeil, T., Grübl, A., Jeltsch, S., Müller, E., Müller, P., Petrovici, M. A., et al. (2013). Six networks on a universal neuromorphic computing substrate. *Front. Neurosci*. 7:11. doi: 10.3389/fnins.2013.00011

Pickard, S. C., Quinn, R. D., and Szczecinski, N. S. (2020). A dynamical model exploring sensory integration in the insect central complex substructures. *Bioinspir. Biomimet*. 15:026003. doi: 10.1088/1748-3190/ab57b6

Sauer, A. E., Driesang, R. B., Büschges, A., Bässler, U., and Borst, A. (1996). Distributed processing on the basis of parallel and antagonistic pathways simulation of the femur-tibia control system in the stick insect. *J. Comput. Neurosci*. 3, 179–198. doi: 10.1007/BF00161131

Szczecinski, N. S., Brown, A. E., Bender, J. A., Quinn, R. D., and Ritzmann, R. E. (2014). A neuromechanical simulation of insect walking and transition to turning of the cockroach *Blaberus discoidalis*. *Biol. Cybernet*. 108, 1–21. doi: 10.1007/s00422-013-0573-3

Szczecinski, N. S., Hunt, A. J., and Quinn, R. (2017a). Design process and tools for dynamic neuromechanical models and robot controllers. *Biol. Cybernet*. 111, 105–127. doi: 10.1007/s00422-017-0711-4

Szczecinski, N. S., Hunt, A. J., and Quinn, R. D. (2017b). A functional subnetwork approach to designing synthetic nervous systems that control legged robot locomotion. *Front. Neurorobot*. 11:37. doi: 10.3389/fnbot.2017.00037

Szczecinski, N. S., and Quinn, R. D. (2017a). Leg-local neural mechanisms for searching and learning enhance robotic locomotion. *Biol. Cybernet*. 112, 99–112. doi: 10.1007/s00422-017-0726-x

Szczecinski, N. S., and Quinn, R. D. (2017b). "Mantisbot changes stepping speed by entraining cpgs to positive velocity feedback," in *Conference*

on Biomimetic and Biohybrid Systems (Stanford, CA: Springer), 440–452 doi: 10.1007/978-3-319-63537-8_37

Wilson, H. R., and Cowan, J. D. (1972). Excitatory and inhibitory interactions in localized populations of model neurons. *Biophys. J.* 12, 1–24. doi: 10.1016/S0006-3495(72)86068-5

Zill, S. N., Dallmann, C. J., Büschges, A., Chaudhry, S., and Schmitz, J. (2018). Force dynamics and synergist muscle activation in stick insects: the effects of using joint torques as mechanical stimuli. *J. Neurophysiol.* 120, 1807–1823. doi: 10.1152/jn.00371.2018

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 26 November 2020
doi: 10.3389/fnbot.2020.589532

Robust Trajectory Generation for Robotic Control on the Neuromorphic Research Chip Loihi

Carlo Michaelis, Andrew B. Lehr and Christian Tetzlaff*

Department of Computational Neuroscience, University of Göttingen, Göttingen, Germany

Neuromorphic hardware has several promising advantages compared to von Neumann architectures and is highly interesting for robot control. However, despite the high speed and energy efficiency of neuromorphic computing, algorithms utilizing this hardware in control scenarios are still rare. One problem is the transition from fast spiking activity on the hardware, which acts on a timescale of a few milliseconds, to a control-relevant timescale on the order of hundreds of milliseconds. Another problem is the execution of complex trajectories, which requires spiking activity to contain sufficient variability, while at the same time, for reliable performance, network dynamics must be adequately robust against noise. In this study we exploit a recently developed biologically-inspired spiking neural network model, the so-called anisotropic network. We identified and transferred the core principles of the anisotropic network to neuromorphic hardware using Intel's neuromorphic research chip Loihi and validated the system on trajectories from a motor-control task performed by a robot arm. We developed a network architecture including the anisotropic network and a pooling layer which allows fast spike read-out from the chip and performs an inherent regularization. With this, we show that the anisotropic network on Loihi reliably encodes sequential patterns of neural activity, each representing a robotic action, and that the patterns allow the generation of multidimensional trajectories on control-relevant timescales. Taken together, our study presents a new algorithm that allows the generation of complex robotic movements as a building block for robotic control using state of the art neuromorphic hardware.

Keywords: robot control, neuromorphic computing, Loihi, anisotropic network, spiking neural network, computational neuroscience

OPEN ACCESS

Edited by:
Jorg Conradt,
Royal Institute of Technology, Sweden

Reviewed by:
Alejandro Linares-Barranco,
Sevilla University, Spain
Terrence C. Stewart,
National Research Council Canada (NRC-CNRC), Canada

****Correspondence:***
Carlo Michaelis
carlo.michaelis@phys.uni-goettingen.de

Received: *30 July 2020*
Accepted: *28 October 2020*
Published: *26 November 2020*

Citation:
Michaelis C, Lehr AB and Tetzlaff C (2020) Robust Trajectory Generation for Robotic Control on the Neuromorphic Research Chip Loihi. Front. Neurorobot. 14:589532. doi: 10.3389/fnbot.2020.589532

1. INTRODUCTION

During infancy, humans acquire fine motor control, allowing flexible interaction with real world objects. For example, most humans can effortlessly grasp a glass of water, despite variations in object shape and surroundings. However, achieving this level of flexibility in artificial autonomous systems is a difficult problem. To accomplish this, such a system must accurately classify inputs and take appropriate actions under noisy conditions. Thus, increasing robustness to input noise is crucial for the development of reliable autonomous systems (Khalastchi et al., 2011; Naseer et al., 2018).

Neuromorphic hardware is based on highly parallel bio-inspired computing, which employs decentralized neuron-like computational units. Instead of the classical separation of processing and memory, on neuromorphic hardware information is both processed and stored in a network of these computational units. Neuromorphic architectures offer faster and more energy-efficient

computation than traditional CPUs or GPUs (Blouw et al., 2019; Tang et al., 2019), which is a vital feature for autonomous systems. However, porting existing robot control algorithms (e.g., Ijspeert et al., 2002) to neuromorphic hardware is *per se* ambitious (but see Eliasmith and Anderson, 2004; DeWolf et al., 2016; Voelker and Eliasmith, 2017) and difficult to optimize to the specific hardware architecture. At the same time, the development of new algorithms is also challenging due to the decentralized design principle of neuromorphic hardware as a network of computational units (Lee et al., 2018).

The basic network type for the various neuromorphic architectures developed in recent years (Schemmel et al., 2010; Furber et al., 2014; Davies et al., 2018; Neckar et al., 2018) are spiking neural networks (SNNs), coined third generation neural networks (for review, see Maass, 1997; Tavanaei et al., 2019). In particular, the reservoir computing paradigm, such as echo state networks (Jaeger, 2001, 2007) or liquid state machines (Maass et al., 2002), often serves as an algorithmic basis. In reservoir computing a randomly connected SNN provides a "reservoir" of diverse computations, which can be exploited by training weights from the reservoir units to additional units that constitute time-dependent outputs of the system.

The internal dynamics of the reservoir or SNN generally provide a sufficient level of variability such that arbitrary output functions on a control-relevant timescale can be read out. However, the system fails if the input is noisy or perturbations arise while the trajectory is being performed (Maass et al., 2002; Sussillo and Abbott, 2009; Laje and Buonomano, 2013; Hennequin et al., 2014). That is to say, spiking dynamics in SNNs are often unstable, meaning that small changes in the initial conditions result in different spiking patterns (Sompolinsky et al., 1988; Van Vreeswijk and Sompolinsky, 1996; Brunel, 2000; London et al., 2010). Thus, when an output is trained using such a spiking pattern, low levels of noise lead to a deviation of the estimated output from the target output and stable trajectories can only be obtained on a timescale of milliseconds. On the other hand, attractor dynamics provide highly stable, persistent activity (Amit, 1992; Tsodyks, 1999); however, they tend to lack the variability in the spiking dynamics required for complex output learning (Nachstedt and Tetzlaff, 2017). This implies a stability-variability trade-off, also denoted as a robustness-flexibility trade-off (Pehlevan et al., 2018).

A number of approaches have been developed in recent years to stabilize the spiking dynamics of SNNs while retaining sufficient variability for output learning (Laje and Buonomano, 2013; Hennequin et al., 2014; Pehlevan et al., 2018; Vincent-Lamarre et al., 2020). To improve stability, recent approaches used feed-forward structures (Pehlevan et al., 2018) or employed supervised learning rules (Laje and Buonomano, 2013). While feed-forward structures provide stable activity patterns, in general these play out on a very fast timescale (Zheng and Triesch, 2014) or require neural/synaptic adaptation such that activity moves between neuron groups (York and Van Rossum, 2009; Itskov et al., 2011; Murray et al., 2017; Maes et al., 2020). And since for supervised learning all states in the network need to be accessible at each computing unit, these so-called global learning rules are not compatible with most neuromorphic hardware.

Thus, achieving stable activity patterns on a control-relevant timescale in a network architecture and learning regime capable of running on neuromorphic hardware remains an open problem. Necessary criteria are that (1) learning or adaptation mechanisms in the SNN should be local to individual synapses, or synapses should be static, (2) sequential activity patterns should remain active for hundreds of milliseconds, (3) spike patterns should contain sufficient variability for arbitrary output learning, and (4) the network should possess noise-robust neuronal dynamics. Meeting these criteria is especially difficult for recurrent network structures, like reservoir networks. However, the so-called anisotropic network model appears to be a promising candidate (Spreizer et al., 2019). The model is based on a biologically-inspired rule for forming spatially asymmetric non-plastic connections. Thus, synapses are static, meeting the first criterion, and the timescale of activity sequences is on the order of tens to hundreds of milliseconds, fulfilling the second criterion. However, whether the model also fulfills the third and fourth criteria, sufficient variability and stability under input noise, has not yet been assessed.

In this paper we use the anisotropic network as a building block for a novel algorithm yielding robust robotic control. We implement the network architecture on Kapoho Bay, a neuromorphic hardware system from Intel containing two Loihi chips (Davies et al., 2018), and show that this approach can be used to learn complex trajectories under noisy input conditions on a control-relevant timescale. Furthermore, we demonstrate that this neuromorphic network architecture can not only robustly represent complex trajectories, but even generalize beyond its training experience.

2. METHODS

We first describe the architecture of the novel algorithm implemented on the neuromorphic chip Loihi, which supports robust robotic control of movement trajectories. The anisotropic network and its implementation is then explained in detail. Finally analyses methods to evaluate the implementation of the anisotropic network on Loihi, the stability of its network dynamics, and the learning of complex movement trajectories are described.

2.1. Architecture of the Algorithm for Robotic Control

The architecture, shown in **Figure 1A**, was designed to support the storage and execution of stable movement trajectories in real-time. The architecture consists of an input layer, an anisotropic network layer, and a pooling layer, all of which are fully implemented on Loihi. Spike patterns from the anisotropic network or the pooling layer are read out and serve as the basis for training output units.

The basic computational structure for the robotic control algorithm is the anisotropic network. Excitatory and inhibitory neurons are initialized with local, spatially inhomogeneous

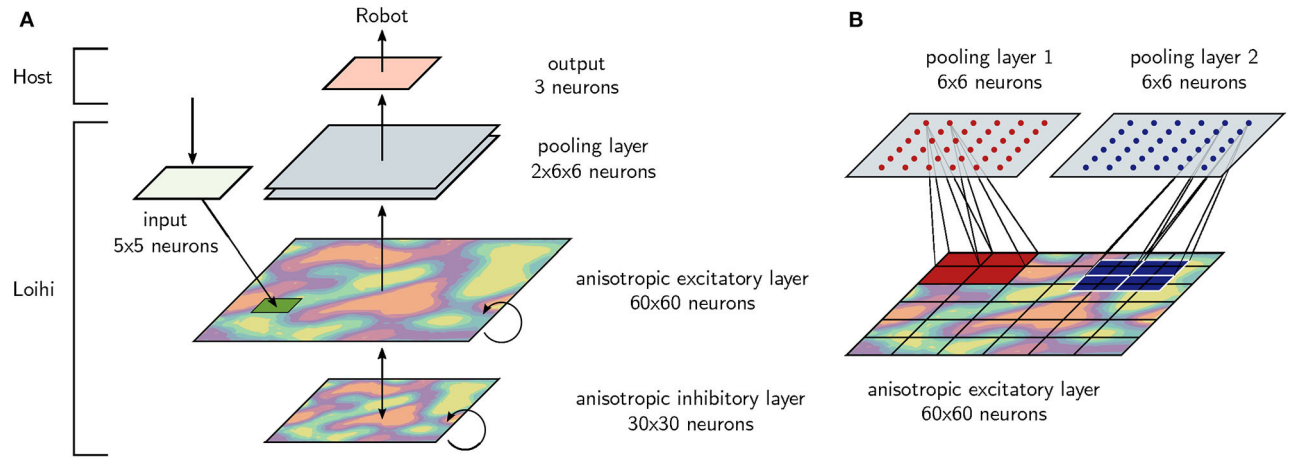


FIGURE 1 | The architecture of the robot control algorithm. **(A)** The anisotropic network is the basis of the architecture, indicated by the colored layers, and consists of 3,600 excitatory and 900 inhibitory neurons. An area of 25 excitatory neurons receive external input. The excitatory neurons of the anisotropic network project to a pooling layer. The anisotropic network, the input neurons and the 72 pooling neurons are simulated on-chip. Finally, the spiking activity is used to train a trajectory. **(B)** In detail, the pooling layer contains 72 neurons organized into two 6 × 6 grids. Each pooling neuron receives all-to-one feed-forward connections from a 10 × 10 square patch of the anisotropic network's excitatory layer. Hence, the pooling layer has 10 × 10 window size and stride 5.

connections as described below. An input is connected to a grid of 5 × 5 excitatory neurons in order to start spiking activity with a short input pulse.

The excitatory neurons of the anisotropic network are connected to a pooling layer with 72 excitatory neurons. Pooling layer neurons are organized into two grids with a size of 6 × 6 neurons, as shown in **Figure 1B**. Each neuron in the pooling layer receives input from a 10 × 10 group of excitatory neurons from the anisotropic network. These projections are all-to-one and all feed-feedforward weights are equal. In other words, the pooling layer has 10 × 10 window size and stride 5.

Depending on the task, either the excitatory neurons of the anisotropic network or the 72 neurons of the pooling layer are read out. Since reading out data from Loihi is a bottle neck that reduces the simulation speed considerably, the pooling layer is designed to reduce read out and therefore increase simulation speed. Finally, linear regression is applied to the spiking activity of the read-out (see section 2.5).

2.2. The Anisotropic Network

We briefly describe the main principles of the anisotropic network. For an in depth treatment, we refer readers to Spreizer et al. (2019).

In locally connected random networks (LCRNs), neurons are distributed in (connectivity) space (e.g., on a 2D grid or torus) and the connection probability between two neurons decreases (possibly non-monotonically) with the distance between them. Stable bumps of spatially localized activity can arise in LCRNs (Roxin et al., 2005; Hutt, 2008; Spreizer et al., 2017, 2019) and these activity bumps can move through the network in a stream-like manner if spatial asymmetries are introduced into the local connectivity (Spreizer et al., 2019).

The anisotropic EI-network consists of both excitatory and inhibitory neurons arranged on a 2D torus. Neurons project their axons in a distance dependent way with connection probability decreasing monotonically according to a Gaussian distribution. In a standard LCRN, axon projection profiles are centered at the neuron and axons project symmetrically in all directions. In the anisotropic EI-network, the Gaussian distribution is shifted for excitatory neurons such that connections to other excitatory neurons are formed preferentially in a particular direction (**Figure 2A**).

A so-called landscape is computed on the torus using Perlin noise (Perlin, 1985), and each point on the grid (neuron) is assigned a direction based on this. The Perlin landscape ensures that the preferred direction of nearby neurons are similar while preferred directions of those far apart are uncorrelated (**Figure 2B**). Each excitatory neuron's connectivity profile is shifted by one grid point in its preferred direction, resulting in spatially asymmetric but correlated connectivity. When a set of neurons in close proximity are stimulated, spatio-temporal sequences of activity lasting tens to hundreds of milliseconds are elicited (**Figure 2C**).

Taken together, a biologically plausible rule can generate spatially asymmetric connectivity structures supporting spatio-temporal sequences. Spreizer et al. (2019) show that if (1) individual neurons project a small fraction (∼2–5%) of their axons preferentially in a specific direction (**Figure 2A**), and (2) neighboring neurons prefer similar directions (**Figure 2B**), then sequences of neural activity propagate through the network (**Figure 2C**). This simple generative connectivity rule results in feed-forward paths through the otherwise locally connected random network.

2.3. Anisotropic Network Implementation

We adapted the anisotropic EI-network model from Spreizer et al. (2019). Since the total number of connections currently supported by the Loihi NxSDK-API is limited (see section 4),

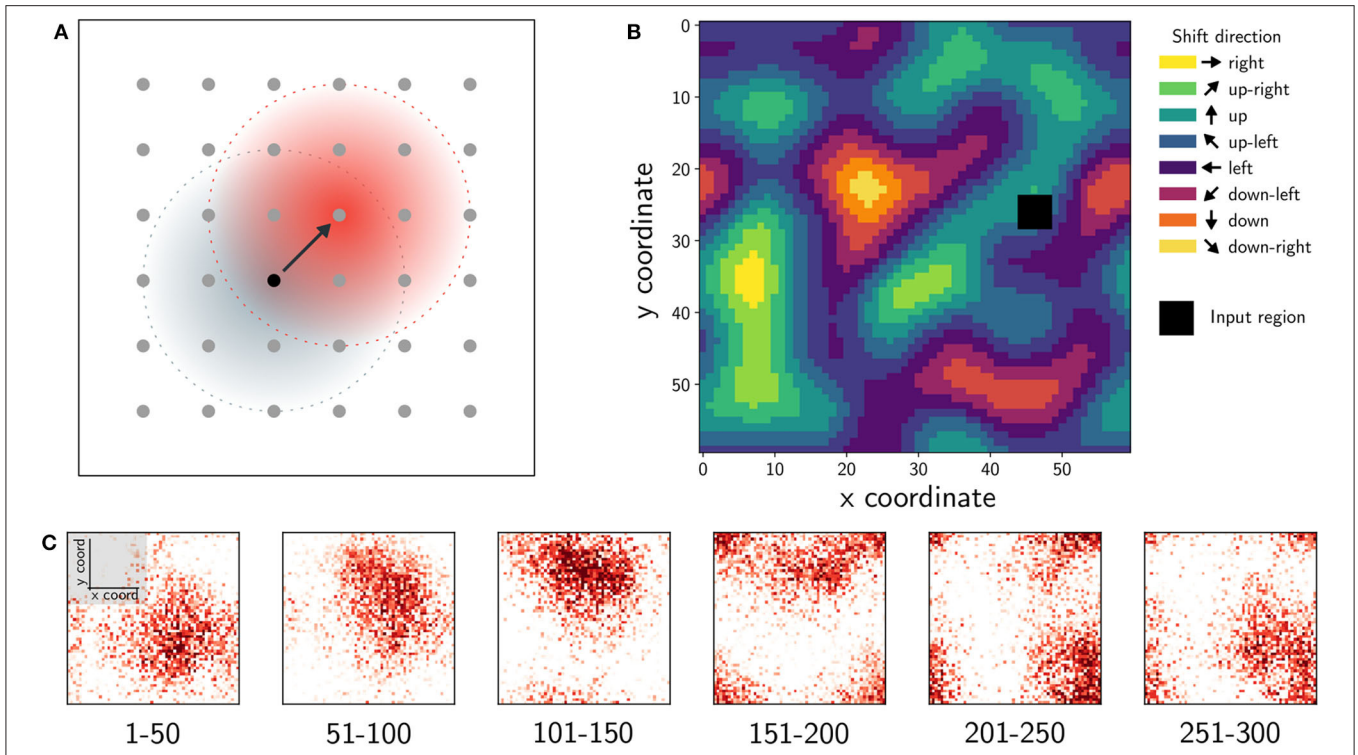


FIGURE 2 | The connectivity underlying the anisotropic network and resulting network dynamics. **(A)** Neurons are evenly distributed on a 2D grid, folded to form a torus. The black dot represents a reference neuron. From here outgoing connections can be drawn locally with a Gaussian distribution, either symmetrically (gray-blue) or non-symmetrically (red). In the non-symmetrical case the center of the Gaussian distribution is shifted by one neuron. This shift can be chosen in different ways. Either with the same shift direction for every neuron (homogeneously), a random shift direction for every neuron, or with a specific distribution. **(B)** If for each neuron the shift direction is chosen based on Perlin noise, the distribution of shift directions for the whole network results in an *anisotropic* connectivity structure. The black region represents spatially-localized input. **(C)** Given local input, the anisotropic network forms a bump of activity, which moves in a stream-like pattern through the inhomogeneous structure of connections. Here, we used the Loihi implementation and binned the spikes into non-overlapping time windows. Each graph shows the average firing rate over 50 time steps, color code depicts firing rate.

it was necessary to reduce network size by a factor of four to $npop_E = 3,600$ and $npop_I = 900$. Each neuron projects to $p_{conn} \times npop_E = 180$ excitatory targets and $p_{conn} \times npop_I = 45$ inhibitory targets, where $p_{conn} = 0.05$ is the connection probability. Connection probability decreases with distance according to a Gaussian distribution with space constants given in **Supplementary Table 1**. We first adapted the anisotropic EI-network model within NEST and then transferred it to Loihi, tuning the network to qualitatively match the behavior of the NEST simulation.

NEST Implementation

Neurons were modeled as leaky integrate-and-fire (LIF) neurons, with sub-threshold membrane potential v of neuron i evolving according to:

$$C_m \frac{dv_i}{dt} = -g_L(v_i(t) - E_L) + I_i(t) + I_i^{input}(t), \qquad (1)$$

where C_m is the membrane capacitance, g_L the leak conductance, and E_L the reversal potential. For neuron i, $I_i(t)$ is the total synaptic current from its recurrent connections and $I_i^{input}(t)$ the current induced by external input.

The total synaptic current $I_i(t)$ to neuron i at its recurrent synapses is the sum of the current transients at each of its synapses, $I_i(t) = \sum_j I_{ij}(t)$. When a pre-synaptic neuron spikes, a current transient is elicited with temporal profile given by an alpha function:

$$I_{ij}(t) = J^{syn} \frac{t - t_{j,k}}{\tau_{syn}} \exp\left(- \frac{t - t_{j,k}}{\tau_{syn}} \right). \qquad (2)$$

Note here that the superscript syn can denote both excitatory (exc) and inhibitory (inh) synapses. Synaptic strength is J^{syn}, synaptic time constant is τ_{syn}, and spike time is $t_{j,k}$ for the k^{th} spike from neuron j.

To compensate for the decreased network size and hence fewer recurrent connections (see above) we scaled up the synaptic weights. The excitatory synaptic current was scaled up by a factor of four to $J^{exc} = 40.0\,pA$. To ensure persistent spiking activity in response to an input pulse, the ratio of recurrent inhibition and excitation was reduced to $g = 4$. As a result, $J^{inh} = -g \times J^{exc} = -160.0\,pA$.

Activity was triggered by external input to a subset of neighboring neurons, each of which receives an input pulse of 500

spikes with synaptic strength $J^{input} = 1.0\,pA$ arriving according to a Gaussian distribution with standard deviation of 1 *ms*.

Loihi Implementation

For the implementation on neuromorphic hardware we used the research chip Loihi from Intel (Davies et al., 2018), which is a digital and deterministic chip that is based on an asynchronous design. The board we used contains two chips, with each chip providing 128 neuron cores and three embedded x86 CPUs. Each neuron core time-multiplexes the calculation and allows the implementation of up to 1,024 neurons each. We distributed the total of 4,572 utilized neurons (reservoir and pooling layer) with 20 neurons per core. Computation on the chip is performed in discrete time steps and has no relation to physical time. Finally, the Loihi board is connected to a desktop computer, called host in the following, via a serial bus (USB).

We translated the `NEST` implementation to the `NxSDK` (version 0.9.5-daily-20191223) for Loihi, provided by Intel labs (Lin et al., 2018). For this, we developed a software framework `PeleNet`[1], based on the `NxSDK`, especially for reservoir networks on Loihi. This framework was used for all simulations in this study.

The Loihi chip implements a leaky integrate-and-fire (LIF) neuron with current-based synapses and the membrane potential v of neuron i evolves according to

$$\frac{dv_i}{dt} = -\tau_v^{-1} v_i(t) + I_i(t) + I_i^{input}(t) - v_{th}\sigma_i(t), \tag{3}$$

where τ_v describes the time constant, v_{th} the firing threshold, $I_i(t)$ the total synaptic current from recurrent connections, $I_i^{input}(t)$ the current induced by the input, and $\sigma_i(t)$ denotes whether neuron i spiked at time t. The first term on the right-hand side controls voltage decay, the second/third term increases the voltage according to the synaptic/input currents, and the last term resets the membrane potential after a spike occurs.

While the `NEST` implementation uses alpha-function shaped synaptic currents (see Equations 1 and 2), Loihi's current-based synapses implement instantaneous rise and exponential decay. The total synaptic current from recurrent connections to neuron i is given by

$$I_i(t) = \sum_{i \neq j} J_{ij}^{syn}(\alpha_I * \sigma_j)(t) + I_i^{bias}, \tag{4}$$

where J_{ij}^{syn} is the synaptic strength from neuron j to neuron i which can be excitatory (J^{exc}) or inhibitory (J^{inh}) and I_i^{bias} is a bias term. The $\sigma_j(t)$ represents the incoming spike train from neuron j and $\alpha_I(t)$ a synaptic filter. The spike train for a neuron j is given by a sum of Dirac delta functions with

$$\sigma_j(t) = \sum_k \delta(t - t_{j,k}), \tag{5}$$

where $t_{j,k}$ is the time of spike k for neuron j. The function simply indicates whether neuron j spiked in time step t. The spike train is convolved with a synaptic filter given by

$$\alpha_I(t) = \tau_I^{-1} \exp\left(-\frac{t}{\tau_I}\right) H(t), \tag{6}$$

where τ_I is a time constant and $H(t)$ the unit step function.

With Equations (5) and (6), we can bring Equation (4) into a form which is comparable to Equation (2). Setting $I_{bias} = 0$, we get

$$\begin{aligned} I_i(t) &= \sum_{i \neq j} J_{ij}^{syn}(\alpha_I * \sigma_j)(t) \\ &\overset{\text{Equation(5)}}{=} \sum_{i \neq j} J_{ij}^{syn} \sum_x \alpha_I(x) \sum_k \delta((t-x) - t_{j,k})) \\ &= \sum_{i \neq j} J_{ij}^{syn} \sum_k \alpha_I(t - t_{j,k}) \\ &\overset{\text{Equation(6)}}{=} \sum_{i \neq j} J_{ij}^{syn} \sum_k \tau_I^{-1} \exp\left(\frac{t_{j,k} - t}{\tau_I}\right) H(t - t_{j,k}). \end{aligned} \tag{7}$$

Due to the filter, the input current induced by a pre-synaptic spike decays exponentially for each following time step. And instead of rising slowly, at the time of a spike, $t = t_{j,k}$, synaptic current increases by $\tau_I^{-1} J_{ij}^{syn}$. Thus, compared to the neuron model from the anisotropic network implementation in `NEST` (Spreizer et al., 2019), the hardware-implemented neuron model on Loihi differs since it lacks a current rise time.

2.4. Comparing the Implementations

Network activity was started with the input mentioned above and 500 discrete time steps in Loihi and 500 *ms* in `NEST` were recorded. In `NEST` the resolution was set to $dt = 0.1\,ms$ (see also **Supplementary Table 1**) per simulation step, while in Loihi a physical time is not defined. After the simulation, the `NEST` spiking data was binned to 1 *ms* to match the Loihi data. Note that, given the refractory period of 2 *ms*, the binned spike trains still contain binary values, but with a less precise information about the sub-millisecond spike times. In the end, both data sets, the spike trains for `NEST` and the spike trains for Loihi contained 500 discrete steps.

To compare the spiking patterns between `NEST` and Loihi quantitatively, we calculated the mean firing rate of groups of excitatory neurons in both networks, which is shown in **Figure 3B**. For this, we split the two dimensional network topology into a 6×6 grid, analogous to the grid used for the pooling layer (see **Figure 1B**) such that each grid position represents a group of 100 neurons. The indices of the groups are chosen from top left to bottom right. For each group, we averaged the firing rate over the 500 time steps resulting in 36 values.

2.5. Stability and Output Learning

To analyse the stability (**Figure 5**) of the network and for the output learning (**Figure 6**), we applied another protocol. Note

[1]https://github.com/sagacitysite/pelenet/tree/neurorobotics

that from this point on, the NEST implementation was not used. Out of the 25 excitatory neurons connected to the input, we stimulated only 24 neurons such that 1 neuron stays silent (**Figure 4A**). This input grid then allows 25 different input configurations and therefore 25 different trials with a noise level of 4%. Every trial was recorded for 215 time steps and then the activity was stopped by resetting the membrane voltages. We did this by applying a C code that runs on one x86 core on each chip (a so called SNIP). After waiting 30 time steps, the next input was applied to the network. The applied protocol is indicated by arrows in **Figure 5A** on top of the spike train plots.

To learn trajectories from the spike patterns we used multiple linear regression, which was applied to two different tasks. In the *representation task*, we estimated model parameters based on all 25 trials and tested on one of them. In the *generalization task*, training was performed on only 24 trials and testing was done using the remaining trial. Both tasks are sketched in **Figure 4B**. To compare the anisotropic network with a classical reservoir computing approach, we also implemented a randomly connected network on Loihi and exchanged the anisotropic network in our network architecture with a randomly connected network of equal size. We set the parameters of the random network such that the main statistics of both networks match. The firing rate in both networks is in a range of $0.1 - 0.2$ spikes per number of neurons (**Figure 5A** bottom) and the mean Fano factor over all trials is relatively similar with $\overline{FF}_{\text{rand}} = 0.80 \pm 0.01$ for the randomly connected network and $\overline{FF}_{\text{aniso}} = 0.84 \pm 0.002$ for the anisotropic network.

After all data were recorded, in a first step, we prepared the data for the estimation of the regression model. Due to the slow rise in the firing rate of the network (**Figure 3C**), the very first time steps contain little information. Therefore, we omitted the first 5 time steps which reduces the length of the data set to 210 per trial. Before the linear regression was applied to the spike data, the spike trains were binned in order to smooth our spiking data. We used a sliding window with a width of 10 time steps, which reduced the length of the data set again from 210 to 200.

Next we used the binned data of the 200 time steps to estimate the regression parameters. In addition to the spiking data from the neurons, an intercept was added such that the number of parameters equal the number of neurons plus one. The linear regression model was performed on the CPU of the host computer, using the spiking data from the readout provided by Loihi. The two different tasks, the representation task and the generalization task, were performed using the spiking data from the anisotropic network as well as those from the randomly connected network. Furthermore, we estimated output weights based on either the pooling layer neurons (72 neurons) or the excitatory neurons of the reservoir (3,600 neurons), see also **Figure 1**. The excitatory neuron readout serves as a control and compares the pooling layer approach to a traditional readout.

For the estimation based on all excitatory reservoir neurons, we applied an elastic net regularization (Zou and Hastie, 2005) to avoid overfitting, due to the numerous parameters. This regularization approach for regression models simply combines LASSO and ridge regression. We used the `fit_regularized` function from the `statsmodels` package in `Python`, which applies elastic net as

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left[\|y - X\beta\|_2^2 + \alpha \left((1-\lambda)\|\beta\|_2^2 + \lambda\|\beta\|_1 \right) \right]. \quad (8)$$

In this variant the parameter α determines the degree of regularization and λ balances between LASSO ($L1$ regularization) and ridge regression ($L2$ regularization).

To better compare the predicted function with the target function, we applied a Savitzky–Golay filter (Savitzky and Golay, 1964) to smooth the predicted function. For this we used the `savgol_filter` function of the `Python` package `scipy`. We chose a window length of 21 and an order of the polynomial of 1 as parameters for smoothing.

We trained our algorithm on 7 different trajectories performing ordinary robotic tasks. The tasks are *hide*, *unhide*, *move down*, *move up*, *pick and place*, *put on top* and *take down* (see e.g., Wörgötter et al. (2020)). Movement data is given in 3 dimensional Cartesian coordinates, resulting in three outputs or – biologically speaking – in three rate coded output neurons.

3. RESULTS

We start by demonstrating that the main principles of the anisotropic network are preserved by the Loihi implementation and then confirm that the Loihi-based anisotropic network admits noise-robust spiking dynamics. Based on these findings, we demonstrate that our architecture can learn complex trajectories under noisy input conditions.

3.1. Implementing the Computer-Based Anisotropic Network on Loihi

Due to the different hardware architectures, we first assess the extent to which the Loihi-based implementation of the anisotropic network agrees with the computer-based NEST simulation. Please note that it is not our goal to compare two neural network simulators, but to ensure that the anisotropic network implementation on Loihi preserves the main features. For the sake of comparison, we used the same connectivity structure and input positions for both implementations. The networks were initialized at rest and spike patterns were evoked via a spatially-localized input. Raster plots of evoked spike trains indicate that, although the detailed spiking activity is not identical, the overall spiking pattern is mainly preserved (**Figure 3A**). Accordingly, the mean firing rate of the network for each implementation evolves similarly over time (**Figure 3C**).

We confirmed the similarity between both implementations quantitatively, comparing the mean firing rate and firing rate variability over several input and network initializations. **Figure 3D** shows the distribution of mean firing rates over (1) 15 different input positions for the same network connectivity and over (2) 15 different initializations of the network connectivity with a fixed input position. Across input positions in the same network, firing rates for the Loihi implementation were $\bar{f}^{\text{L}}_{\text{inp}} = 0.131 \pm 0.008$ and for the NEST implementation $\bar{f}^{\text{N}}_{\text{inp}} = 0.118 \pm 0.004$. Across network initializations, firing rates were $\bar{f}^{\text{L}}_{\text{init}} = 0.120 \pm 0.013$ for Loihi and $\bar{f}^{\text{L}}_{\text{init}} = 0.113 \pm 0.009$ for NEST. In

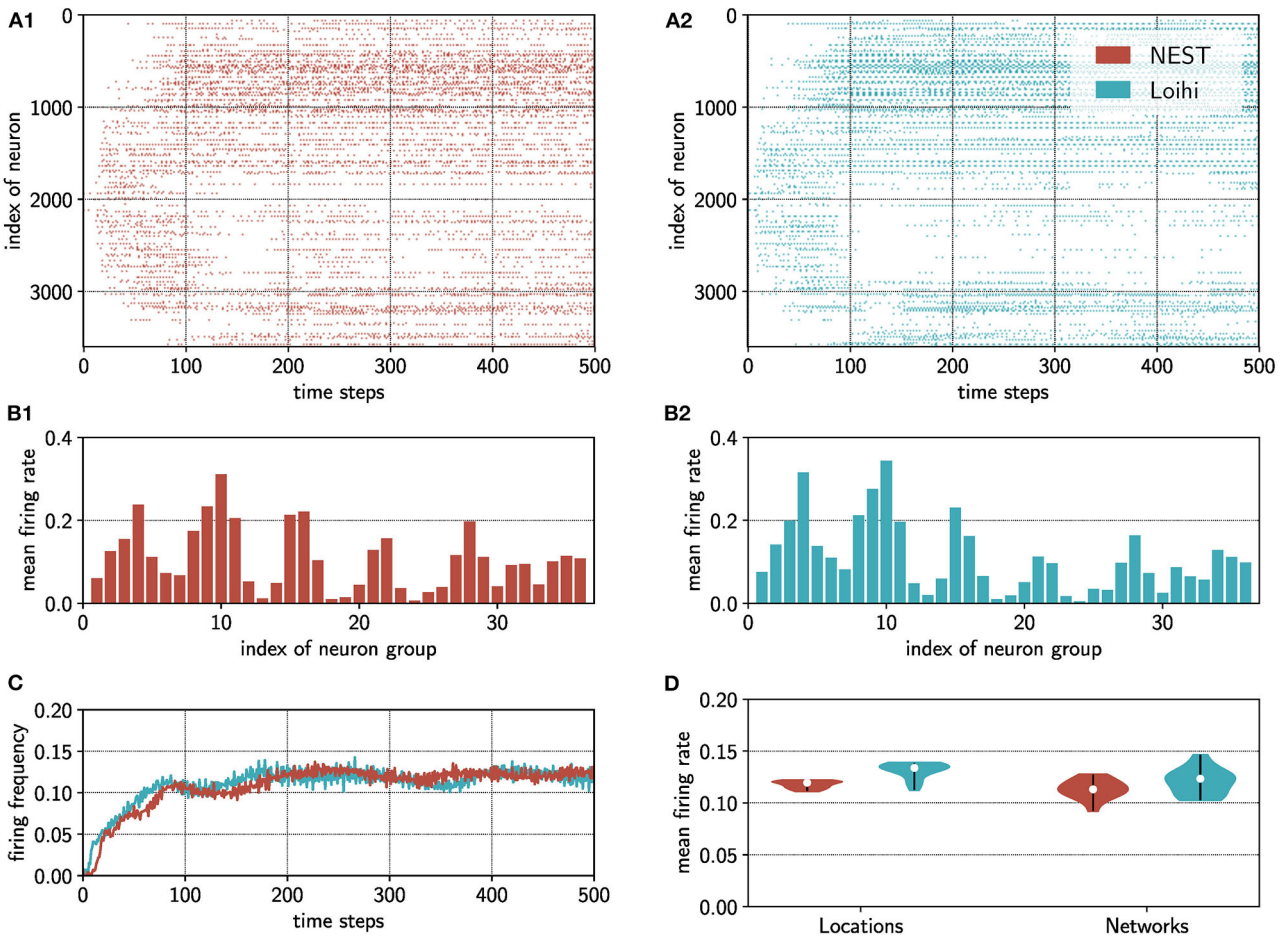


FIGURE 3 | One example of the anisotropic network implementation on Loihi compared to activity statistics of the NEST simulations. **(A)** The spiking activity is shown as spike raster plots. Red: NEST simulation; blue: Loihi implementation. Both networks are initialized with the same weight matrix and triggered with the same input. **(B)** By pooling neurons into groups, each consisting of 100 neurons, and binning across the whole simulation time, we obtain the distribution of mean firing rates (# spikes / # neurons per group) for the NEST and Loihi implementation. **(C)** The time course of the network firing rate (# spikes / # neurons) from the NEST and Loihi simulations is depicted. **(D)** Testing one network initialization with 15 different input locations (Locations) and one location but 15 different network initialization (Networks) for the NEST as well as the Loihi implementation. White dots indicate the median of the mean firing rate (# spikes / # neurons) of the 15 simulations for each case.

addition, the ranges (minimum to maximum mean firing rate) of the obtained mean firing rates are very tight and overlap largely between both implementations. For the locations, the values for Loihi are in a range of $0.11 \leq f^{\mathrm{L}}_{\mathrm{inp}} \leq 0.14$ and for NEST in an interval of $0.11 \leq f^{\mathrm{N}}_{\mathrm{inp}} \leq 0.12$. In case of the different initializations, we obtained mean firing rates between $0.10 \leq f^{\mathrm{L}}_{\mathrm{init}} \leq 0.15$ for Loihi and $0.09 \leq f^{\mathrm{N}}_{\mathrm{init}} \leq 0.13$ for the NEST implementation. To compare the variability of the firing rate in both implementations, we evaluated the Fano factor (FF): For different input positions, we obtained a mean of $\overline{FF}^{\mathrm{L}}_{\mathrm{inp}} = 0.83 \pm 0.03$ for Loihi and $\overline{FF}^{\mathrm{N}}_{\mathrm{inp}} = 0.86 \pm 0.01$ for NEST. In the case of the 15 network initializations, the mean FF for Loihi is $\overline{FF}^{\mathrm{L}}_{\mathrm{init}} = 0.84 \pm 0.02$ and $\overline{FF}^{\mathrm{N}}_{\mathrm{init}} = 0.86 \pm 0.01$ for NEST. All FF values between Loihi and NEST are very close to each other and indicate that spiking is less variable than a Poisson process.

Given that the neural activity in the anisotropic network forms spatially-localized bumps moving through the network, we next measured its average spatial distribution. For the spike rasters shown in **Figure 3A**, we pooled the neurons into groups of 100, taking into account the topology of the network (see **Figure 1B**), and calculated the mean firing frequencies averaged across the whole simulation time. This procedure provides a distribution of the mean activity across the network for both implementations (**Figure 3B**). Normalizing these distributions and comparing them with a Kolmogorov–Smirnov test reveals that the activity distributions from the NEST- and Loihi-based implementations do not differ significantly ($D = 0.11$, $p = 0.97 > 0.05$). Hence the spatial structure of activity patterns is similar in both implementations.

Taken together, we conclude that the Loihi implementation matches the NEST-based anisotropic network implementation according to diverse statistics of the network activity. This

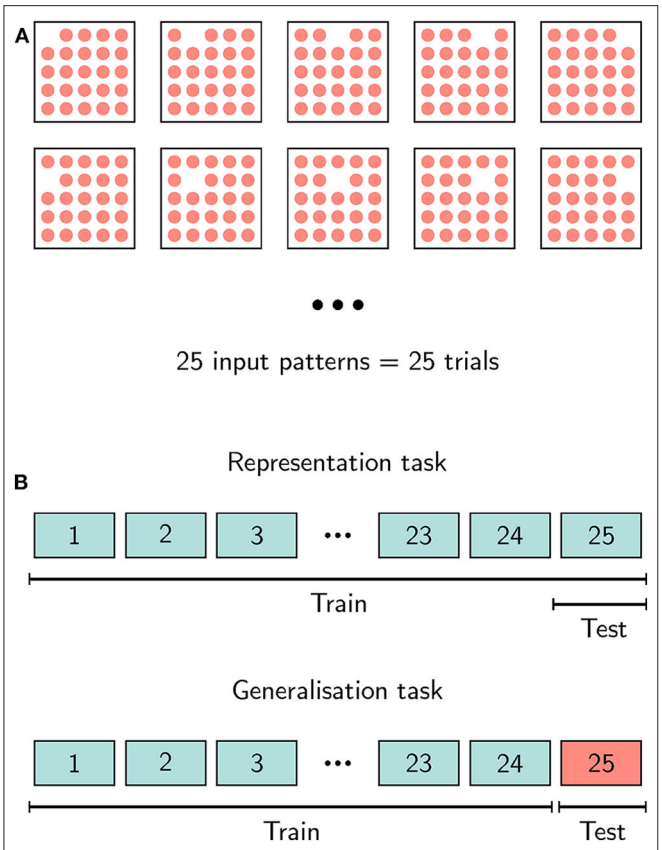


FIGURE 4 | The input protocols for the representation and generalization task. **(A)** A square represents an area of 25 possible input neurons. During a trial, we triggered the network activity by activating 24 out of the 25. This leads to 25 different input patterns and therefore to 25 different trials. **(B)** We used two different training tasks. In the representation task, for the trajectory training, we used the spiking activity of all 25 trials and tested on one of them. In the generalization task, we trained only on the spiking activity of 24 trials and tested on the spikes of a different trial, which was not used for training.

indicates a successful transfer of the core principles of the anisotropic network to neuromorphic hardware despite the differences in architecture.

3.2. The Loihi Implementation of the Anisotropic Network Is Robust to Input Noise

Next, to assess the robustness of the Loihi-based anisotropic network to input noise, we evaluate the stability of spiking dynamics. An input pulse is administered to an area of the anisotropic excitatory layer consisting of 25 neurons (**Figure 1A**). In each trial, 24 of these neurons were activated and a different neuron was systematically excluded from the input, leading to 25 different possible input configurations and, thus, to 25 unique trials (**Figure 4**).

For each trial the network activity was started with a short input pulse of one time step. We then recorded 200 time steps of activity, stopped the activity manually and activated it again by the next input. The protocol is also indicated on top of **Figure 5A**.

As a control, we applied the same protocol to a randomly connected network implemented on Loihi and compared it with the anisotropic network implementation. For this, we implemented the same algorithmic architecture, but exchanged the anisotropic network with a randomly connected network of equal size. The spiking activity of the first three trials is shown in **Figure 5A1** for the anisotropic network (green) and **Figure 5A2** for the randomly connected network (brown). Due to the inhomogeneous connectivity structure, the activity of the anisotropic network spreads out like a stream in the network. Note that, given the torus network topology (see section 2), the activity stream wraps around from neurons with low indices to neurons with high indices (**Figure 5A**). As expected, in the randomly connected network such a stream-like spread of activity does not form.

The population firing rates progress differently in the anisotropic and randomly connected networks. The mean firing rate of the anisotropic network increases slowly until it reaches a relatively constant rate slightly above 0.1. The randomly connected network was tuned such that it generates a similar mean population firing rate (see section 2). However, unlike in the anisotropic network, the firing rate does not rise gradually, but instead starts at about $0.1 - 0.2$ straight away. The slow start in the anisotropic network is due to the relatively small input area and the local connectivity of the network. While moving forward in the 2D-topology, the area of activity grows step by step, which can intuitively be understood as a snowball effect.

In order to measure the stability of the spiking dynamics between different input trials, we calculated the pairwise differences between the spike patterns of all combinations of the 25 trials. The mean and standard deviations of these differences are shown in **Figure 5B** for both the anisotropic network (green) and the randomly connected network (brown). The differences between trials are much higher over the whole time course for the randomly connected network than for the anisotropic network. For the anisotropic network, the deviations of the trial-to-trial differences are very small in the beginning and drift apart over time. To quantify this, we performed a Levene test with three samples at time steps 10, 100, and 190 which revealed that the variance stays constant between the differences of the randomly connected network trials ($W = 0.60$, $p = 0.54 > 0.05$) but increases for the anisotropic network trials over time ($W = 208.87$, $p = 3.36 \cdot 10^{-75} < 0.05$). This means that, over time, spiking patterns between some trials stay very similar whereas some trial comparisons tend to differ more. Therefore, the anisotropic network tends to slowly diverge with time, which can also be seen by the increasing mean differences. Importantly the mean differences in the anisotropic network remain much lower than the spiking differences between the trials in the randomly connected network, even at the end of the 200 time steps. This clearly demonstrates the stabilizing feature of the anisotropic network.

To visualize differences between the single trials, we reduced the dimensionality of the spiking data by applying principal component analysis (PCA) to all trials (see section 2). The

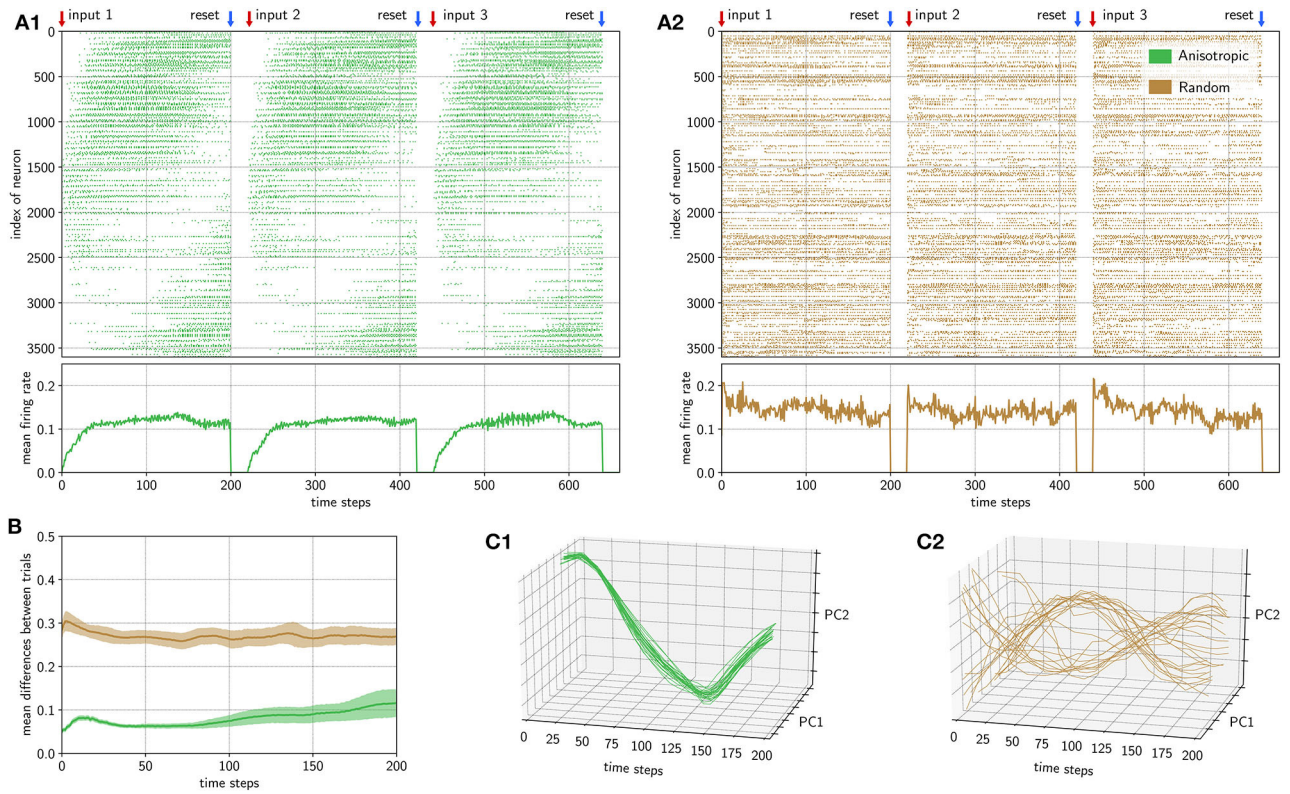


FIGURE 5 | The Loihi implementation of the anisotropic network is robust to varying input conditions, while a randomly connected network is not. **(A)** Three examplary trials out of the 25 trials are shown for a simulation based on an anisotropic connectivity structure (green) and with randomly initialized weights (brown). The anisotropic structure clearly has a stream-like spiking pattern, where the firing rate starts slowly until it reaches a constant rate. The randomly connected network shows a Poisson-like spiking pattern, where the firing rate starts directly at a high level. **(B)** The solid lines show the mean difference between all trial combinations and the shaded area indicates the standard deviation of the trial-to-trial differences for the anisotropic (green) and random (brown) network. **(C)** In the reduced space of the first two principal components of the activity of all 25 trials over time for both networks, we can clearly see that the spiking pattern of the anisotropic network are very similar between trials, while the activity in the randomly connected network differ much more between trials.

results are shown in **Figure 5C**. For the anisotropic network (**Figure 5C1**), all trajectories are very similar whereas for the randomly connected network (**Figure 5C2**) the trajectories differ considerably. We quantified this by calculating statistics in the first dimension of the PCA space. First, we obtained the pairwise normalized mean squared error between all trials for each network type. The normalized mean error between the trials of the randomly connected network is $\mathrm{MSE}_{\mathrm{rand}} = 1.66$, while the anisotropic network has a mean error of only $\mathrm{MSE}_{\mathrm{aniso}} = 0.03$, which is significantly lower (Mann–Whitney U-test: $U = 3572.0$, $p = 4.27 \cdot 10^{-85} < 0.05$). Even though some trajectories seem to follow a common path, in the random network, the mean standard deviation for the first principle component $\bar{\sigma}_{\mathrm{rand}} = 2.50$ is significantly higher than in the anisotropic network with $\bar{\sigma}_{\mathrm{aniso}} = 0.41$ (Mann–Whitney U-test: $U = 0.0, p = 1.56 \cdot 10^{-8} < 0.05$). This indicates sufficient stability over 200 time steps for the anisotropic network.

Taken together, this shows the ability of the anisotropic network to produce stable spiking dynamics under noisy input conditions. In addition it confirms the successful implementation of the network on Loihi. In the next step we will use this intrinsic stability feature of the anisotropic network to learn robust trajectories and examine if our network produces sufficient variability to learn arbitrary functions.

3.3. Learning Robust Trajectories

After having tested and demonstrated the stabilizing feature of the anisotropic network, we aimed to use its robustness to train arbitrary output trajectories. This step makes use of the underlying network architecture shown in **Figure 1** and adds a linear regression model on top of this architecture for a robot control task. The overall algorithm contains the initialization, creation and simulation of the anisotropic network, which is running on the neuromorphic hardware Loihi, and the output learning of the trajectories, which is calculated on the host CPU.

To show the robustness of this algorithm, we learned 7 different 3D-trajectories commonly used in robotic research, like pick-and-place or put-on-top (see section 2). Using these target functions, we applied two different tasks, a representation and a generalization task, as shown in **Figure 4B**. In the representation task we estimated the linear regression model based on all 25 trials and predicted one of them, showing that

the variability in the anisotropic network is sufficient to learn an arbitrary function. To show the ability of our algorithm to robustly generalize for variations in the input, we also apply a generalization task, where we estimate the regression model on 24 trials and predicted the trajectory for an unseen trial.

As before, here we also compare the performance of the anisotropic network with the randomly connected network as a control. For our algorithm we estimate our model based on the 72 pooling layer neurons, which we can read out efficiently from the chip. Since we reduce the parameter space of the linear regression model by using only the spiking activity of the pooling layer neurons as data, we also estimated all models based on the 3, 600 excitatory reservoir neurons for comparison.

Results for the representation task, based on all excitatory neurons, revealed that the excitatory reservoir neurons contain enough variability to represent an arbitrary output function with high accuracy. An example is shown in **Supplementary Figure 1A1** for the randomly connected network and in **Supplementary Figure 1A2** for the anisotropic network. If the model was estimated on the 72 pooling layer neurons the number of available parameters is heavily decreased by a factor of 50. But still the amount of information seems to be satisfactory for the anisotropic network (**Supplementary Figure 1A4**), but not for the randomly connected case (**Supplementary Figure 1A3**). Normalized root mean squared error between the predicted trajectory and the target trajectory, averaged over 7 3D-trajectories, are shown in **Figure 6A1**. The errors for all trajectories using the spiking activity of the 3, 600 excitatory reservoir neurons are very low (left plot) for both networks, but interestingly even lower for the anisotropic network. For the errors of the estimation based on the pooling layer neurons (right plot in **Figure 6A1**), the mean error over all trajectories is still low for the anisotropic network $e_{\text{aniso}} = 0.02 \pm 0.003$, compared to the randomly connected network $e_{\text{rand}} = 0.33 \pm 0.07$. Due to the inhomogeneous weight structure and the stream-like spread of spiking activity in the anisotropic network, the neurons in the pooling layer can maintain variability, as can be seen in **Supplementary Figure 1B1**. Intuitively, since nearby neurons have correlated activity patterns, pooling over them preserves information. In contrast, as shown in **Supplementary Figure 2B2**, the spiking activity in the pooling layer of the randomly connected network simply produces downsampled random spiking activity and therefore reduced variability.

In the generalization task, the parameters for the movement trajectory were estimated based on the spiking activity of 24 trials. We then predicted the same trajectory based on the spiking activity elicited by a 25th trial, not seen during training. This task was designed to test the robustness of the system to a variation in initial conditions. To compare the classical reservoir computing approach with our network architecture, we trained the network based on all 3, 600 neurons and on the 72 output neurons. In addition, this tested the ability of the pooling neurons to preserve sufficient variability while reducing the number of parameters.

For the full network read-out, we applied a linear regression model based on all excitatory neurons of the anisotropic network. Since fitting a model based on all 3, 600 neurons requires many parameters, here we used an elastic net regularization estimation method (see section 2) to reduce the number of parameters and to avoid overfitting. Optimizing the regularization parameters resulted in $\alpha = 0.001$ and $\lambda = 0.05$. For the pooling layer read-out, we estimated a linear regression model based on the pooling layer neurons without regularization.

In **Figure 6A2**, we show the average normalized root-mean-squared deviation over 7 trajectories. In both cases (using excitatory neurons or pooling layer neurons) the error of the anisotropic network is much lower, showing that the anisotropic network has a better performance compared to a classical randomly connected network.

For the network architecture with the randomly connected network, the elastic net approach, based on all excitatory neurons, has a better performance than the linear regression approach, based on the pool neurons (t-test: $t = -4.24$, $p = 0.0001 < 0.05$). Interestingly, the error for the anisotropic network is lower when the trajectories are estimated based on the pool neurons compared to the excitatory neurons (Mann–Whitney U-test: $U = 47.0$, $p = 6.75 \cdot 10^{-6} < 0.05$). This indicates that, for the anisotropic network, the pooling layer is an equivalent, or even better regularization method compared to the elastic net approach with all excitatory neurons.

Figure 6B shows all three dimensions of the predicted trajectory over time for a *hide* movement. The overall trajectory is shown in **Figure 6C**. We also calculated a smoothed version, using a Savitzky–Golay filter (Savitzky and Golay, 1964) (see section 2), to better compare the prediction with the target. This shows that the anisotropic network implemented on Loihi, combined with the pooling layer, contains sufficient variability to represent complex 3D trajectories while at the same time remaining stable for at least 200 time steps.

3.4. Simulation on Loihi in Real-Time

In addition to evaluating the stability of the system, we also looked at the speed of the network simulation. The data we used came from a Kuka robot arm, which can run fluently with an output frequency of 100 Hz, therefore the 200 time steps equal 2 s of movement. In the following we denote this reference as "real-time." To achieve a real-time output of spiking data from the Loihi chip, the speed of the simulation of the neurons and the data transfer from the chip to the host must be higher than the necessary data frequency of the robot for a smooth movement.

The simulation of these 200 time steps requires $t_{3,600}^{\text{aniso}} = 15.73\,s$ for one trial on Loihi, when all 3, 600 excitatory reservoir neurons were read out from the system. This speed is about 8 times slower than real-time. When reading out only from the 72 pooling neurons, the speed increases to $t_{72}^{\text{aniso}} = 1.49\,s$ per trial, which is 25% faster than real-time and therefore well-suited for robot control.

The simulation speed of the anisotropic network ($t_{3,600}^{\text{aniso}} = 15.73\,s$ & $t_{72}^{\text{aniso}} = 1.49\,s$ per trial) and the randomly connected network ($t_{3,600}^{\text{rand}} = 16.11\,s$ & $t_{72}^{\text{rand}} = 1.73$ per trial) were nearly the same, which is expected since the number of neurons is the same and the number of synapses is similar. Thus, the anisotropic

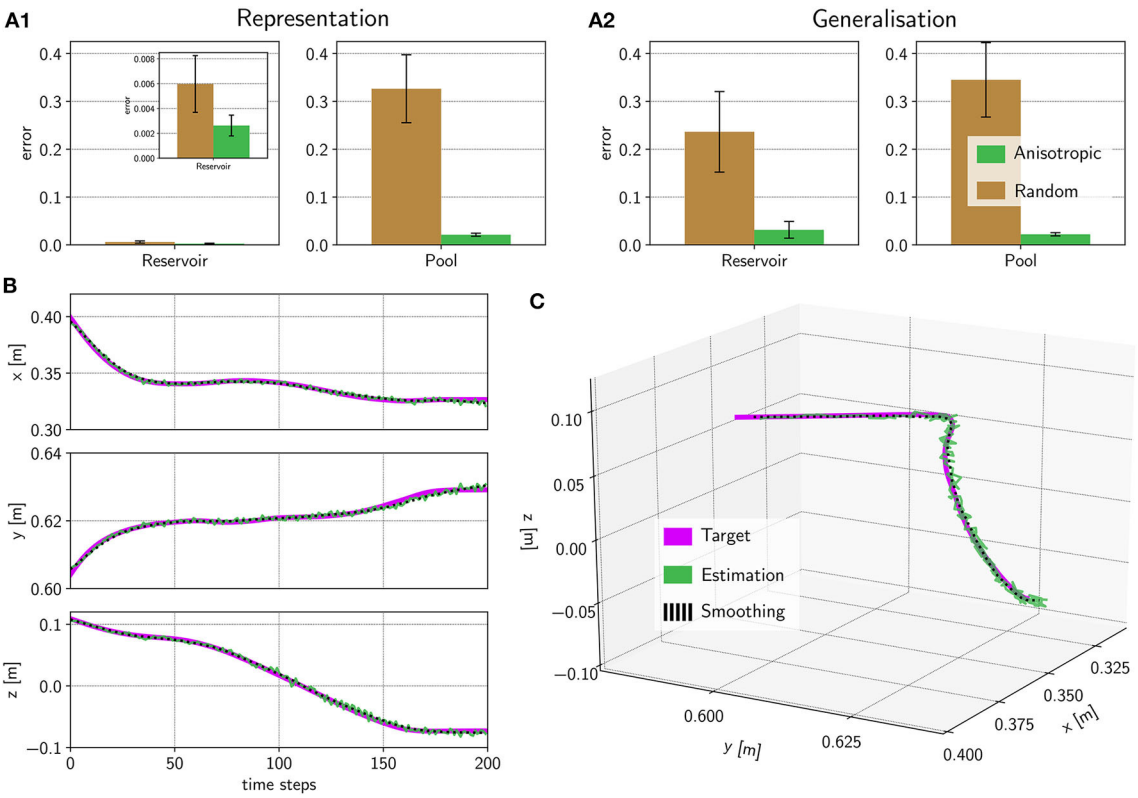


FIGURE 6 | Predicting a trajectory on the spiking activity. **(A)** We predicted 7 different trajectories with 3 dimensions each and calculated the error regarding the target trajectory using a normalized root-mean-square error. In all cases, the anisotropic network has a lower error than the randomly connected network. **(B)** For the *hide* movement, we show one example of the predicted trajectory of the anisotropic network for each of the 3 dimensions. Here, we used the pooling neurons (green) in comparison with the target (pink) and a smoothed variant of the prediction (black dotted). The whole trajectory is shown in **(C)**.

network has, in terms of speed, no disadvantage compared to the randomly connected network.

Therefore, the pooling layer does not only reduce the sensitivity of the system but also helps to speed up the system considerably. Together, this supports robotic applications where trajectories can be stored and replayed robustly in real-time.

4. DISCUSSION

We aimed to develop an algorithm for neuromorphic hardware, which provides stable spiking dynamics under noisy input conditions, in order to make use of the low power neuromorphic chips for future autonomous systems. For this, we derived an algorithm to store and control robotic movement sequences that unfold on a control-relevant timescale of seconds. To validate our approach, we chose a set of 2-s-long robot arm movements that were triggered by noisy inputs.

For our approach we chose a recently developed spiking neural network (Spreizer et al., 2019) with an inhomogeneous weight structure. In a first step, we successfully transferred the main principles of this network to the Loihi research chip from Intel (Davies et al., 2018), a neuromorphic hardware architecture implementing spiking neurons. In a second step, we tested the stability of the anisotropic network implementation and compared its stability to a classical randomly connected network, similar to echo state networks (Jaeger, 2001, 2007) or liquid state machines (Maass et al., 2002). We finally used a pooling layer (**Figure 1**) to efficiently read out spiking data from the chip. Using these spiking data we were able to learn 3D trajectories in a noise-robust way (**Figure 6C**). The pooling layer successfully increased the simulation speed to faster than real-time. It was also intended to make the spiking activity more invariant to small changes in the network, which is the exact purpose of using pooling layers in deep neural networks (Goodfellow et al., 2016, Chapter 9.3; Boureau et al., 2010). A pooling layer has been applied to spiking neural networks before (Tavanaei and Maida, 2017; Tavanaei et al., 2019), but – to the best of our knowledge – such a structure has never been applied to enhance the performance of read-outs from recurrent network architectures. The fact that the pooling layer improved performance for the anisotropic network in our study indicates that implementing pooling layers in reservoir computing architectures could be useful in other cases, for example when the reservoir has spatially-dependent connectivity (Maass et al., 2002), and especially for reducing parameters on algorithms running on neuromorphic hardware.

Taken together, in this study we provide an algorithm for storing stable trajectories in spiking neural networks, optimized

for the neuromorphic hardware Loihi. The network architecture is capable of executing these trajectories on demand in real-time given noisy, and even never-before-seen, inputs. While an exhaustive exploration of the parameter space remains the subject of future work, we have shown that the anisotropic network admits stable sequences with sufficient variability for output learning across hundreds of milliseconds, making it suitable for applications reaching far beyond motor control. Further, we demonstrated that spike-based pooling can implement on-chip regularization for the ansiotropic network, improving read out speed and accuracy. In contrast, in the randomly connected network nearby neurons show uncorrelated activity and spatial pooling has no benefit. Thus, spatial pooling in locally-connected SNNs proved to be a promising feature, specifically for real-time robotic control on neuromorphic hardware. Importantly, we provide the first neuromorphic implementation which has no global learning or adaptation mechanism and produces noise-robust spiking patterns on a control-relevant timescale with sufficient variability to learn arbitrary functions.

While other approaches employing spiking neural networks exist, in general they fail to meet at least one of the mentioned criteria. This means, in their current form, these models are either not implementable on neuromorphic hardware or do not produce sequences that are stable, variable and long enough. We briefly describe these models and highlight how they may be adapted for neuromorphic implementation.

Laje and Buonomano (2013) presented an "innate training" approach. The network was initialized with a short input pulse and a modified FORCE algorithm (Sussillo and Abbott, 2009) was used to train the recurrent connections. This stabilizes the innate structure of the recurrent connections and allows a network state between chaotic and locally stable activity patterns. A trained output trajectory was robust to perturbations, due to the tuned recurrent weights. Unfortunately this algorithm uses a rate coded network and non-local learning rules, both of which are not applicable for most neuromorphic systems.

Pehlevan et al. (2018) analyzed different approaches to solve the stability-variability trade-off in the context of songbird songs. One additional and important criterion for their evaluation was the ability of an algorithm to provide temporal flexibility, such that outputs can be replayed faster or slower. They concluded that a synfire chain model fits best to solve this task. While this approach seems to model the dynamics underlying songbird songs with flexible timing, synfire chains have a feed-forward structure which makes them less flexible than recurrent network types.

Hennequin et al. (2014) put more focus on getting stable output from unstable initial conditions. They used an optimization algorithm to build an inhibitory structure that helps to stabilize the excitatory activity. More precisely, the strength of existing inhibitory connections was changed or new inhibitory synapses were created or removed using an algorithm based on a relaxation of the spectral abscissa of the weight matrix (Vanbiervliet et al., 2009). With this they obtained relatively stable spiking dynamics. Interestingly, this approach is similar to our study in a sense that both approaches focus on the weight matrix. While their proposed solution to the stability-variability trade-off is promising, so far the algorithm has mainly been tested with rate coded networks. A more elaborate analysis with a spiking neural network would be of interest.

Another recent approach involves multiplexing oscillations in a spiking neural network (Miall, 1989; Vincent-Lamarre et al., 2020). Two input units inject sine-waves into a reservoir of neurons and the spiking dynamics in the reservoir follow a stable and unique pattern, which enables the learning of a long and stable output. Compared to our algorithm, the oscillating units provide a continuous input to the network. We see this approach as a potential alternative to the anisotropic network for robotic control. Interestingly, stability is encoded in time rather than space, which raises the question whether this approach could be combined with a pooling layer, reflecting temporal structure instead of spatial structure.

Maes et al. (2020) trained a recurrently connected spiking network such that small groups of neurons become active in succession and thus provide the basis for a simple index code. Via a supervisor signal, output neurons are trained to become responsive to a particular group or index from the recurrent network and, thus, fire in a temporal order encoded in the feed-forward weights to the output layer. Importantly, learning within the recurrent network and from the recurrent network to the output layer is done using spike-timing dependent plasticity. However, as is, their implementation has a few small, but likely reconcilable, incompatibilities with the neuromorphic hardware considered here. For example, learning and synaptic normalization is only local to the neuron, and not to the synapse and they rely on adaptive exponential integrate and fire neurons, which are not implemented by Loihi. With some modifications, their model may provide another neuromorphically implementable approach.

While our approach provides an algorithm for storing stable trajectories, our two-chip Loihi system is limited in the number of neurons available, constrained mainly by the high number of synapses in our recurrent network. Since this limitation is mainly caused by the current `NxSDK` software and not by hardware, we expect an improvement in upcoming releases. With more neurons available we expect even better stability, reducing the last remaining variations in our predictions and allowing even longer movement actions, beyond 2 s. At this point, further investigation of how performance depends on network size, network parameters, and pooling layer configuration will be of interest.

With more neurons available, one could add multiple inputs to the network. We hypothesize that nearby input locations lead to similar activity patterns, while input regions far from each other produce distinct activity patterns. This behavior could be used to train multiple trajectories from different input locations. With this, more complex robotic control tasks could be performed, beyond the generation of single trajectories.

One general hurdle in developing neuromorphic implementations is the difficulty in transferring existing spiking neural network models from CPU-based implementations to neuromorphic hardware. As outlined in the section 2, Loihi provides a fixed hardware-implemented neuron model. It is possible to adjust parameters, but not the neuron model itself. Therefore, a perfect match between traditional simulators like `NEST` (Gewaltig and Diesmann, 2007) or `Brian2` (Stimberg

et al., 2019) and neuromorphic hardware, like Loihi, is in general an issue for future neuromorphic algorithms. Efficient methods for translating neuroscientific models to Loihi is the subject of current work.

Finally, to complete the algorithm for autonomous use cases, in which Loihi is able to control a robot independently, an on-chip output learning algorithm is vital. This requires the implementation of an output neuron on the chip, with appropriate on-chip output weights. It is already possible to train weights offline and transfer them to Loihi, applied for example in `Nengo Loihi` (Bekolay et al., 2014; Hampo et al., 2020). We expect that the on-chip regularization inherent in spatial pooling will improve the robustness of future online output learning algorithms.

5. CONCLUSION

Taken together, we developed an algorithm which can serve as a basic unit in robotic applications. The anisotropic network structure offers stability against noisy inputs and the overall architecture, especially using the pooling layer, paves the way for further steps in the development of algorithms for neuromorphic hardware. Our study proposes an algorithm based on *intrinsic* self-stabilizing features of a well-initialized anisotropic connectivity structure, which can overcome the instability problem of spiking neural networks and support robust outputs on a timescale of seconds.

DATA AVAILABILITY STATEMENT

The PeleNet framework for Loihi, which was written for this study, can be found on GitHub (https://github.com/sagacitysite/pelenet/tree/neurorobotics). The data that support the findings of this study are available from the corresponding author on request.

AUTHOR CONTRIBUTIONS

AL contributed the network simulations. CM contributed the Loihi implementation, including the Pelenet framework. CT acquired funding and supervised the study. All authors designed the study and reviewed the manuscript.

FUNDING

The research was funded by the H2020-FETPROACT project Plan4Act (#732266) [CM, AL, CT], by the German Research Foundation (#419866478) [AL, CT], and by the Intel Corporation via a gift without restrictions.

ACKNOWLEDGMENTS

The authors are thankful to Osman Kaya for providing Kuka robot trajectories, to Arvind Kumar and Lukas Ruff for helpful discussions, and to Tristan Stöber for improving the text. All of them helped to improve the quality of this work. Furthermore, we thank the Intel Corporation for providing access to their Loihi chip.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2020.589532/full#supplementary-material

REFERENCES

Amit, D. J. (1992). *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge: Cambridge University Press.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., et al. (2014). Nengo: a Python tool for building large-scale functional brain models. *Front. Neuroinform.* 7, 1–13. doi: 10.3389/fninf.2013.00048

Blouw, P., Choo, X., Hunsberger, E., and Eliasmith, C. (2019). "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop* (New York, NY; Albany, NY: Association for Computing Machinery), 1:8. doi: 10.1145/3320288.3320304

Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (Madison, WI: Omnipress), 111–118.

Brunel, N. (2000). Dynamics of networks of randomly connected excitatory and inhibitory spiking neurons. *J. Physiol.* 94, 445–463. doi: 10.1016/S0928-4257(00)01084-6

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

DeWolf, T., Stewart, T. C., Slotine, J.-J., and Eliasmith, C. (2016). A spiking neural model of adaptive arm control. *Proc. R. Soc. B Biol. Sci.* 283:20162134. doi: 10.1098/rspb.2016.2134

Eliasmith, C., and Anderson, C. H. (2004). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. Cambridge, MA: MIT Press.

Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. (2014). The SpiNNaker project. *Proc. IEEE* 102, 652–665. doi: 10.1109/JPROC.2014.2304638

Gewaltig, M.-O., and Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia* 2:1430. doi: 10.4249/scholarpedia.1430

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press, 330–334.

Hampo, M., Fan, D., Jenkins, T., DeMange, A., Westberg, S., Bihl, T., et al. (2020). "Associative memory in spiking neural network form implemented on neuromorphic hardware," in *International Conference on Neuromorphic Systems 2020*, 1–8. doi: 10.1145/3407197.3407602

Hennequin, G., Vogels, T. P., and Gerstner, W. (2014). Optimal control of transient dynamics in balanced networks supports generation of complex movements. *Neuron* 82, 1394–1406. doi: 10.1016/j.neuron.2014.04.045

Hutt, A. (2008). Local excitation-lateral inhibition interaction yields oscillatory instabilities in nonlocally interacting systems involving finite propagation delay. *Phys. Lett. A* 372, 541–546. doi: 10.1016/j.physleta.2007.08.018

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, Vol. 2 (Washington, DC: IEEE), 1398–1403. doi: 10.1109/ROBOT.2002.1014739

Itskov, V., Curto, C., Pastalkova, E., and Buzsáki, G. (2011). Cell assembly sequences arising from spike threshold adaptation keep

track of time in the hippocampus. *J. Neurosci.* 31, 2828–2834. doi: 10.1523/JNEUROSCI.3773-10.2011

Jaeger, H. (2001). *The "Echo State" Approach to Analysing and Training Recurrent Neural Networks-With an Erratum Note*. German National Research Center for Information Technology, Bonn. GMD Technical Report.

Jaeger, H. (2007). Echo state network. *Scholarpedia* 2:2330. doi: 10.4249/scholarpedia.2330

Khalastchi, E., Kaminka, G. A., Kalech, M., and Lin, R. (2011). "Online anomaly detection in unmanned vehicles," in *The 10th International Conference on Autonomous Agents and Multiagent Systems* (Richland, SC; Taipei: International Foundation for Autonomous Agents and Multiagent Systems), 115–122.

Laje, R., and Buonomano, D. V. (2013). Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nat. Neurosci.* 16:925. doi: 10.1038/nn.3405

Lee, C., Panda, P., Srinivasan, G., and Roy, K. (2018). Training deep spiking convolutional neural networks with stdp-based unsupervised pre-training followed by supervised fine-tuning. *Front. Neurosci.* 12:435. doi: 10.3389/fnins.2018.00435

Lin, C.-K., Wild, A., Chinya, G. N., Cao, Y., Davies, M., Lavery, D. M., et al. (2018). Programming spiking neural networks on intel's Loihi. *Computer* 51, 52–61. doi: 10.1109/MC.2018.157113521

London, M., Roth, A., Beeren, L., Häusser, M., and Latham, P. E. (2010). Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature* 466, 123–127. doi: 10.1038/nature09086

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput.* 14, 2531–2560. doi: 10.1162/089976602760407955

Maes, A., Barahona, M., and Clopath, C. (2020). Learning spatiotemporal signals using a recurrent spiking network that discretizes time. *PLoS Comput. Biol.* 16:e1007606. doi: 10.1371/journal.pcbi.1007606

Miall, C. (1989). The storage of time intervals using oscillating neurons. *Neural Comput.* 1, 359–371. doi: 10.1162/neco.1989.1.3.359

Murray, J. M. et al. (2017). Learning multiple variable-speed sequences in striatum via cortical tutoring. *eLife* 6:e26084. doi: 10.7554/eLife.26084

Nachstedt, T., and Tetzlaff, C. (2017). Working memory requires a combination of transient and attractor-dominated dynamics to process unreliably timed inputs. *Sci. Rep.* 7, 1–14. doi: 10.1038/s41598-017-02471-z

Naseer, T., Burgard, W., and Stachniss, C. (2018). Robust visual localization across seasons. *IEEE Trans. Robot.* 34, 289–302. doi: 10.1109/TRO.2017.2788045

Neckar, A., Fok, S., Benjamin, B. V., Stewart, T. C., Oza, N. N., Voelker, A. R., et al. (2018). Braindrop: a mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proc. IEEE* 107, 144–164. doi: 10.1109/JPROC.2018.2881432

Pehlevan, C., Ali, F., and Ölveczky, B. P. (2018). Flexibility in motor timing constrains the topology and dynamics of pattern generator circuits. *Nat. Commun.* 9, 1–15. doi: 10.1038/s41467-018-03261-5

Perlin, K. (1985). An image synthesizer. *SIGGRAPH Comput. Graph.* 19, 287–296. doi: 10.1145/325165.325247

Roxin, A., Brunel, N., and Hansel, D. (2005). Role of delays in shaping spatiotemporal dynamics of neuronal activity in large networks. *Phys. Rev. Lett.* 94:238103. doi: 10.1103/PhysRevLett.94.238103

Savitzky, A., and Golay, M. J. (1964). Smoothing and differentiation of data by simplified least squares procedures. *Anal. Chem.* 36, 1627–1639. doi: 10.1021/ac60214a047

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modelling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris: IEEE), 1947–1950. doi: 10.1109/ISCAS.2010.5536970

Sompolinsky, H., Crisanti, A., and Sommers, H.-J. (1988). Chaos in random neural networks. *Phys. Rev. Lett.* 61:259. doi: 10.1103/PhysRevLett.61.259

Spreizer, S., Aertsen, A., and Kumar, A. (2019). From space to time: spatial inhomogeneities lead to the emergence of spatiotemporal sequences in spiking neuronal networks. *PLoS Comput. Biol.* 15:e1007432. doi: 10.1371/journal.pcbi.1007432

Spreizer, S., Angelhuber, M., Bahuguna, J., Aertsen, A., and Kumar, A. (2017). Activity dynamics and signal representation in a striatal network model with distance-dependent connectivity. *eNeuro* 4. doi: 10.1523/ENEURO.0348-16.2017

Stimberg, M., Brette, R., and Goodman, D. F. (2019). Brian 2, an intuitive and efficient neural simulator. *eLife* 8:e47314. doi: 10.7554/eLife.47314

Sussillo, D., and Abbott, L. F. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron* 63, 544–557. doi: 10.1016/j.neuron.2009.07.018

Tang, G., Shah, A., and Michmizos, K. P. (2019). Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. *arXiv preprint arXiv:1903.02504*. doi: 10.1109/IROS40897.2019.8967864

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., and Maida, A. (2019). Deep learning in spiking neural networks. *Neural Netw.* 111, 47–63. doi: 10.1016/j.neunet.2018.12.002

Tavanaei, A., and Maida, A. (2017). "Bio-inspired multi-layer spiking neural network extracts discriminative features from speech signals," in *Neural Information Processing*, eds L. Derong , X. Shengli, L. Yuanqing, Z. Dongbin,and E.-A. El-Sayed (Cham: Springer International Publishing), 899–908. doi: 10.1007/978-3-319-70136-3_95

Tsodyks, M. (1999). Attractor neural network models of spatial maps in hippocampus. *Hippocampus* 9, 481–489. doi: 10.1002/(SICI)1098-1063(1999)9:4<481::AID-HIPO14>3.0.CO;2-S

Van Vreeswijk, C., and Sompolinsky, H. (1996). Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science* 274, 1724–1726. doi: 10.1126/science.274.5293.1724

Vanbiervliet, J., Vandereycken, B., Michiels, W., Vandewalle, S., and Diehl, M. (2009). The smoothed spectral abscissa for robust stability optimization. *SIAM J. Optimizat.* 20, 156–171. doi: 10.1137/070704034

Vincent-Lamarre, P., Calderini, M., and Thivierge, J.-P. (2020). Learning long temporal sequences in spiking networks by multiplexing neural oscillations. *Front. Comput. Neurosci.* 14:78. doi: 10.3389/fncom.2020.00078

Voelker, A. R., and Eliasmith, C. (2017). Methods for applying the neural engineering framework to neuromorphic hardware. *arXiv [Preprint]*. arXiv:1708.08133.

Wörgötter, F., Ziaeetabar, F., Pfeiffer, S., Kaya, O., Kulvicius, T., and Tamosiunaite, M. (2020). Humans predict action using grammar-like structures. *Sci. Rep.* 10, 1–11. doi: 10.1038/s41598-020-60923-5

York, L. C., and Van Rossum, M. C. (2009). Recurrent networks with short term synaptic depression. *J. Comput. Neurosci.* 27:607. doi: 10.1007/s10827-009-0172-4

Zheng, P., and Triesch, J. (2014). Robust development of synfire chains from multiple plasticity mechanisms. *Front. Comput. Neurosci.* 8:66. doi: 10.3389/fncom.2014.00066

Zou, H., and Hastie, T. (2005). Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B* 67, 301–320. doi: 10.1111/j.1467-9868.2005.00503.x

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 26 January 2021
doi: 10.3389/fnbot.2020.578803

Reverse Engineering and Robotics as Tools for Analyzing Neural Circuits

*Ioannis Pisokas**

Institute of Perception, Action and Behaviour, School of Informatics, University of Edinburgh, Edinburgh, United Kingdom

Understanding neuronal circuits that have evolved over millions of years to control adaptive behavior may provide us with alternative solutions to problems in robotics. Recently developed genetic tools allow us to study the connectivity and function of the insect nervous system at the single neuron level. However, neuronal circuits are complex, so the question remains, can we unravel the complex neuronal connectivity to understand the principles of the computations it embodies? Here, I illustrate the plausibility of incorporating reverse engineering to analyze part of the central complex, an insect brain structure essential for navigation behaviors such as maintaining a specific compass heading and path integration. I demonstrate that the combination of reverse engineering with simulations allows the study of both the structure and function of the underlying circuit, an approach that augments our understanding of both the computation performed by the neuronal circuit and the role of its components.

Keywords: robotics, neurorobotics, navigation, head direction cells, ring attractor, insect, central complex, Drosophila melanogaster

1. INTRODUCTION

Neurorobotics attempts to derive inspiration from neuroscience on how the brain solves problems in order to develop robust and adaptive artificial agents. The combination of neuroscience with embodied robot agents provides a platform for testing hypotheses and deciphering the principles on which the brain operates. One approach for deciphering the principles of neuronal circuit operation is to implement phenomenological computational models of the neuronal circuit and then identify and analyze similarities between the models and the neuronal circuit in the hope of learning about the neuronal circuit's architecture. Such an approach is exemplified by work comparing features learned by deep convolutional neural networks with those found in the ventral visual system of animals (e.g., Yamins et al., 2014; Cichy et al., 2016; Yamins and DiCarlo, 2016). Phenomenological models attempt to reproduce the mapping of inputs to outputs while being only weakly constrained with respect to the actual neuronal circuit's architecture, thus admitting a range of possible implementations. Therefore, this approach has the potential to provide inspiration for hypothesis formulation and for focusing further research but does not unravel the actual neuronal circuits of biological organisms.

Another approach for analyzing neuronal circuits is to simulate part of the connectome in order to study the circuit's function. This approach is faithful to the actual neuronal connectivity, thus imposing strong constraints with respect to the biological architecture (as done for example by Kakaria and de Bivort, 2017). This approach has the potential to provide insights about the computation performed by the actual neuronal circuit; however, it does so based on phenomenological observations about computation at the system level and does not provide us with a real mechanistic understanding of the underlying neuronal circuit structure and component interaction.

OPEN ACCESS

Edited by:
Christian Tetzlaff,
University of Göttingen, Germany

Reviewed by:
Takeshi Kano,
Tohoku University, Japan
Alexander J. Cope,
The University of Sheffield, United Kingdom

***Correspondence:**
Ioannis Pisokas
i.pisokas@sms.ed.ac.uk

Received: *01 July 2020*
Accepted: *18 December 2020*
Published: *26 January 2021*

Citation:
Pisokas I (2021) Reverse Engineering and Robotics as Tools for Analyzing Neural Circuits. Front. Neurorobot. 14:578803. doi: 10.3389/fnbot.2020.578803

A third approach is to reverse engineer the actual neuronal circuit in order to decipher its organization and structure. Reverse engineering is a technique traditionally used for unraveling the inner workings of hardware devices (Rekoff, 1985). It aims to describe a system at the component level and explain how its components interact with each other. Once the structure of a neuronal circuit is reverse engineered, we can study how its neurons interact and draw hypotheses about the circuit's function on the basis of its neuronal components, thereby offering a mechanistic level of understanding.

Each of the three approaches has merits on its own, but their combination can provide an even more powerful tool for deciphering the function of neuronal circuits. A component-level understanding of the neuronal circuit structure through reverse engineering can be combined with the second approach, that is, computational simulations in order to understand the circuit's function. Deriving such a mechanistic understanding of the neuronal circuit at the neuron level will enable us to modify and customize it for use in specific applications, including robotics. I present here an example of this approach by reverse engineering the head direction circuit of the fruit fly and then utilizing simulations of a situated robotic agent to characterize the circuit's performance.

1.1. Insects as an Example Organism

A limiting factor in the study of any system, including the brain, is the level of detail at which it can be scrutinized. However, where detail is available, understanding structure and function may be difficult because naturally evolved neural systems do not obey an overarching structural simplicity principle. On an interesting crossroad of complexity and available neuroanatomical detail are insects. Insects have relatively small and simple brains compared with vertebrates and yet solve many similar problems, such as perception, navigation, foraging, homing, and reproduction. Recent developments of genetic tools and methods provide us with the unique opportunity to study insect brains at the single neuron level. The relative simplicity, together with the fine level of detail available about insect brains, enable us to reverse engineer their neuronal circuits, understand their operation and derive principles that can guide our design of solutions to problems in robotics.

Recent research in insect neurobiology has focused on the study of the central complex of the fruit fly *Drosophila melanogaster*. The central complex is a brain structure that has been preserved through millions of years of evolution and exists across all insect species (Homberg et al., 2011). This brain structure has been implicated in spatial orientation (Neuser et al., 2008; Triphan et al., 2010; Homberg et al., 2011), locomotor control (Strauss, 2002; Ritzmann et al., 2012; Martin et al., 2015; Varga et al., 2017), visual memory (Liu et al., 2006; Neuser et al., 2008; Ofstad et al., 2011), and path integration (Cope et al., 2017; Stone et al., 2017). The central complex consists of five neural formations: the protocerebral bridge, the ellipsoid body, the fan shaped body, the noduli, and the asymmetric bodies (Wolff and Rubin, 2018). The neuronal connectivity of the central complex has an intricate and yet topographically regular structure. Tracing the neurons of the whole central complex is still an ongoing task; however, most of the neurons innervating two of its structures, the protocerebral bridge (PB) and the ellipsoid body (EB), have been traced in adequate detail in the fruit fly *D. melanogaster*, by multiple labs (e.g., Green and Maimon, 2018; Wolff and Rubin, 2018; Turner-Evans et al., 2020), allowing us to reverse engineer the underlying circuit.

Calcium imaging of the neurons that innervate both the PB and the EB, while a tethered fruit fly is walking or flying in a virtual reality environment, has revealed a striking relationship between neuronal activity and behavior. Specifically, it has been observed that the neuronal ensemble maintains localized spiking activity—commonly called an activity "bump"—that moves from one group of neurons to the next as the animal rotates with respect to its surroundings (Seelig and Jayaraman, 2015; Kim et al., 2017; Giraldo et al., 2018). The neuronal activity "bump" is maintained even when the visual stimulus is removed, and it moves relative to the no longer visible cue as the animal walks in darkness (Seelig and Jayaraman, 2015). Thus, this neuronal activity appears to constitute an internal encoding of heading, which is strongly reminiscent of the hypothetical ring attractor (Amari, 1977) proposed by Skaggs et al. (1995) to account for the "head direction" cells of rats (Taube et al., 1990).

Ring attractor models typically consist of a topological ring of neurons utilizing opposing excitatory and inhibitory synapses to establish a unique activity "bump" around the ring, with neurons forming lateral excitatory connections to neighboring neuronal units and inhibitory connections inhibiting neurons on the opposite side of the ring (Taube et al., 1990; Skaggs et al., 1995; Zhang, 1996). The result is that the most active neurons suppress the activity of all other neurons around the ring and a unique "bump" of activity emerges. Adequate external stimulation of a neuron in the ring causes the activity "bump" to move to the new most active neuron and this new attractor state to be maintained even after the stimulus is removed. This type of ring attractor model can reproduce the phenomena recorded via calcium imaging of fruit flies (Kim et al., 2017). However, this is only a phenomenological similarity and does not reveal whether the actual neuronal circuit in the animal's brain has the same form as this hypothetical ring attractor or if a different circuit structure produces the phenomena.

In this paper, I investigate the circuit structure and function separately. I illustrate that using reverse engineering on the projection patterns of the fruit fly's heading tracking neuronal circuit is possible to reveal an underlying connectivity that has a ring structure with eight-fold radial symmetry. I subsequently illustrate that combining insights from reverse engineering with simulations allows us to explore the circuit's function and identify some notable differences from classic ring attractor models, which may contribute to the stability and flexibility of its function.

2. NEURONAL CIRCUIT ANALYSIS

As an illustrative example of the usefulness of reverse engineering of a neuronal circuit, I will present a detailed explanation of the process applied to the fruit fly's head tracking circuit. This

technique was recently applied to two insect species and the results were presented in Pisokas et al. (2020). Here, I illustrate the reverse engineering process in detail to enable others to apply it to different neuronal circuits and I show that this approach can help us understand neuronal circuit structure and function.

The circuit structure will be reverse engineered at the neuron level of abstraction, removing details about neuron anatomy, biophysics, and location. In the particular case of the central complex, neurons follow a topographically regular pattern, which offers an advantage that will be exploited in the process. The reverse engineering procedure described in the sequel consists of three steps:

1. First, we identify neuron classes. Each neuron class follows a particular connectivity pattern.
2. Second, we identify the neural volumes where neurons form synapses with each other. We number these volumes so that we can systematically inspect them.
3. Third, for each class of neurons, we record connections between neurons in a directed graph. To this end, we focus on each neuron in turn and add its output connections with other neurons.

In the central complex, there is redundancy in the neuronal circuit due to the mirrored connectivity in the left and right hemispheres. The final graphs shown here have eight neurons for each neuron class, which is the result of an iterative process removing redundancy in each iteration. In the first iteration, there were as many graph nodes as there are neurons in the circuit. In each iteration, duplicate neurons were removed and the same process was repeated to reach the final result.

2.1. What Is the Effective Neuronal Circuit Structure?

A subset of neuron types in the central complex appear to be the key elements of a circuit with a ring structure. The connectivity of the neurons has been inferred here from anatomical data, with overlapping neuronal terminals assumed to form synapses between them (Wolff et al., 2015; Wolff and Rubin, 2018). The following analysis considers four types of neurons, the E-PG, P-EG, P-EN, and Delta7 neurons (**Table 1**), in accordance with previous work (Green et al., 2017; Kakaria and de Bivort, 2017; Kim et al., 2017; Su et al., 2017). Each of the four types of neurons follows a particular connectivity pattern.

These neurons innervate two of the central complex structures: the protocerebral bridge and the ellipsoid body. The protocerebral bridge (PB) consists of nine "glomeruli" in each hemisphere, arranged the one next to the other (**Figure 1**). The ellipsoid body (EB) consists of eight sectors called "tiles." Each tile is further divided into two "wedges" (**Figure 1**). Neurons form synapses within glomeruli of the PB or tiles of the EB. Since all neurons considered here form synapses in the PB, we number the neurons by the glomerulus they innervate. Since Delta7 neurons have both their input and output terminals in the PB we number them by the glomerulus where their output terminals are located.

The E-PG, P-EG, and P-EN neurons are assumed to have excitatory effect on their postsynaptic neurons, while Delta7 neurons are assumed to form inhibitory synapses with their postsynaptic neurons, as Kakaria and de Bivort (2017) proposed. These assumptions are consistent with RNA sequencing, indicating that E-PG, P-EG, and P-EN are cholinergic while Delta7 glutamatergic (Turner-Evans et al., 2020). At this point, we have done the preparatory work (steps 1 and 2) and we can proceed with deriving the underlying effective circuit by redrawing the connectivity as a directed graph, which is a convenient representation for studying circuit topology.

TABLE 1 | Neuronal nomenclature.

Model neuron name	Included neurons	Systematic names (Wolff and Rubin, 2018)
E-PG	E-PG and E-PG$_T$	PBG1–8.b-EBw.s-D/V GA.b and PBG9.b-EB.P.s-GA-t.b
P-EN	P-EN	PBG2-9.s-EBt.b-NO1.b
P-EG	P-EG	PBG1–9.s-EBt.b-D/V GA.b
Delta7	Delta7 or Δ7	PB18.s-GxΔ7Gy.b and PB18.s-9i1i8c.b

Correspondence between neuron names used in the model and the neurons names used in the literature. The first column shows the names used in this paper to refer to each group of neurons. The other two columns provide the shorthand consensus names and the full neuron names used in the literature.

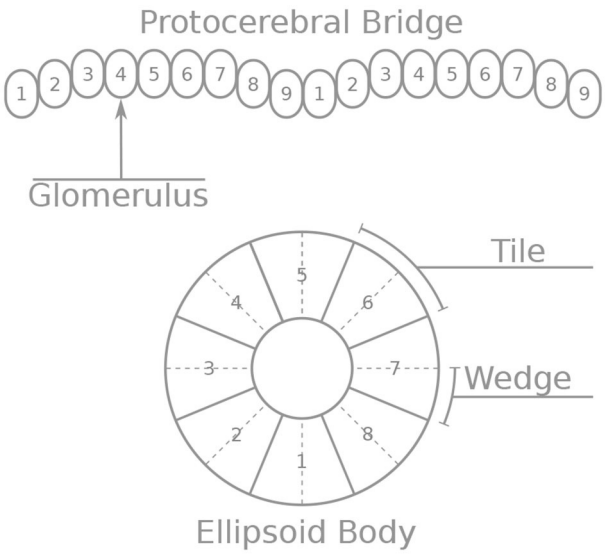


FIGURE 1 | Schematic depiction of the protocerebral bridge and ellipsoid body anatomy. The protocerebral bridge (PB) consists of nine "glomeruli" in each hemisphere, arranged the one next to the other. The ellipsoid body (EB) consists of eight sectors called "tiles" further divided into "wedges".

2.1.1. Inhibitory Circuit

First, we will walk through reverse engineering the connectivity of the first class of neurons, the eight inhibitory Delta7 neurons. These neurons innervate the whole length of the PB (**Figure 2A**). Anatomical evidence shows that each Delta7 neuron has output synaptic terminals in two or three glomeruli along the PB and input terminals across all remaining glomeruli (Wolff and Rubin,

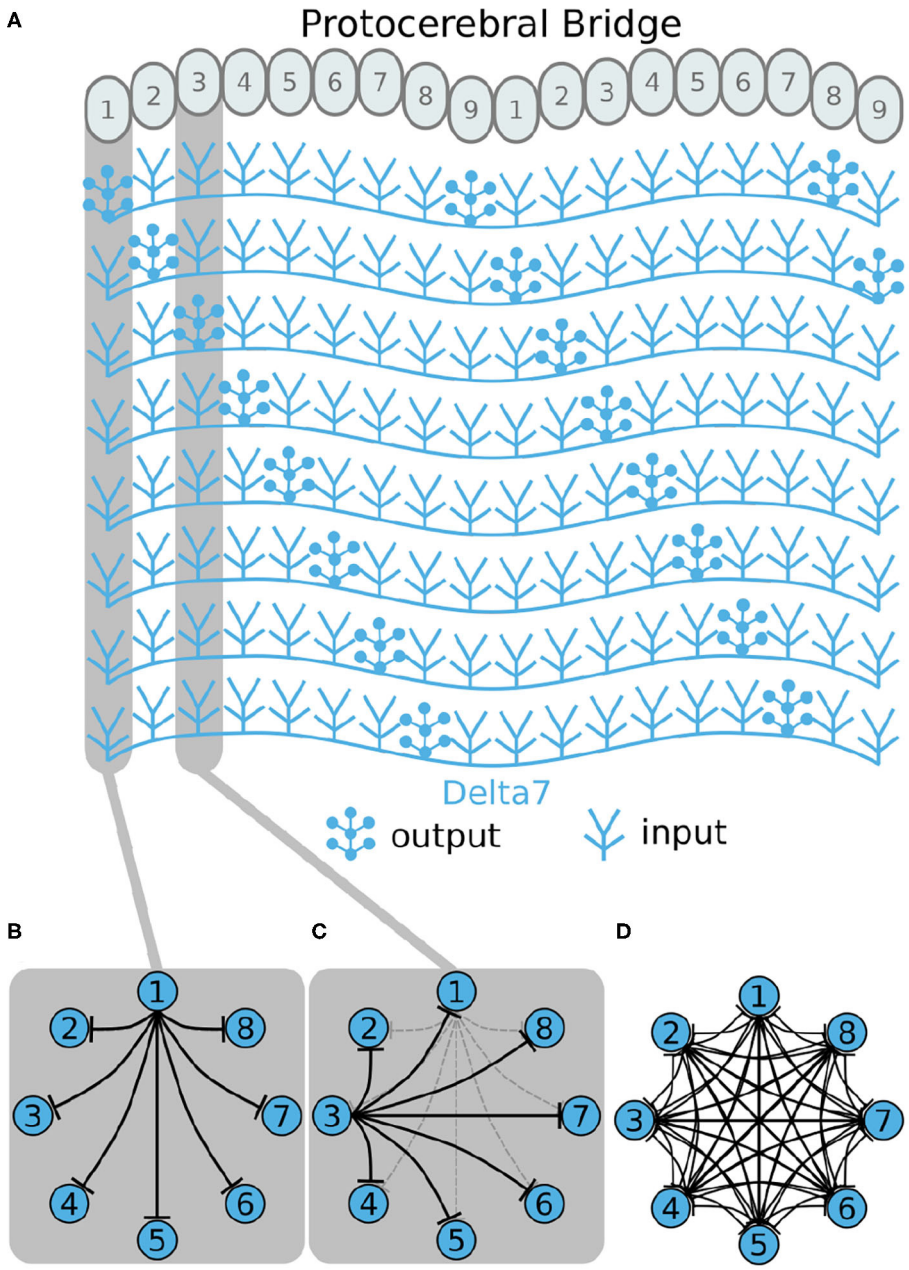


FIGURE 2 | Effective connectivity of the inhibitory (Delta7) neurons. **(A)** Schematic depiction of how the eight Delta7 neuron types innervate the glomeruli of the protocerebral bridge. **(B,C)** The effective connectivity in the first and third glomeruli is depicted as directed graphs with discs representing neurons and lines inhibitory synapses between them. **(D)** The effective neuronal circuit connectivity of the eight Delta7 neurons. Each Delta7 neuron inhibits all other Delta7 neurons resulting in an all-to-all inhibition pattern.

2018). Output terminal domains of each neuron are separated by seven glomeruli (**Figure 2A**).

Each Delta7 neuron forms synapses with all other Delta7 neurons in two or three glomeruli along the PB (**Figure 2A**). Starting from the first glomerulus (glomerulus 1) in the left hemisphere, we see that one neuron has output terminals while the other seven neurons have input terminals; we add arrows in the directed graph to indicate which neurons receive input synapses from this first neuron (**Figure 2B**). This can be systematically repeated for the synapses in each glomerulus from left to right (glomeruli 1–8 in the left hemisphere). Then proceeding to glomerulus 9 and through 1–9 in the right hemisphere, we observe that the same connectivity pattern repeats. Since we are interested only in the effective connectivity, we do not preserve information about repeated connections between neurons in the final directed graph (**Figure 2D**). As

such, the two or three synaptic connections between pairs of Delta7 neurons are reduced to one single connection between each pair of nodes in the simplified effective circuit in **Figure 2D**. This reduction to the essential connectivity is crucial for gaining an understanding of the circuit structure. The directed graph depiction of the circuit makes it evident that each Delta7 neuron forms synapses with and inhibits all other Delta7 neurons. Therefore, a uniform, all-to-all, inhibition pattern is revealed.

2.1.2. Excitatory Circuit

Now, we will walk through the steps of reverse engineering the excitatory portion of the circuit. The excitatory portion of the circuit consists of three classes of neurons, the P-EG, E-PG, and P-EN neurons. The synaptic terminals of each neuron are confined to one glomerulus of the PB (**Figures 3–5**). In the EB, the synaptic terminals of E-PG neurons are constrained in single wedges (half tiles) while the synaptic terminals of P-EN and P-EG neurons extend to whole tiles. In our schematic of the anatomy (see **Figure 3**), the glomeruli are numbered 1–9, left-to-right, in each PB hemisphere, and the EB tiles are numbered 1–8 clockwise. The neurons are numbered by the glomerulus they innervate, e.g., P-EN_1. For brevity, a tile numbered *n* is denoted as T*n* and a glomerulus numbered *m* as G*m*.

According to calcium and electrophysiology recordings (Turner-Evans et al., 2017), there must be one activity "bump" emerging around the EB and two activity "bumps" along the PB, one in each hemisphere. Preliminary simulation of the neuronal circuit, using the connectivity matrix derived from the neuronal anatomy, confirmed that the two activity "bumps" are centered around neurons innervating identically numbered PB glomeruli. That is, if the one activity "bump" is centered around G5 in the left hemisphere, the second activity "bump" will be centered around G5 in the right hemisphere. This observation about function will be used here in order to simplify the circuit structure and derive the effective connectivity.

Under the aforementioned numbering scheme, each P-EG neuron has synaptic terminals in identically numbered PB glomeruli and EB tiles (**Figure 3A**). That is, P-EG_1 has synaptic terminals in tile T1 and glomeruli G1 in both hemispheres of the PB. Since the two P-EG_1 neurons receive equal input in glomeruli G1, in both hemispheres, and connect to the same EB tile, T1, they are replaced with a single effective functional unit, as shown at the bottom of panel **Figure 3A**, in the form of a directed graph. The same reasoning can be repeated for the next pair of neurons, P-EG_2, that connect glomeruli G2 to tile T2 (**Figure 3B**). **Figure 3C** shows the resulting effective circuit if these steps are followed all the way until P-EG_8, the pair of neurons connecting glomeruli G8 to tile T8. Finally, we consider the last pair of neurons, P-EG_9; this pair of neurons connects glomeruli G9 to tile T1, breaking the pattern. These neurons are represented with a new node in the graph, but as it will become apparent in the next paragraph, the P-EG_9 neurons receive the same input as P-EG_1 neurons allowing us to combine them.

A second class of cells, E-PG neurons, also have synaptic terminals in equally numbered EB tiles and PB glomeruli, following a similar pattern with the P-EG neurons but with their input and output terminals on opposite ends (**Figure 4**). Pairs of these neurons can again be replaced by single equivalent neuronal units because they receive input from the same EB tile and innervate equally numbered glomeruli in both hemispheres. The first pair of E-PG neurons, E-PG_1, receive input in tile T1 and provide output in glomeruli G1 in both hemispheres (**Figure 4A**). Adding the corresponding connections results in the directed graph shown at the bottom of **Figure 4A**. Repeating the same for neurons E-PG_2 to E-PG_8 results in the graph shown in **Figure 4C**. Here, again there is a ninth pair of cells, the E-PG_9 neurons, that connect T1 to G9 in both hemispheres. These neurons receive the same input signal as E-PG_1 neurons but provide output to neurons in G9 instead of G1. Therefore, P-EG_1 and P-EG_9 neurons receive the same signal, in glomeruli G1 and G9, and provide the same output to both E-PG_1 and E-PG_9 neurons, as mentioned in the previous paragraph. This allows us to combine the P-EG_1 and P-EG_9 neurons into one single unit in the graph of **Figure 4D**.

Unlike the P-EG and E-PG neurons, the P-EN neurons do not innervate the two middlemost glomeruli (G9 in the left hemisphere and G1 in the right hemisphere, Wolff et al., 2015). There are, therefore, eight pairs of P-EN neurons, spanning glomeruli 1–8 in the left hemisphere and 2–9 in the right hemisphere. P-EN_2 through P-EN_8 form pairs connecting equally numbered glomeruli to two different EB tiles, one shifted to the left and one to the right, i.e., P-EN_2 would connect glomeruli G2 to tiles T1 and T3 (**Figure 5B**), P-EN_3 would connect glomeruli G3 to tiles T2 and T4, etc. P-EN_2 neurons form synapses with E-PG_1 neurons in T1 and E-PG_3 neurons in T3, which would innervate glomeruli G1 and G3, respectively. The exceptions in this pattern are the two P-EN neurons receiving input from the outermost glomeruli of the PB, P-EN_1 and P-EN_9. P-EN_1 is unpaired and connects G1 of the left hemisphere to T2 (**Figure 5A**). P-EN_9 is also unpaired and connects G9 of the right hemisphere to T8 (**Figure 5D**). Since P-EN_1 and P-EN_9 receive the same input from E-PG_1 and E-PG_9 neurons, they constitute a pair closing the ring, as shown in **Figure 5D**. In the directed graphs, each pair of P-EN neurons is preserved as two overlapped discs because P-EN neurons not only receive common input in the glomeruli but may also receive differential angular velocity input depending on which PB hemisphere they innervate (Turner-Evans et al., 2017).

It becomes apparent from **Figure 5D** that the E-PG neurons provide input to the P-EN and P-EG neurons, with P-EG neurons forming recurrent synapses back to E-PG neurons. P-EN neurons provide input to E-PG neurons with a shift of one octant to the left or right.

2.1.3. Overall Circuit

In each PB glomerulus, the inhibitory Delta7 neurons form synapses with the three types of excitatory neurons. **Figure 6** shows the interaction of the excitatory and inhibitory portions of the circuit. Each Delta7 neuron makes inhibitory synapses to P-EG and P-EN neurons, as well to all other Delta7 neurons (**Figures 6A,B**). Due to their projection patterns, the Delta7 neurons provide uniform inhibition to all eight octants of the circuit, while E-PG neurons provide input to all Delta7 neurons (**Figures 6C,D**). For drawing the graphs in **Figure 6**,

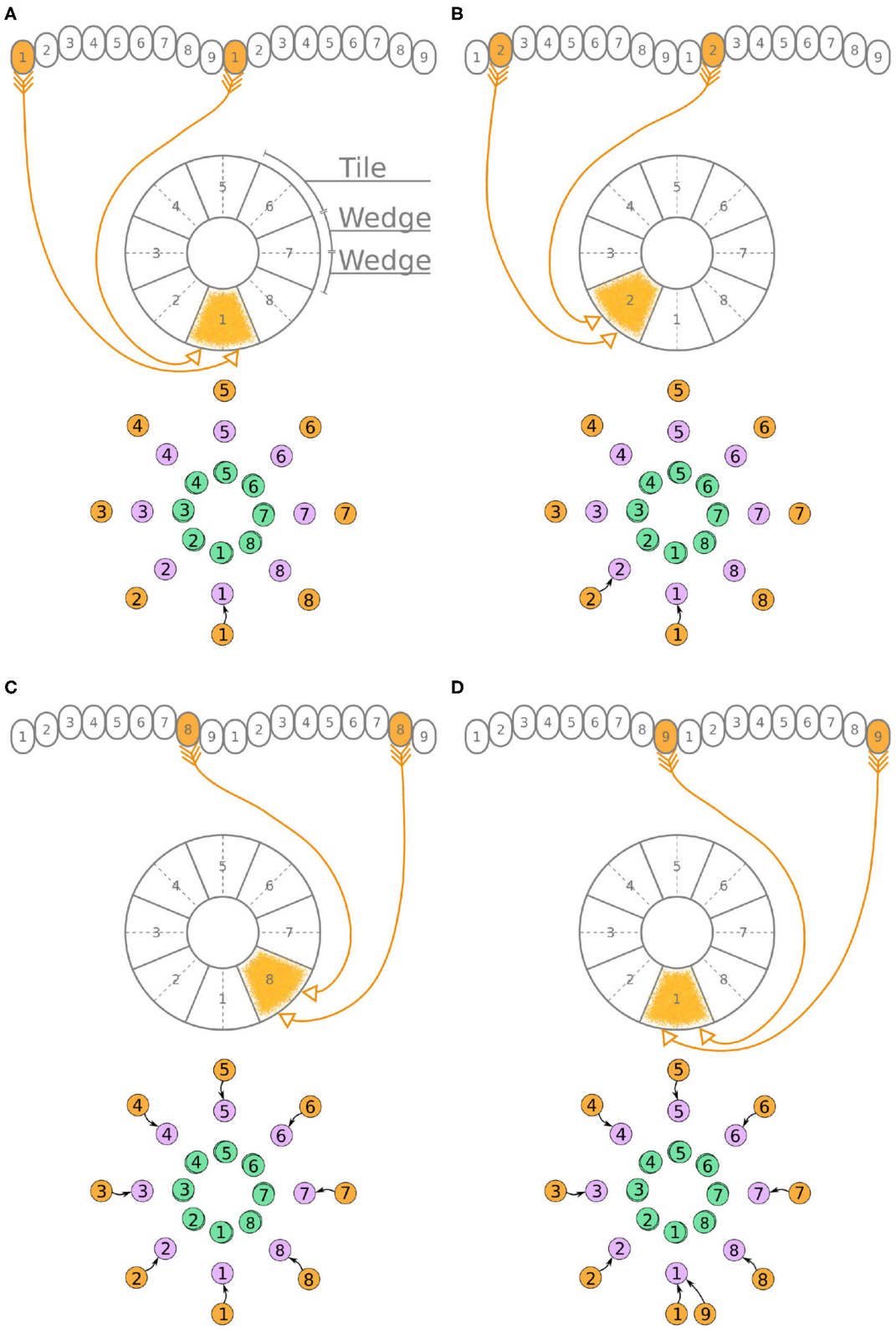


FIGURE 3 | (A–D) The connectivity pattern of P-EG neurons of the fruit fly. The top of each panel shows the connectivity pattern of a pair of P-EG neurons with their synaptic domains and connectivity patterns (see main text for detailed description). The bottom of each panel depicts the effective connectivity of the circuit as a directed graph. In the top portion of the panels each arrow represents a neuron. In the bottom portions of the panels, colored discs represent neurons and arrows represent synaptic connections.

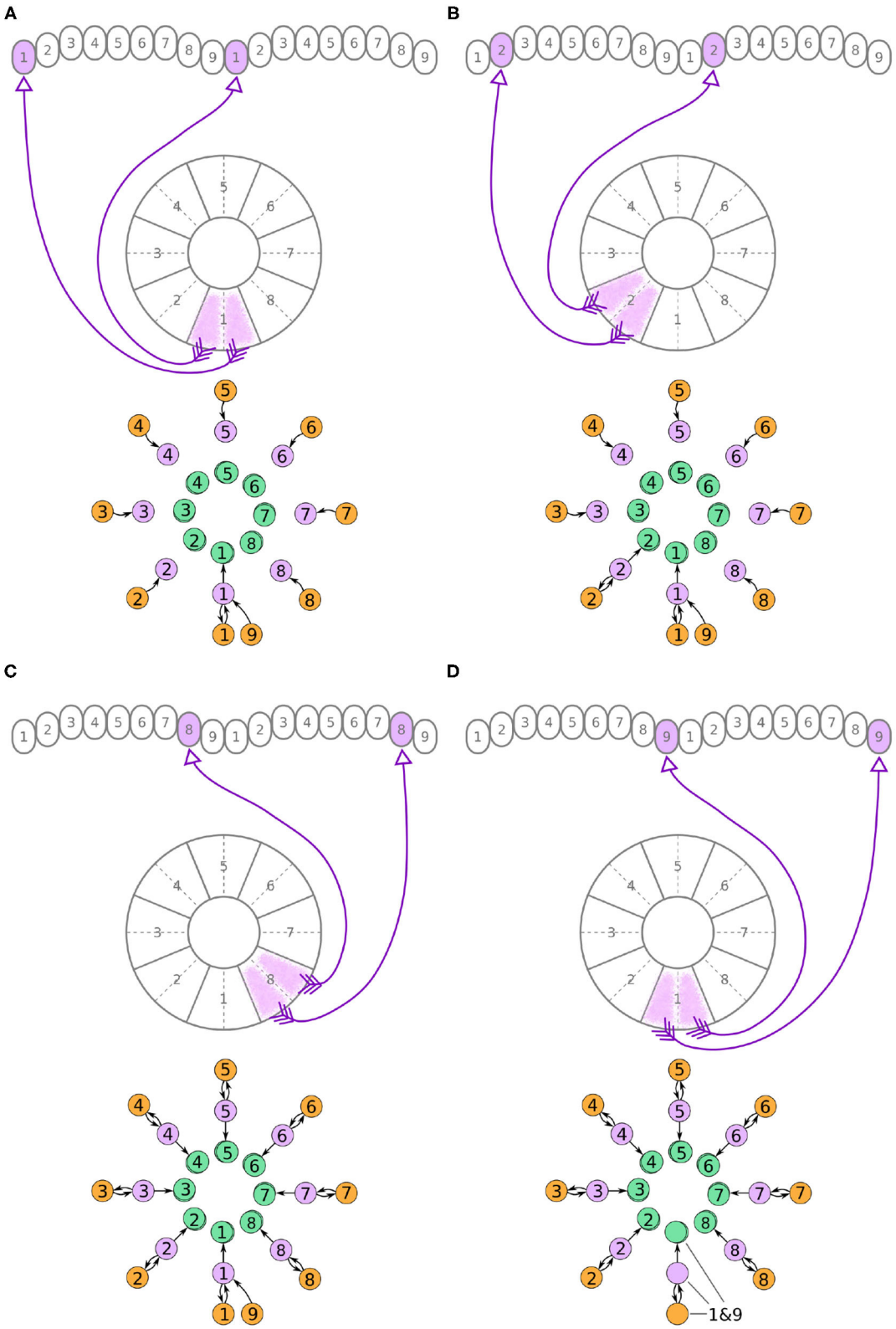


FIGURE 4 | (A–D) The connectivity pattern of the E-PG neurons of the fruit fly. The top of each panel shows the connectivity pattern of pairs of E-PG neurons with their synaptic domains and connectivity patterns (see main text for detailed description). The bottom of each panel depicts the effective connectivity of the circuit as a directed graph. In the top portion of the panels each arrow represents a neuron. In the bottom portions of the panels, colored discs represent neurons and arrows represent synaptic connections.

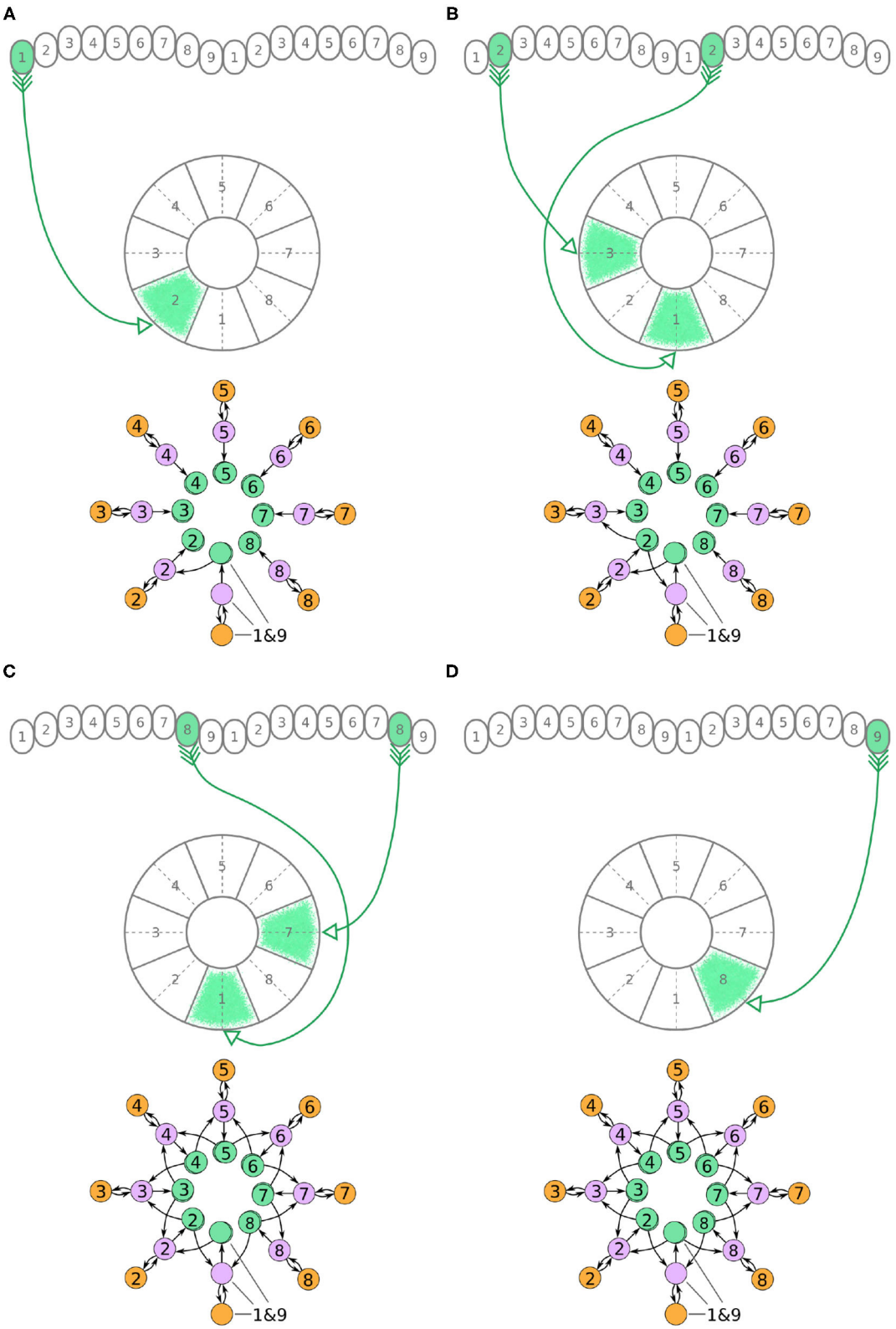


FIGURE 5 | (A–D) The connectivity pattern of P-EN neurons of the fruit fly. The top of each panel shows examples of P-EN neurons with their synaptic domains and connectivity patterns (see main text for detailed description). The bottom of each panel depicts the effective connectivity of the circuit as a directed graph. In the top portion of the panels each arrow represents a neuron. In the bottom portions of the panels, colored discs represent neurons and arrows represent synaptic connections.

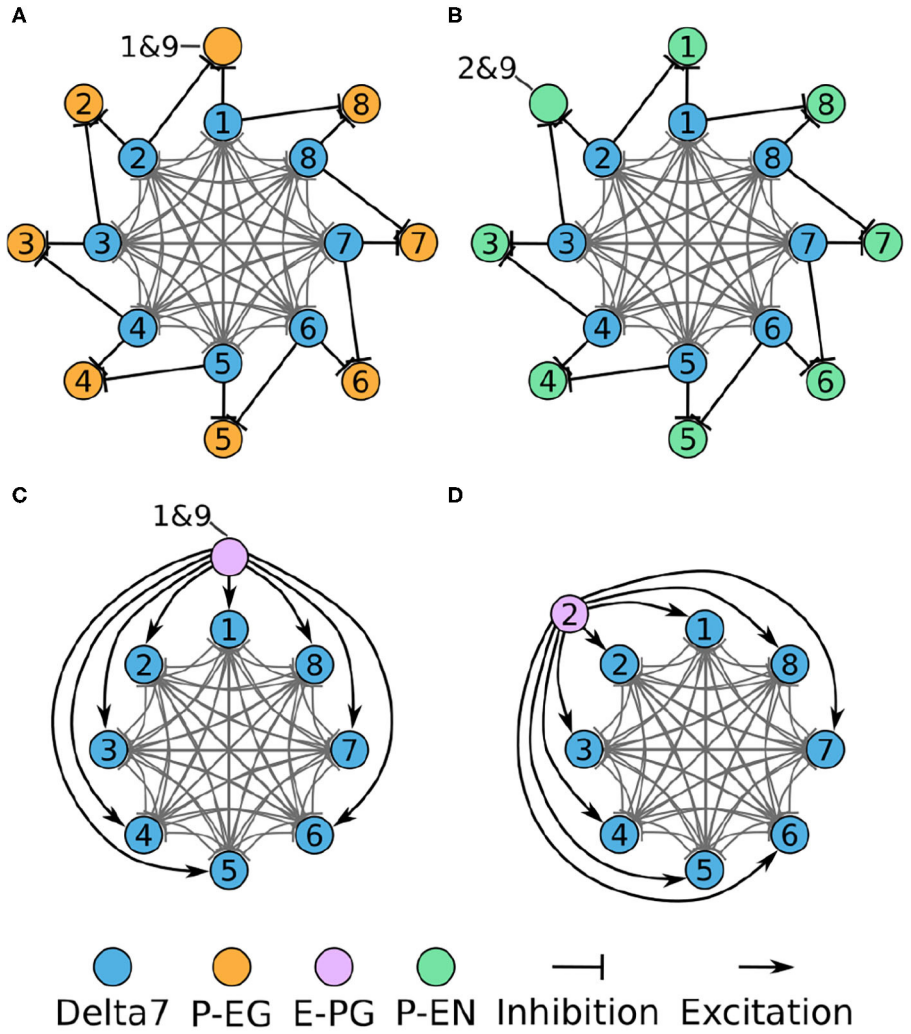


FIGURE 6 | Connectivity of combined excitatory and inhibitory portions of the circuit. Each colored disc represents one or more neurons with the lines representing synaptic connections. **(A)** The connectivity pattern of the Delta7 neurons with the P-EG neurons. **(B)** The connectivity pattern of the Delta7 neurons with the P-EN neurons. **(C,D)** The connectivity pattern of E-PG neurons ($E\text{-}PG_{1\&9}$ and $E\text{-}PG_{2}$). The other E-PG neurons follow the same connectivity pattern rotated around the Delta7 neurons. Each pair of E-PG neurons excites all Delta7 neurons.

Figures 2A, **3**–**5** were revisited and the connections within each glomerulus were added in the graphs.

The resulting directed graph representation removed the details about the anatomical organization of the EB and the PB while preserving the effective connectivity of the circuit. This analysis revealed that even though the PB is organized in nine glomeruli in each hemisphere, the effective circuit has an eight-fold radial symmetry. This is because the E-PG and P-EG neurons innervating the PB glomeruli G1 and G9, in both hemispheres, have synaptic domains in the same EB tile, T1. This aggregation of synaptic connections between the edges of the PB and T1, results in the closing of the ring between octants 1 and 8 (**Figure 5D**). The ring topology of the circuit reveals the interaction between components and is indicative of its function.

2.2. Computational Model

Now that we have reverse engineered the circuit structure, we can use simulations to investigate its function and corroborate the role of its components. To this end, a spiking neuron model of the derived circuit was implemented using the connectivity matrix and utilizing leaky integrate and fire neuron models with refractory period (section 4). Since neurophysiological evidence suggests a ring attractor resembling function and the effective circuit structure has the topology and necessary elements for a ring attractor, it was decided to impose the constraint that the circuit should function as a ring attractor. Using this constraint, an optimization algorithm was used to search for synaptic weights that result in a working ring attractor (section 4). The activity "bump" location was set by a heading stimulus provided as incoming spiking activity directly to the E-PG

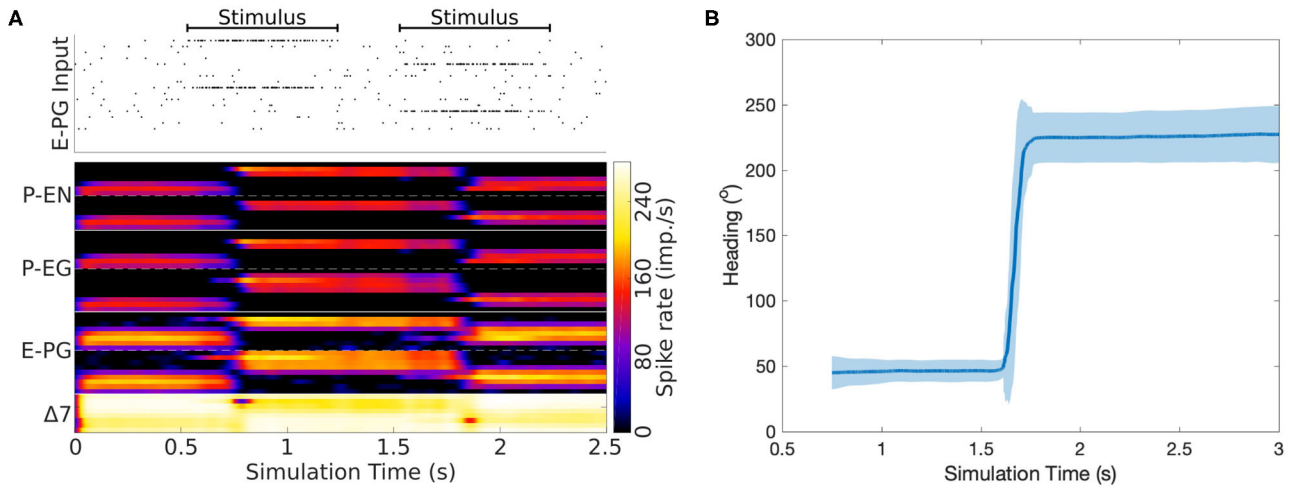


FIGURE 7 | Response of the ring attractor to abrupt changes of stimulus azimuth. **(A)** The spike raster plot on top shows the stimulus provided to the E-PG neurons. The lower part of the plot shows in color coding the spiking rate activity of each neuron in the circuit. At 0.5 s an incoming stimulus sets the initial attractor state of the ring attractor. A "darkness" period of no stimulus follows, during which the "bump" of activity is maintained at the same location. Then a second stimulus, corresponding to a sudden change of heading by 180^o, is provided, producing a sudden change in the position of the "bump," with this new location then maintained after the stimulus is removed. The order of recorded neurons is the same as shown in the connectivity matrix (**Figure 12**). **(B)** The mean activity "bump" heading and corresponding standard deviation across time when the ring attractor is stimulated with a step change of heading (80 trials).

neurons, corresponding to input from Ring neurons (Young and Armstrong, 2010). This heading input mapped the position of a visual cue, or retinotopic landmark position (Seelig and Jayaraman, 2015), around the animal to higher firing rates of E-PG neurons in the corresponding tile of the EB. The neuronal parameters were set to values consistent with evidence from measurements in *D. melanogaster*, as described in section 4. **Figure 7** shows examples of neuronal activity in the simulated ring attractor circuit with the activity "bump" transitioning from one attractor state to another in response to a change of the stimulus azimuth.

2.3. Situated Agent Behavior

The stimulus used in the preceding simulation was a step function of time, but a real fruit fly or robot would not perform instantaneous turns between heading directions; instead, they would exhibit smoother transitions between headings and a generally variable angular velocity over time. It is, therefore, important to characterize the circuit's performance in such a more natural scenario. For this reason, the flight trajectory of a real fruit fly was next used to simulate an agent turning with respect to a visual landmark. The fruit fly's heading over time was extracted from such a flight trajectory and was used to generate the time series of headings the agent adopts.

Figure 8A shows the motion trajectory of a fruit fly flying in a circular arena (Tammero and Dickinson, 2002; **Figure 2**). From the power spectral density plot of the heading over time, we can see that the fruit fly's heading signal has a main period of 1.092 s, corresponding to the fruit fly completing a full rotation around the arena in approximately 1 s (spectral peak at 0.916 Hz in **Figure 8B**). This was confirmed with calculation of the auto-covariance that produced a mean period of 1.087 s.

The visual landmark's azimuth with respect to the agent was retinotopically mapped to the E-PG neurons around the ring attractor (section 4). The correspondence between the heading of the agent and the heading encoded by the ring attractor circuit is shown in **Figure 8C**. The ring attractor tracked the agent's heading with an average lag of 100 ms. The exact phase lag depended on the frequency component of the signal, with a trend for higher frequencies—faster heading changes—resulting in increased lag (see bottom plot of **Figure 8B**). This is an expected effect because neurons have non-zero time constants and response times.

Overall, even though the heading encoded by the ring attractor accumulated error during fast turns of the agent, it caught up with the actual heading as soon as the agent's angular velocity was reduced (**Figure 8C**). This effect is due to the ring attractor circuit being continually driven by the stimulus' azimuthal position, so if given enough time to respond, the circuit state is readjusted to the stimulus position. It becomes apparent with this situated agent simulation that even though the agent's heading may change faster than the circuit's ability to track it, as soon as the agent slows down, the visual cue input corrects the location of the activity "bump" (**Figure 8C**).

2.4. Role of Circuit Elements

Now that we have both the underlying circuit structure and its computational model, we can draw hypotheses and ask pointed questions about the role of each circuit component. We can artificially manipulate the circuit by removing or replacing functional elements in order to study their effect on circuit function. We recently used this method to investigate the stability

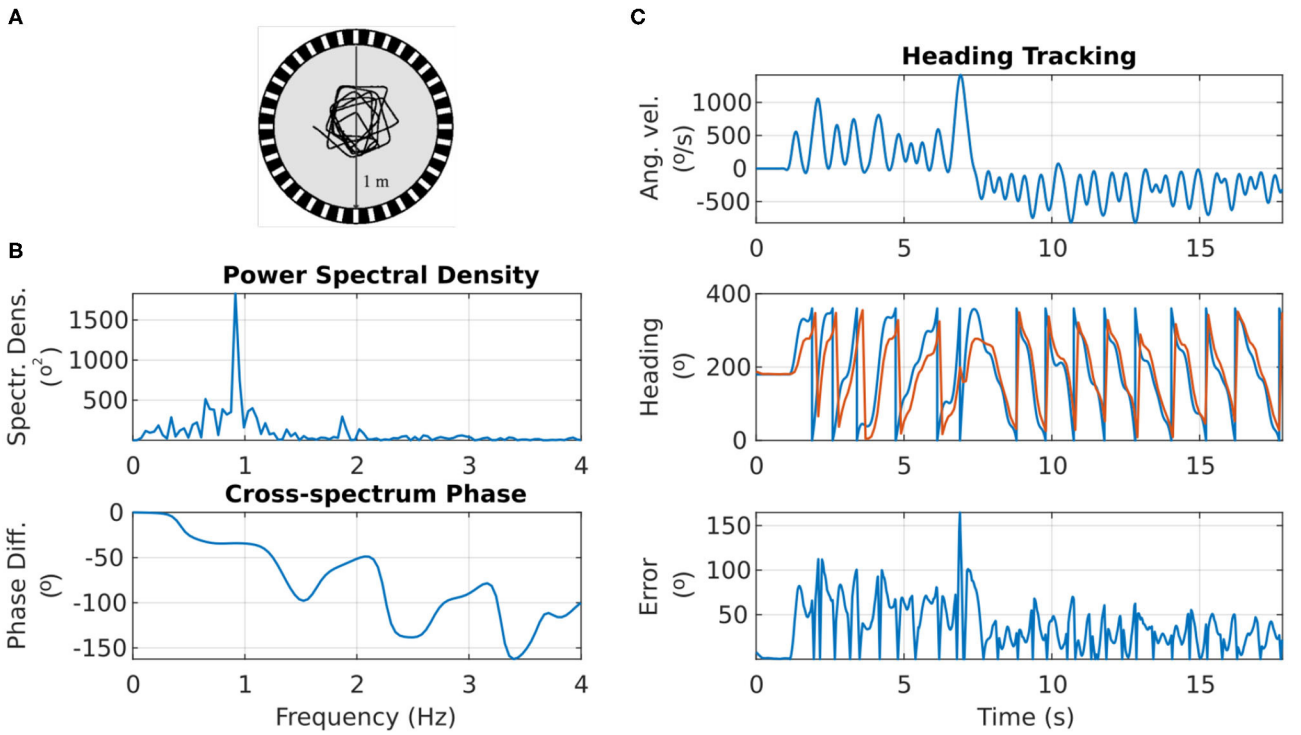


FIGURE 8 | Response of the ring attractor activity "bump" to the heading changes of a real fruit fly. **(A)** Top view of the trajectory of a fruit fly flying in a circular arena (Modified from Tammero and Dickinson, 2002). **(B)** Top: the spectral content of the fruit fly heading progress over time. Bottom: the phase difference between the fruit fly heading and the heading tracked by the ring attractor across frequency components. **(C)** Top: the angular velocity of the fruit fly over time. Middle: the heading of the fruit fly (in blue) and the azimuth of the activity "bump" around the ring attractor (in red). Bottom: the heading error (absolute difference) between the fruit fly heading and the ring attractor tracked heading.

of the activity "bump" in the absence of stimulus (Pisokas et al., 2020). We extend this approach here and investigate the circuit's performance as part of a situated agent that turns with respect to a visual cue.

Figure 9 shows the effect of heterogeneity of synaptic weights on the ability of the circuit to track the agent's heading when turning with respect to a visual cue. The ability to accurately track the agent's heading deteriorates with increasing heterogeneity (additive Gaussian noise) of synaptic weights.

Furthermore, when the circuit is driven by heading stimulus, it is significantly more tolerant of heterogeneity in neuronal membrane conductance than in membrane capacitance (**Figure 10**). The circuit can successfully track the agent's heading even when the membrane conductance deviates 50% away from its nominal value.

Next, we investigate the effect of heterogeneity introduced in different neuron synapses. While Pisokas et al. (2020) found that the P-EG neurons enhance the stability of the activity "bump," in **Figure 11A** we see that the ability of the activity "bump" to successfully track the agent's heading, when the circuit is driven by heading stimulus, is unaffected by variation of the P-EG to E-PG synaptic weights. The ring attractor successfully tracks the agent's heading even if the P-EG neurons are completely silenced. This means that the P-EG neurons play an important role in maintaining a stable heading when no stimulus is provided but are not necessary when such a heading stimulus is present. Whether the inclusion of these neurons is justified in a particular ring attractor design would therefore depend on the operational environment and the agent's behavioral repertoire.

We can observe that the circuit is more sensitive to variations in the E-PG to P-EN synapses than variations of the P-EN to E-PG synapses (**Figure 11A**). The circuit is also sensitive to heterogeneity introduced in the inhibitory synapses from Delta7 neurons to P-EG and P-EN neurons since inhibition of excitatory neurons is an essential aspect of a ring attractor circuit for the emergence of an activity "bump" (**Figure 11B**).

However, the circuit is tolerant to variations of the input weights of Delta7 neurons (**Figure 11B**). This is because Delta7 neurons reciprocally synapse with each other, resulting in similar spiking activity in all of them due to averaging out the effect of synaptic weight variation.

Such insights drawn from observations about the ring attractor found in the brain of the fruit fly can be incorporated in building improved ring attractors with applications in robotics as well as in developing theoretical models. The ability to manipulate the circuit in robotic simulations can be used

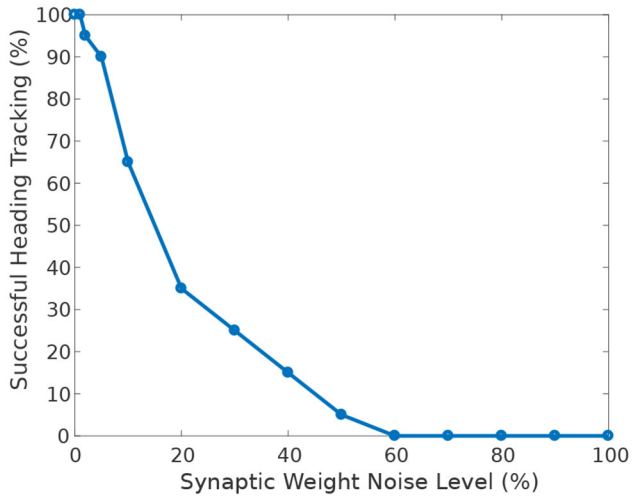


FIGURE 9 | Effect of synaptic weights heterogeneity on heading tracking performance. The ability of the ring attractor to track the heading of a simulated robot, replicating the turns of a real fruit fly, deteriorates as function of synaptic noise. Synaptic noise was introduced by adding values drawn from the Gaussian distribution to the nominal values of all synaptic weights.

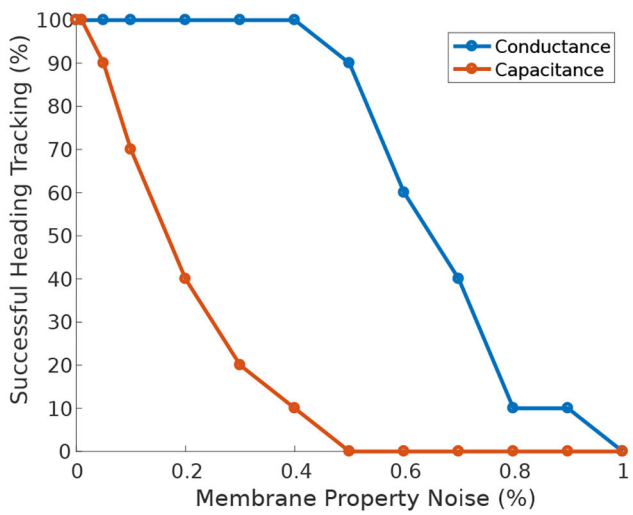


FIGURE 10 | Effect of membrane properties heterogeneity on heading tracking performance. The ability of the ring attractor to track the heading of a simulated robot, replicating the turns of a real fruit fly, deteriorates as function of Gaussian noise added to the neuronal membrane conductance and capacitance.

for testing hypotheses both at the neuron level and at the system level.

3. DISCUSSION

The increasing availability of detail about neuronal structure, particularly in invertebrate brains, raises the possibility to simulate complete circuits. However, while directly implementing and simulating a biological neuronal circuit model allows us to understand the computation performed by it and to potentially derive its transfer function, it does not necessarily provide us with a real mechanistic understanding of its principle of operation and how its components interact. Reverse engineering the neuronal circuit can provide a real mechanistic understanding of the underlying principles of the computational structure. Such a mechanistic understanding is necessary for transfer to robotic technology because it would allow engineers to adapt the design to each application's particular needs.

An intriguing challenge was posed by Jonas and Kording (2017) who asked whether the tools and methods available to a neuroscientist would allow understanding of a microprocessor. Here, I have used reverse engineering techniques, borrowed from engineering, to reverse engineer the neuronal circuit that is encoding the head direction of the fruit fly. I derived the effective topological structure of the circuit and then determined (through optimization) the synaptic weights that would allow it to function as a ring attractor, mimicking the dynamics of the biological circuit. This illustrates that reverse engineering of a neuronal circuit with fewer than a hundred neurons is feasible.

It is worth noting that the circuit studied here, even though highly recurrent, has a regular structure that facilitates the systematic application of the presented procedure. It remains to be seen how this approach would need to be augmented in order to be tractably applied to circuits exhibiting less regularity. This highlights the need to develop tools that would assist the systematic analysis of larger neuronal circuits.

The availability of detailed neuron-level anatomical data and neuronal recordings from behaving animals in combination with computational simulations enabled the analysis and study of the circuit's organization and function. This level of detailed information is currently available for a few species, mainly insects. The fruit fly is one of these, allowing the application of the method to it. As data become available for more species and brain areas, we could have the opportunity to analyze more circuit structures and their function.

3.1. Assumptions and Simplifications

As any model, the present model is a simplification of the neuronal circuit found in the fruit fly brain; therefore, it is important to outline the assumptions made. The presented analysis is based on data collected using light microscopy (Wolff et al., 2015; Wolff and Rubin, 2018). Neurons with input and output synaptic terminals occupying the same volume were assumed to form synapses. Analysis of recently published electron microscopy data will allow more definite determination of synaptic connections between neurons and lead to more accurate models. Furthermore, all neurons in the model were assumed to have the same nominal biophysical property values. Of course, this will not be the case in the actual animals, but currently, there is no adequate data available about the biophysical properties of the individual neurons included in the model.

It was also assumed that Delta7 neurons have a uniform distribution of input terminals along the PB. Imaging of Delta7

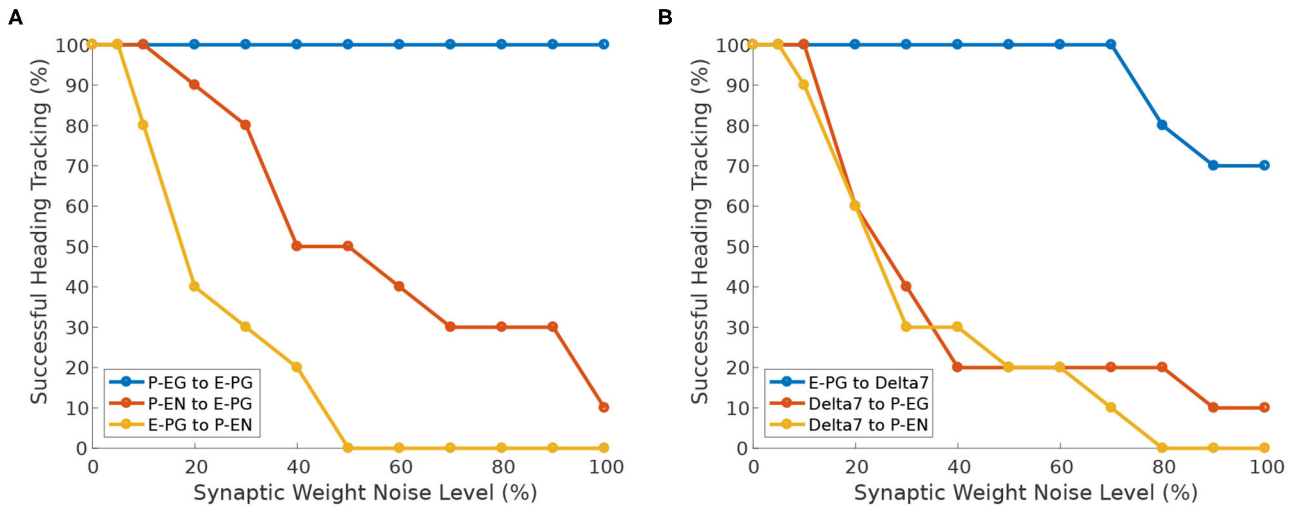


FIGURE 11 | Effect of synaptic weights heterogeneity on heading tracking performance. The ability of the ring attractor to track the heading of a simulated robot, replicating the turns of a real fruit fly, as function of Gaussian noise added to synaptic weights for different classes of neuron synapses. **(A)** Effect of Gaussian noise on excitatory synapses. **(B)** Effect of Gaussian noise on synapses with Delta7 neurons on the presynaptic or postsynaptic side.

neurons suggests a subtle variation of dendritic density along the PB, but it is yet unclear how this variation might relate to synaptic density and efficacy. Therefore, the simplifying assumption that the synaptic efficacy of Delta7 neurons along the PB is uniform was made. It was also assumed that neuronal terminals are clearly delineated and confined within the volumes of glomeruli and tiles. However, in some cases, stray terminals are known to sprout out to neighboring tiles of the EB (Turner-Evans et al., 2020). Such cross-innervation and interaction of EB volumes might have consequences for the connectivity of the circuit, potentially allowing a smoother transition of the activity "bump" between circuit octants. Future work will build upon the core circuit analyzed here and incorporate more circuit detail based on new electron microscopy data.

Occasionally neurons have mixed input and output terminals within the same volume. Given the uncertainty in the identification of the type of synaptic terminals, in those cases, the predominant terminal type was used. Furthermore, the synaptic weights of each type of synapse were assumed to be identical across neurons. This is not expected to be the case in actual fruit flies, especially for the neurons innervating tile T1 of the EB. This tile is innervated by twice the number of E-PG and P-EG neurons as other tiles; thus, some modulation of synaptic efficacy is expected in this volume in order to maintain a functional radial symmetry in the circuit. Such synaptic efficacy variation is suggested by the fact that the volumes of the innermost glomeruli of the PB are smaller than those of the other glomeruli (Wolff et al., 2015). Future functional connectivity studies will allow further investigation of this aspect.

It should also be noted that the ring topology of the resulting circuit alone does suggest but does not prove a ring attractor function. Here, the prior observation of neurobiological studies that the circuit maintains an activity "bump" that tracks the heading of the animal was used to impose constraints in the search for synaptic weights. For simplifying the computational complexity of the search for synaptic weights, it was assumed that all synapses between each neuron pair type are identical. Had the computational complexity of the search not been an issue, it would have been preferable to optimize all synaptic weights as independent parameters because that would have potentially revealed alternative weight configurations satisfying the objective function.

3.2. Nature as Inspiration for Theory and Engineering

The presented analysis method allowed us to unravel that the underlying head direction circuit has an eight-fold radial structure forming a closed ring (Pisokas et al., 2020). Without reverse engineering of the neuronal circuit, we would not have been able to see this underlying circuit structure, especially because, even though there are eight tiles in the EB, the PB has nine glomeruli in each hemisphere. As the connectivity results in a closed ring, it is an important aspect of the circuit, allowing the activity "bump" to move around the ring as the agent changes heading.

Combining reverse engineering with simulations enabled the identification of circuit elements that differ in several ways from the "canonical" ring attractor described in earlier theoretical models (e.g., Amari, 1977; Skaggs et al., 1995; Zhang, 1996). The P-EG neurons are a novel element in a ring attractor, forming local feedback loops within each octant of the circuit (reciprocal synapses between P-EG and E-PG neurons). These local reciprocal connections increase the tolerance of the circuit to structural noise in the synaptic weights, hence reducing the drift of the activity "bump" when no stimulus is provided (Pisokas et al., 2020); however, they are not important if the stimulus can be assumed at all times. This circuit component will be a useful trick in the toolkit of neuromorphic circuit designers.

Another difference from textbook ring attractor circuits revealed by the presented analysis method is that the P-EN neurons, instead of functioning as mere input neurons, are also part of the lateral excitation circuit (Pisokas et al., 2020). These neurons provide lateral excitation to their two nearest neighbors. P-EN neurons' dual function suggests a more efficient use of neuronal resources compared with typical ring attractor models that use separate sets of neurons for providing the lateral excitation and for rotating the activity "bump" around the ring in response to angular velocity input. The architecture of the ring attractor circuit found in the fruit fly and its differences from classical ring attractor models can inspire the design of novel ring attractor architectures with increased stability and efficient use of neuronal resources, both valuable aspects for applications in neuromorphic hardware and neurorobotics.

Reverse engineering gives us a mechanistic understanding of the underlying circuit, while computational simulations give us the tools to study the circuit's performance without having an analytical description of the model. Combined reverse engineering and computational simulations are tools that enable us to isolate and manipulate components of the neuronal structure in order to study their role in whole circuit. The mechanistic understanding of how the circuit components interact allows us to infer the circuit behavior under regimes beyond those explicitly tested with simulations. Combining these two tools allows us to obtain a deep understanding of neuronal circuits and enables us to learn their principles of operation.

Furthermore, the approach illustrated here shows that simulating the circuit as part of a robotic agent reveals aspects of the circuit's function that are masked when studying the circuit in isolation. For example, we saw that even if the ring attractor's response time is not sufficient for keeping up with fast turns of the agent, as long as the agent does not constantly turn faster than the circuit's response capability, and the heading stimulus is available, the ring attractor can readjust to the correct heading. We also saw that the P-EG neurons' presence, while essential for the stability of the activity "bump" when no stimulus is available, is not important to the circuit's function when a heading stimulus is available. These findings highlight the importance of characterizing neuronal circuits as part of behaving agents.

The studied circuit appears to be an effective means for an animal to internally track its orientation with respect to its surroundings and in insects appears to be a core component of a variety of navigation behaviors spanning from long-range migration to local path integration. The continued study of the detailed anatomy of the insect brain provides an exciting opportunity for the further unraveling of this circuit's function that evolved to support complex adaptive behavior.

4. MATERIALS AND TOOLS

4.1. Neuronal Nomenclature

Throughout this paper, I refer to neurons using their short names for brevity. The correspondence between the nomenclature used here and in the literature is shown in **Table 1**.

4.2. Neuron Model

The computational models and simulations were based on the source code of Kakaria and de Bivort (2017). The neurons were modeled as leaky integrate and fire units with refractory period. The membrane potential of each neuron was modeled by the differential Equation (1).

$$\frac{dV_i}{dt} = \frac{1}{C_m}\left(\frac{V_0 - V_i}{R_m} + I_i + \sum_{j=1}^{N} M_{j,i} I_j\right) \tag{1}$$

where V_i is the membrane potential of neuron i, R_m the membrane resistance, C_m the membrane capacitance, I_i the external input current to neuron i, V_0 the resting potential, $M_{j,i}$ the network connectivity matrix, I_j the output current of each neuron in the circuit and N is the number of neurons.

The neuron properties were set to the same values as those used by Kakaria and de Bivort (2017). These values are consistent with evidence from measurements in *D. melanogaster*. C_m was set to $2nF$ and R_m to $10M\Omega$ for all neurons, assuming a surface area of $10^{-3}cm^2$ (Gouwens and Wilson, 2009). The resting potential V_0 was $-52mV$ for all neurons (Rohrbough and Broadie, 2002; Sheeba et al., 2008) and the action potential threshold was -45 mV (Gouwens and Wilson, 2009). The action potential template was defined as (Kakaria and de Bivort, 2017):

$$V(t) = \begin{cases} V_{thr} + (V_{max} - V_{thr})\frac{\mathcal{N}\left(\frac{t_{tp}}{2}, \left(\frac{t_{AP}}{2}\right)^2\right) - \alpha_1}{\beta_1}, & \text{if } 0 \le t < \frac{t_{AP}}{2} \\ V_{min} + (V_{max} - V_{min})\frac{sin\left(\left(t - \frac{t_{AP}}{2}\right)\frac{2\pi}{t_{AP}} + \frac{\pi}{2}\right) + \gamma_1}{\delta_1}, & \text{if } \frac{t_{AP}}{2} \le t \le t_{AP} \end{cases} \tag{2}$$

When the membrane potential reached the threshold voltage V_{thr}, the action potential template was inserted in the recorded voltage time series. $V_{max} = 20$ mV is the peak voltage (Rohrbough and Broadie, 2002) and $V_{min} = -72$ mV is the undershoot potential (Nagel et al., 2015). $t_{AP} = 2$ ms is the duration of the action potential (Gouwens and Wilson, 2009; Gaudry et al., 2012). $\mathcal{N}(\mu, \sigma^2)$ is a Gaussian function with mean μ and standard deviation σ. α_1, β_1, γ_1, and δ_1 are normalization parameters for scaling the range of the Gaussian and the sinusoidal to [0,1]. No other action potentials were allowed during the template duration in effect producing a refractory period.

The postsynaptic current generated by the action potential was modeled as (Kakaria and de Bivort, 2017):

$$I(t) = \begin{cases} I_{PSC}\frac{sin\left(\frac{t\pi}{2} - \frac{\pi}{2}\right) + \alpha_2}{\beta_2}, & \text{if } 0 \le t < 2ms \\ I_{PSC}\frac{2^{-(t-2)/t_{PSC}} + \gamma_2}{\delta_2}, & \text{if } 2ms \le t \le 2ms + 7t_{PSC} \end{cases} \tag{3}$$

Excitatory and inhibitory postsynaptic currents were assumed to have the same magnitude but opposite signs. The parameters

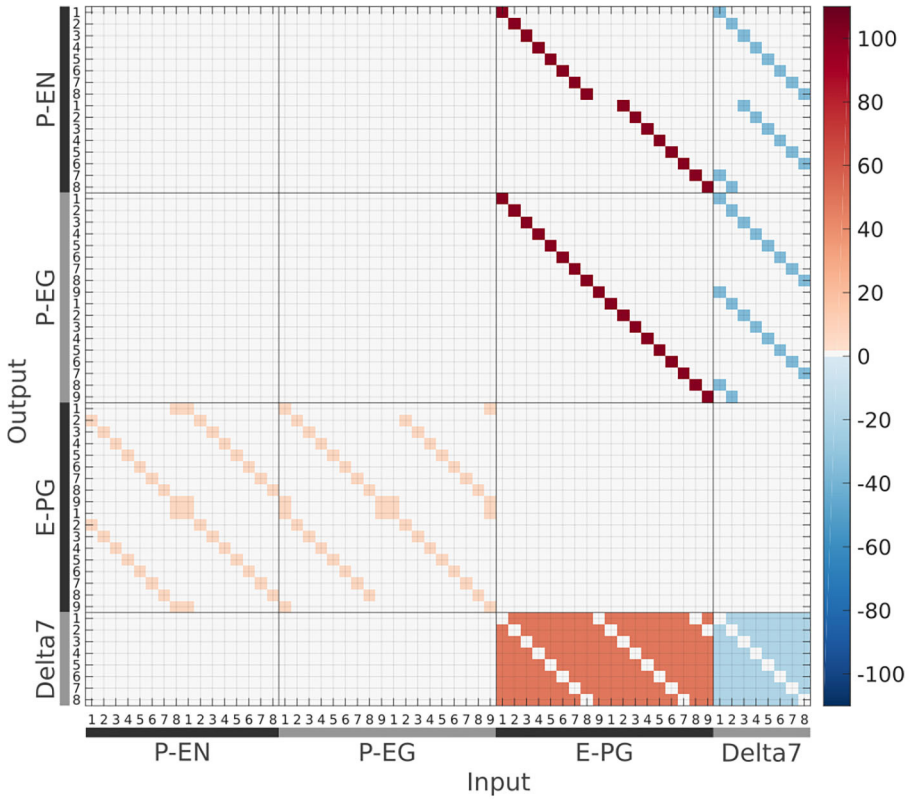


FIGURE 12 | The connectivity matrix derived by the neuronal projection data of the fruit fly *Drosophila melanogaster* (Wolff et al., 2015; Wolff and Rubin, 2018). Synaptic weight is denoted by color in units of postsynaptic current equivalents.

were set to $I_{PSC} = 5$ nA (Gaudry et al., 2012) and $t_{PSC} = 5$ ms (Gaudry et al., 2012). The postsynaptic current traces had duration $2ms + 7t_{PSC}$ (2 ms of rise time plus $7t_{PSC}$ of decay time). α_2, β_2, γ_2, and δ_2 are normalization constants so that the range of the sinusoidal and exponential terms is [0,1]. Our simulation code was derived from the source code published by Kakaria and de Bivort (2017). The simulations were implemented in Matlab using Euler's method with a simulation time step of $10^{-4}s$. The source code is available at https://github.com/johnpi/Frontiers_Neurorobotics_Pisokas_2020.

4.3. Neuronal Projections and Connectivity Matrix

The connectivity matrix of the circuit (**Figure 12**) has been inferred from anatomical data derived using light microscopy, with overlapping neuronal terminals assumed to form synapses between them (Wolff et al., 2015; Wolff and Rubin, 2018).

4.4. Stimuli

The heading stimulus was provided as incoming spiking activity directly to the E-PG neurons. The heading, visual cue azimuth (Seelig and Jayaraman, 2015) around the animal or agent, was encoded as higher firing rates supplied to E-PG neurons at the corresponding location around the EB ring (**Figure 13**). The heading stimulus followed spatially a von Mises distribution with mean equal to the azimuth of the stimulus and full width at half maximum (FWHM) of approximately 90°. This was converted to Poisson distributed spike trains by sampling from a Poisson distribution. The background neuronal activity level was set to 5 impulses/s and the maximum stimulus activity was set to the peak level of activity of the E-PG neurons in the neuronal population.

4.5. Selection of Synaptic Weights

The free parameters of the model were the synaptic weights. The synaptic weights connecting each class of neurons were assumed to be identical, e.g., all E-PG to P-EN synapses had identical weights. Therefore, there was one free parameter for each synaptic class. To reduce the computational complexity during optimization, the synaptic weights of E-PG to P-EN and P-EG were identical as were the synaptic weights of Delta7 to P-EN and P-EG. This was the minimum set of independent synaptic weights that resulted in working ring attractors. The synaptic weights were modeled as the number of I_{PSC} unit equivalents flowing to the postsynaptic neuron per action potential.

The simulated annealing and particle swarm optimization algorithms were used to search for synaptic weights that resulted in working ring attractors (Matlab Optimization Toolbox "simulannealbnd" and "particleswarm" functions). The objective

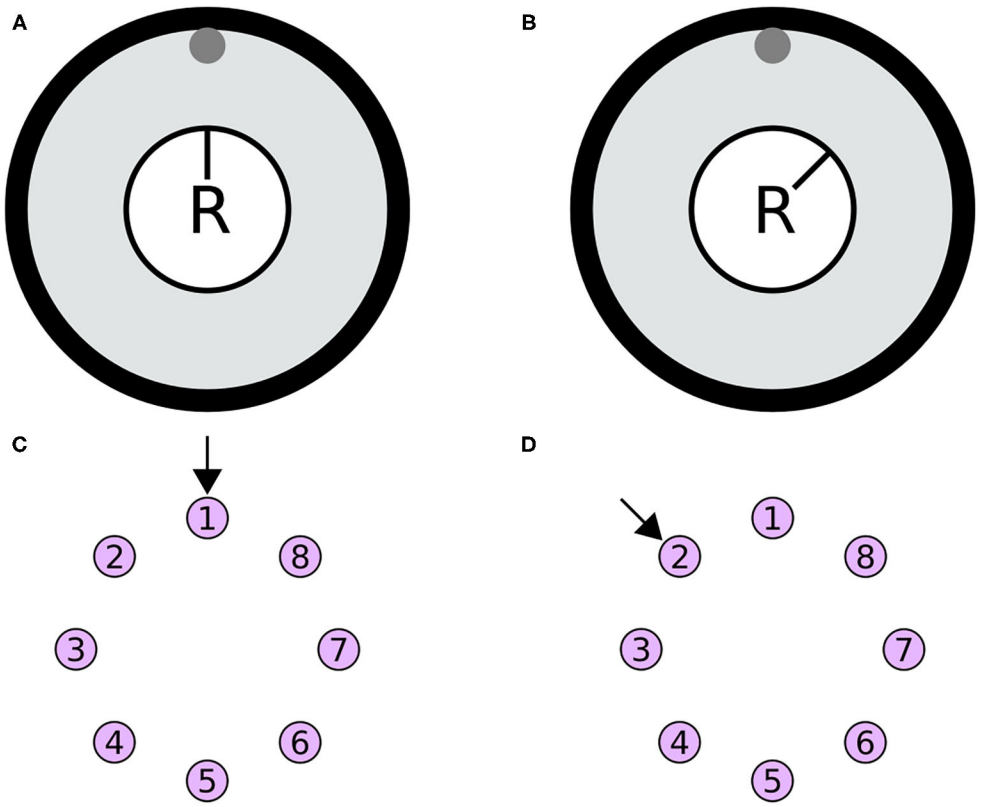


FIGURE 13 | Simulated agent and its environment. **(A,B)** The simulated agent (R) is located in the middle of its environment (large black circle), with a visual cue (gray circle at the top of the environment). During the simulation the heading of the agent changes. **(C,D)** The azimuth of the visual cue with respect to the agent is mapped retinotopically to the E-PG neurons around the ring attractor. The azimuthal location is converted to a von Mises activity distribution and then to Poisson distributed spikes to stimulate the E-PG neurons. **(A,B)** Two poses of the agent with respect to the visual cue. **(C,D)** The corresponding stimulation of the E-PG neurons around the ring attractor.

function optimized for solutions that produced an activity "bump" with a full width at half maximum (FWHM) of approximately 90° since this is the width that has been reported in fruit flies (Kim et al., 2017).

The objective function used to optimize the synaptic weights w_i was:

$$\begin{aligned}
\underset{\mathbf{w}}{\text{argmin}} \quad & 4(\epsilon_{H1}(\mathbf{w}) + \epsilon_{H2}(\mathbf{w})) + \epsilon_{W1}(\mathbf{w}) + \epsilon_{W2}(\mathbf{w}) + Np_0(\mathbf{w}) \\
\text{s. t.} \quad & \epsilon_{H1}(\mathbf{w}) = \frac{|H_d(t_1) - H_a(\mathbf{w}, t_1)|}{360^\circ} \\
& \epsilon_{H2}(\mathbf{w}) = \frac{|H_d(t_2) - H_a(\mathbf{w}, t_2)|}{360^\circ} \\
& \epsilon_{W1}(\mathbf{w}) = \frac{|90^\circ - W_a(\mathbf{w}, t_1)|}{360^\circ} \\
& \epsilon_{W2}(\mathbf{w}) = \frac{|90^\circ - W_a(\mathbf{w}, t_2)|}{360^\circ} \\
& p_0(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(e^{-|w_i|})^2
\end{aligned}$$

$$\begin{aligned}
0 \leq w_1 \leq 100 \\
0 \leq w_2 \leq 100 \\
0 \leq w_3 \leq 100 \\
-100 \leq w_4 \leq 0 \\
-100 \leq w_5 \leq 0
\end{aligned} \tag{4}$$

Where ϵ_{H1}, ϵ_{H2}, ϵ_{W1}, and ϵ_{W2} are the error factors measured as deviations from the desired values. $H_d(t)$ is the desired activity "bump" heading at time t, while $H_a(\mathbf{w}, t)$ is the actual activity "bump" heading at time t given a model with synaptic weights $\mathbf{w}$. $W_a(\mathbf{w}, t)$ is the actual width of the activity "bump" at time t (measured as the full width at half maximum). p_0 is used to penalize synaptic weights that are too close to 0 and N is the number of synaptic weights w_i. The constraints in 4 specify that the synapses with Delta7 neurons at their presynaptic side are inhibitory (negative) and all others are excitatory (positive). Excitatory synaptic weights were initialized with value 0.01 and inhibitory synaptic weights with value −0.01. During optimization, the model was simulated to search the space of synaptic weights. The objective function was used to

optimize the synaptic weights separately for the two models, the fruit fly model and the one without P-EG neurons. The optimized synaptic weight sets were manually tested to verify the results.

4.6. Sensitivity Analysis

For the sensitivity analysis, white Gaussian noise was added to the synaptic weights, using the formula

$$w_i = w_{nominal} + \frac{x}{100} w_{nominal} \epsilon, \quad \epsilon \sim \mathcal{N}(\mu, \sigma^2) \tag{5}$$

where w_i is the resulting noisy value of weight i. $i = \{1, 2, \ldots, M\}$ and M is the number of weights. $w_{nominal}$ is the nominal value of the weight, $x \in [0, 100]$ is the percentage of noise to be added to the nominal value, ϵ is a random variable sampled from the Gaussian distribution with $\mu = 0$ and $\sigma^2 = 1$. The number of successful trials was counted in each condition. The criterion for a successful trial was that the activity "bump" tracked the stimulus heading with an error of $<\pm 10^\circ$ for more than 50% of stimulus duration.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found at: https://doi.org/10.1002/cne.24512.

AUTHOR CONTRIBUTIONS

IP conceptualized and developed the method for deriving the effective circuit and contributed to the experimental design, software, validation of results, statistical analysis, visualizations, and manuscript writing.

FUNDING

IP was funded by the Principal's Career Development Scholarship.

ACKNOWLEDGMENTS

The author would like to thank Stanley Heinze and Barbara Webb for their invaluable input as well as William Berg and Nina Kudryashova for their valuable comments on early versions of this manuscript.

REFERENCES

Amari, S. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biol. Cybern.* 27, 77–87. doi: 10.1007/BF00337259

Cichy, R. M., Khosla, A., Pantazis, D., Torralba, A., and Oliva, A. (2016). Comparison of deep neural networks to spatio-temporal cortical dynamics of human visual object recognition reveals hierarchical correspondence. *Sci. Rep.* 6, 1–13. doi: 10.1038/srep27755

Cope, A. J., Sabo, C., Vasilaki, E., Barron, A. B., and Marshall, J. A. (2017). A computational model of the integration of landmarks and motion in the insect central complex. *PLoS ONE* 12:e0172325. doi: 10.1371/journal.pone.0172325

Gaudry, Q., Hong, E. J., Kain, J., de Bivort, B. L., and Wilson, R. I. (2012). Asymmetric neurotransmitter release enables rapid odour lateralization in Drosophila. *Nature* 493, 424–428. doi: 10.1038/nature11747

Giraldo, Y. M., Leitch, K. J., Ros, I. G., Warren, T. L., Weir, P. T., and Dickinson, M. H. (2018). Sun navigation requires compass neurons in Drosophila. *Curr. Biol.* 28, 2845–2852.e4. doi: 10.1016/j.cub.2018.07.002

Gouwens, N. W., and Wilson, R. I. (2009). Signal propagation in Drosophila central neurons. *J. Neurosci.* 29, 6239–6249. doi: 10.1523/JNEUROSCI.0764-09.2009

Green, J., Adachi, A., Shah, K. K., Hirokawa, J. D., Magani, P. S., and Maimon, G. (2017). A neural circuit architecture for angular integration in Drosophila. *Nature* 546, 101–106. doi: 10.1038/nature22343

Green, J., and Maimon, G. (2018). Building a heading signal from anatomically defined neuron types in the Drosophila central complex. *Curr. Opin. Neurobiol.* 52, 156–164. doi: 10.1016/j.conb.2018.06.010

Homberg, U., Heinze, S., Pfeiffer, K., Kinoshita, M., and El Jundi, B. (2011). Central neural coding of sky polarization in insects. *Philos. Trans. R. Soc. B Biol. Sci.* 366, 680–687. doi: 10.1098/rstb.2010.0199

Jonas, E., and Kording, K. P. (2017). Could a neuroscientist understand a microprocessor? *PLoS Comput. Biol.* 13, 1–24. doi: 10.1371/journal.pcbi.1005268

Kakaria, K. S., and de Bivort, B. L. (2017). Ring attractor dynamics emerge from a spiking model of the entire protocerebral bridge. *Front. Behav. Neurosci.* 11:8. doi: 10.3389/fnbeh.2017.00008

Kim, S. S., Rouault, H., Druckmann, S., and Jayaraman, V. (2017). Ring attractor dynamics in the Drosophila central brain. *Science* 356, 849–853. doi: 10.1126/science.aal4835

Liu, G., Seiler, H., Wen, A., Zars, T., Ito, K., Wolf, R., et al. (2006). Distinct memory traces for two visual features in the Drosophila brain. *Nature* 439, 551–556. doi: 10.1038/nature04381

Martin, J. P., Guo, P., Mu, L., Harley, C. M., and Ritzmann, R. E. (2015). Central-complex control of movement in the freely walking cockroach. *Curr. Biol.* 25, 2795–2803. doi: 10.1016/j.cub.2015.09.044

Nagel, K. I., Hong, E. J., and Wilson, R. I. (2015). Synaptic and circuit mechanisms promoting broadband transmission of olfactory stimulus dynamics. *Nat. Neurosci.* 18, 56–65. doi: 10.1038/nn.3895

Neuser, K., Triphan, T., Mronz, M., Poeck, B., and Strauss, R. (2008). Analysis of a spatial orientation memory in Drosophila. *Nature* 453, 1244–1247. doi: 10.1038/nature07003

Ofstad, T. A., Zuker, C. S., and Reiser, M. B. (2011). Visual place learning in Drosophila melanogaster. *Nature* 474, 204–209. doi: 10.1038/nature10131

Pisokas, I., Heinze, S., and Webb, B. (2020). The head direction circuit of two insect species. *eLife*. 9:e53985 doi: 10.7554/eLife.53985.sa2

Rekoff, M. G. (1985). On reverse engineering. *IEEE Trans. Syst. Man Cybern.* 15, 244–252. doi: 10.1109/TSMC.1985.6313354

Ritzmann, R. E., Harley, C. M., Daltorio, K. A., Tietz, B. R., Pollack, A. J., Bender, J. A., et al. (2012). Deciding which way to go: how do insects alter movements to negotiate barriers? *Front. Neurosci.* 6:97. doi: 10.3389/fnins.2012.00097

Rohrbough, J., and Broadie, K. (2002). Electrophysiological analysis of synaptic transmission in central neurons of Drosophila larvae. *J. Neurophysiol.* 88, 847–860. doi: 10.1152/jn.2002.88.2.847

Seelig, J. D., and Jayaraman, V. (2015). Neural dynamics for landmark orientation and angular path integration. *Nature* 521, 186–191. doi: 10.1038/nature14446

Sheeba, V., Gu, H., Sharma, V. K., O'Dowd, D. K., and Holmes, T. C. (2008). Circadian- and light-dependent regulation of resting membrane potential and spontaneous action potential firing of drosophila circadian pacemaker neurons. *J. Neurophysiol.* 99, 976–988. doi: 10.1152/jn.00930.2007

Skaggs, W. E., Knierim, J. J., Kudrimoti, H. S., and McNaughton, B. L. (1995). A model of the neural basis of the rat's sense of direction. *Adv. Neural Inform. Process. Syst.* 7, 173-80.

Stone, T., Webb, B., Adden, A., Weddig, N. B., Honkanen, A., Templin, R., et al. (2017). An anatomically constrained model for path integration in the bee brain. *Curr. Biol.* 27, 3069–3085.e11. doi: 10.1016/j.cub.2017.08.052

Strauss, R. (2002). The central complex and the genetic dissection of locomotor behaviour. *Curr. Opin. Neurobiol.* 12, 633–638. doi: 10.1016/S0959-4388(02)00385-9

Su, T. S., Lee, W. J., Huang, Y. C., Wang, C. T., and Lo, C. C. (2017). Coupled symmetric and asymmetric circuits underlying spatial orientation in fruit flies. *Nat. Commun*. 8. doi: 10.1038/s41467-017-00191-6

Tammero, L. F., and Dickinson, M. H. (2002). The influence of visual landscape on the free flight behavior of the fruit fly Drosophila melanogaster. *J. Exp. Biol.* 205(Pt 3), 327–343.

Taube, J., Muller, R., and Ranck, J. (1990). Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *J. Neurosci*. 10, 420–435. doi: 10.1523/JNEUROSCI.10-02-00420.1990

Triphan, T., Poeck, B., Neuser, K., and Strauss, R. (2010). Visual targeting of motor actions in climbing Drosophila. *Curr. Biol.* 20, 663–668. doi: 10.1016/j.cub.2010.02.055

Turner-Evans, D., Wegener, S., Rouault, H., Franconville, R., Wolff, T., Seelig, J. D., et al. (2017). Angular velocity integration in a fly heading circuit. *eLife* 6, 2112–2126. doi: 10.7554/eLife.23496

Turner-Evans, D. B., Jensen, K., Ali, S., Paterson, T., Sheridan, A., Ray, R. P., Wolff, T., et al. (2020). The neuroanatomical ultrastructure and function of a biological ring attractor. *Neuron 108,* 145–163. doi: 10.1016/j.neuron.2020.08.006

Varga, A. G., Kathman, N. D., Martin, J. P., Guo, P., and Ritzmann, R. E. (2017). Spatial navigation and the central complex: sensory acquisition, orientation, and motor control. *Front. Behav. Neurosci*. 11:4. doi: 10.3389/fnbeh.2017.00004

Wolff, T., Iyer, N. A., and Rubin, G. M. (2015). Neuroarchitecture and neuroanatomy of the Drosophila central complex: a GAL4-based dissection of protocerebral bridge neurons and circuits. *J. Compar. Neurol.* 523, 997–1037. doi: 10.1002/cne.23705

Wolff, T., and Rubin, G. M. (2018). Neuroarchitecture of the Drosophila central complex: a catalog of nodulus and asymmetrical body neurons and a revision of the protocerebral bridge catalog. *J. Compar. Neurol.* 526, 2585–2611. doi: 10.1002/cne.24512

Yamins, D. L., and DiCarlo, J. J. (2016). Using goal-driven deep learning models to understand sensory cortex. *Nat. Neurosci.* 19, 356–365. doi: 10.1038/nn.4244

Yamins, D. L., Hong, H., Cadieu, C. F., Solomon, E. A., Seibert, D., and DiCarlo, J. J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proc. Natl. Acad. Sci. U.S.A.* 111, 8619–8624. doi: 10.1073/pnas.1403112111

Young, J. M., and Armstrong, J. D. (2010). Structure of the adult central complex in Drosophila: organization of distinct neuronal subsets. *J. Compar. Neurol.* 518, 1500–1524. doi: 10.1002/cne.22284

Zhang, K. (1996). Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *J. Neurosci.* 16, 2112–2126. doi: 10.1523/JNEUROSCI.16-06-02112.1996

Conflict of Interest: The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 22 September 2021
doi: 10.3389/fnbot.2021.629210

SpikePropamine: Differentiable Plasticity in Spiking Neural Networks

*Samuel Schmidgall, Julia Ashkanazy, Wallace Lawson and Joe Hays**

U.S. Naval Research Laboratory, Washington, DC, United States

The adaptive changes in synaptic efficacy that occur between spiking neurons have been demonstrated to play a critical role in learning for biological neural networks. Despite this source of inspiration, many learning focused applications using Spiking Neural Networks (SNNs) retain static synaptic connections, preventing additional learning after the initial training period. Here, we introduce a framework for simultaneously learning the underlying fixed-weights and the rules governing the dynamics of synaptic plasticity and neuromodulated synaptic plasticity in SNNs through gradient descent. We further demonstrate the capabilities of this framework on a series of challenging benchmarks, learning the parameters of several plasticity rules including BCM, Oja's, and their respective set of neuromodulatory variants. The experimental results display that SNNs augmented with differentiable plasticity are sufficient for solving a set of challenging temporal learning tasks that a traditional SNN fails to solve, even in the presence of significant noise. These networks are also shown to be capable of producing locomotion on a high-dimensional robotic learning task, where near-minimal degradation in performance is observed in the presence of novel conditions not seen during the initial training period.

Keywords: spiking neural network, plasticity, neuromodulation, reinforcement learning, backpropagation, temporal learning, motor learning, robotic learning

OPEN ACCESS

Edited by:
Ganesh R. Naik,
Western Sydney University, Australia

Reviewed by:
Claudia Casellato,
University of Pavia, Italy
Sumit Bam Shrestha,
Intel, United States
Vikas Bhandawat,
Drexel University, United States
Erik Christopher Johnson,
Johns Hopkins University, United States

****Correspondence:***
Joe Hays
joe.hays@nrl.navy.mil

Received: *13 November 2020*
Accepted: *11 August 2021*
Published: *22 September 2021*

Citation:
Schmidgall S, Ashkanazy J, Lawson W and Hays J (2021) SpikePropamine: Differentiable Plasticity in Spiking Neural Networks. Front. Neurorobot. 15:629210. doi: 10.3389/fnbot.2021.629210

1. INTRODUCTION AND RELATED WORK

The dynamic modification of neuronal properties underlies the basis of learning, memory, and adaptive behavior in biological neural networks. The changes in synaptic efficacy that occur on the connections between neurons play an especially vital role. This process, termed synaptic plasticity, is largely mediated by the interaction of pre- and post-synaptic activity between two synaptically connected neurons in conjunction with local and global modulatory signals. Importantly, synaptic plasticity is largely believed to be one of the primary bases for enabling both stable long-term learning and adaptive short-term responsiveness to novel stimuli (Martin et al., 2000; Zucker and Regehr, 2002; Liu et al., 2012).

An additional mechanism that guides these changes is neuromodulation. Neuromodulation, as the name suggests, is the process by which select neurons modulate the activity of other neurons; this is accomplished by the use of chemical messaging signals. Such messages are mediated by the release of chemicals from neurons themselves, often using one or more stereotyped signals to regulate diverse populations of neurons. Dopamine, a neuromodulator commonly attributed to learning (Schultz et al., 1997; Frank et al., 2004; Hosp et al., 2011), has been experimentally shown to portray a striking resemblance to the Temporal-Difference (TD) reward prediction error (Montague et al., 1996; Schultz et al., 1997; Niv et al., 2005) and more recently distributional

coding methods of reward prediction (Dabney et al., 2020). Such signals have been shown to play a critical role in guiding the effective changes in synaptic plasticity, allowing the brain to regulate the location and scale with which such changes are made (Gerstner et al., 2018). The conceptual role of dopamine has largely shaped the development of modern reinforcement learning (RL) algorithms, enabling the impressive accomplishments seen in recent literature (Mnih et al., 2013; Bellemare et al., 2017; Barth-Maron et al., 2018; Haarnoja et al., 2018). While dopamine has primarily taken the spotlight in RL, many other important neuromodulatory signals have largely been excluded from learning algorithms in artificial intelligence (AI). For example, acetylcholine has been shown to play a vital role in motor control, with neuromodulatory signals often sent as far as from the brainstem to motor neurons (Zaninetti et al., 1999). Modeling how these neuromodulatory processes develop, as well as how neurons can directly control neuromodulatory signals are likely critical steps toward successfully reproducing the impressive behaviors exhibited by the brain.

Both historically and recently, neuroscience and AI have had a fruitful relationship, with neuroscientific speculations being validated through AI, and advancements in the capabilities of AI being a result of a better understanding of the brain. A major contributor toward enabling this, particularly in AI, is through the application of backpropagation for learning the weights of Artificial Neural Networks (ANNs). Although backpropagation is largely believed to be biologically implausible (Bengio et al., 2015), networks trained under certain conditions using this algorithm have been shown to display behavior remarkably similar to biological neural networks (Banino et al., 2018; Cueva and Wei, 2018).

The promising advances toward more brain-like computations have led to the development of SNNs. These networks more closely resemble the dynamics of biological neural networks by storing and integrating membrane potential to produce binary spikes. Consequently, such networks are naturally suited toward solving temporally extended tasks, as well as producing many of the desirable benefits seen in biological networks such as energy efficiency, noise robustness, and rapid inference (Pfeiffer and Pfeil, 2018). However, until recently, the successes of SNNs have been overshadowed by the accomplishments of ANNs. This is primarily due to the use of spikes for information transmission, which does not naturally lend itself toward being used with backpropagation. To circumvent this challenge, a wide variety of learning algorithms have been proposed including Spike-Timing Dependent Plasticity (STDP) (Masquelier et al., 2009; Bengio et al., 2017; Kheradpisheh et al., 2018; Mozafari et al., 2018), ANN to SNN conversion methods (Diehl et al., 2015; Rueckauer et al., 2017; Hu et al., 2018), Eligibility Traces (Bellec et al., 2020), and Evolutionary Strategies (Pavlidis et al., 2005; Carlson et al., 2014; Eskandari et al., 2016; Schmidgall, 2020). However, a separate body of literature enables the use of backpropagation directly with SNNs typically through the use of surrogate gradients (Bohte et al., 2002; Sporea and Grüning, 2012; Lee et al., 2016; Shrestha and Orchard, 2018). These surrogate gradient methods are primary contributors for many of the state-of-the-art results obtained using SNNs from supervised learning to RL. Counter to biology, temporal learning tasks such as RL interact with an external environment over multiple episodes before synaptic weight updates are computed. Between these update intervals, the synaptic weights remain unchanged, diminishing the potential for online learning to occur. Recent work by Miconi et al. (2018) transcends this dominant fixed-weight approach specifically for the recurrent weights of ANNs by presenting a framework for augmenting traditional fixed-weight networks with Hebbian plasticity, where backpropagation updates both the weights and parameters guiding plasticity. In follow-up work, this hybrid framework was expanded to include neuromodulatory signals, whose parameters were also learned using backpropagation (Miconi et al., 2019).

Learning-to-learn, or meta-learning, is the capability to learn or improve one's own learning ability. The brain is constantly modifying and improving its own ability to learn at both the local and global scale. This was originally theorized to be a product of neurotransmitter distribution from the Basal Ganglia (Doya, 2002), but has also included contributions from the Prefrontal Cortex (Wang et al., 2018) and the Cerebellum (Doya, 1999) to name a few. In machine learning, meta-learning approaches aim to improve the learning algorithm itself rather than retaining a static learning process (Hospedales et al., 2020). For spiking neuro-controllers, learning-to-learn through the discovery of synaptic plasticity rules offline provides a mechanism for learning on-chip since neuromorphic hardware is otherwise incompatible with on-chip backpropagation. Many neuromorphic chips provide a natural mechanism for incorporating synaptic plasticity (Davies et al., 2018; van Albada et al., 2018), and more recently, neuromodulatory signals (Mikaitis et al., 2018).

Despite the prevalence of plasticity in biologically-inspired learning methods, a method for learning both the underlying weights and plasticity parameters using gradient descent has yet to be proposed for SNNs. Building off of Miconi et al. (2019), which was focused on ANNs, this paper provides a framework for incorporating plasticity and neuromodulation with SNNs trained using gradient descent. In addition, five unique plasticity rules inspired by the neuroscientific literature are introduced. A series of experiments are conducted with using a complex cue-association environment, as well as a high-dimensional robotic locomotion task. From the experimental results, networks endowed with plasticity on only the forward propagating weights, with no recurrent self-connections, are shown to be sufficient for solving challenging temporal learning tasks that a traditional SNN fails to solve, even while experiencing significant noise perturbations. Additionally, these networks are much more capable of adapting to conditions not seen during training, and in some cases displaying near-minimal degradation in performance.

2. DIFFERENTIABLE PLASTICITY

Section 2.1 begins by describing the dynamics of an SNN. Using these dynamic equations, section 2.2 then introduces the generalized framework for differentiable plasticity of an SNN

as well as some explicit forms of differentiable plasticity rules. First, the Differentiable Plasticity (DP) form of Linear Decay is introduced, primarily due to the conceptual simplicity of its formulation. Next, the DP form of Oja's rule (Oja, 1983) is presented as DP-Oja's. This rule is introduced primarily because, unlike Linear Decay, it provides a natural and simple mechanism for stable learning, namely a penalty on weight-growth. The next form is based on the Bienenstock, Cooper, and Munro (BCM) rule (Bienenstock et al., 1982), named DP-BCM. Like Oja's rule, the BCM rule provides stability, except in this case the penalty accounts for a given neuron's deviation from the average spike-firing rate. Finally, a respective set of neuromodulatory variants for the Oja's and BCM differentiable plasticity rules are presented in section 2.3, as well as a generalized framework for differentiable neuromodulation. The rules described in this section serve primarily to demonstrate an explicit implementation of the generalized framework on two well-studied synaptic learning rules.

2.1. Spiking Neural Network

We will begin by describing the dynamics of an SNN, and then proceed in the following sections to describe a set of plasticity rules that can be applied to such networks. We begin with the following set of equations:

$$\boldsymbol{a}^{(l)}(t) = \varepsilon * \boldsymbol{s}^{(l-1)}(t) \tag{1}$$

$$\boldsymbol{u}^{(l)}(t) = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l)}(t) + \nu * \boldsymbol{s}^{(l)}(t) \tag{2}$$

$$\boldsymbol{s}^{(l)}(t) = f_s(\boldsymbol{u}^{(l)}(t)). \tag{3}$$

The superscript $l \in \mathbb{N}$ represents the index for a layer of neurons and $t \in \mathbb{N}$ discrete time. We further define $n^{(l)} \in \mathbb{N}$ to represent the number of neurons in layer l. From here, $\varepsilon(\cdot)$ is a spike response kernel which is used to generate a spike response signal, $\boldsymbol{a}^{(l)}(t) \in \mathbb{R}^{n^{(l-1)}}$, by convolving incoming spikes $\boldsymbol{s}^{(l-1)}(t) \in \mathbb{B}^{n^{(l-1)}}, \mathbb{B} = \{0, 1\}$ over $\varepsilon(\cdot)$. We further define the binary vector $\boldsymbol{s}^{(0)}(t)$ to represent sensory input obtained from the environment. Often in practice, the effect of $\varepsilon(\cdot)$ provides an exponentially decaying contribution over time, which consequently has minimal influence after a fixed number of steps. Exploiting this concept, $\varepsilon(\cdot)$ may be represented as a finite weighted decay kernel with dimensionality K, which is chosen heuristically as the point in time with which $\varepsilon(\cdot)$ has minimal practical contribution. $\nu(\cdot)$ is chosen in a similar manner, where $\nu(\cdot)$ is the refractory kernel, convolving together with spikes $\boldsymbol{s}^{(l)}(t)$ to produce the refractory response $\nu * \boldsymbol{s}^{(l)}(t) \in \mathbb{R}^{n^{(l)}}$. Additionally, $\mathbf{W}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$ numerically represents the synaptic strength between each neuron connected from layer $l-1$ and l, which, as a weight is multiplied by $\boldsymbol{a}^{(l)}(t)$ and further summed with the refractory response $\nu * \boldsymbol{s}^{(l)}(t)$ to produce the membrane potential $\boldsymbol{u}^{(l)}(t) \in \mathbb{R}^{n^{(l)}}$. The membrane potential stores the weighted sum of spiking activity arriving from incoming synaptic connections, referred to as the *Post Synaptic Potential (PSP)*.

We further define the spike function $f_s(\cdot)$ as:

$$f_s(u) : u \to s := s(t) + \delta(t - t^{(f)}) \tag{4}$$

$$t^{(f)} = \min\{t : u(t) = \vartheta, t > t^{(f-1)}\} \tag{5}$$

In these equations, the function $f_s(\cdot)$ produces a binary spike based on the neuron's internal membrane potential, $\boldsymbol{u}_i(t)$, $i \in \mathbb{N}$ indexing an individual neuron. When $\boldsymbol{u}_i(t)$ passes a threshold $\vartheta \in \mathbb{R}$, the respective binary spike is propagated downstream to a set of connected neurons, and the internal membrane potential for that neuron is reset to a baseline value $u_r \in \mathbb{R}$, which is often set to zero. The function enabling this is referred to as a dirac-delta, $\delta(t)$, which produces a binary output of one when $t = 0$ and zero otherwise. Here, $t^f \in \mathbb{R}$ denotes the firing time of the f^{th} spike, so that when $t = t^{(f)}$ then $\delta(t - t^{(f)}) = 1$.

Like an artificial neural network, $f_s(\cdot)$ can be viewed as having similar functionality to an arbitrary non-linear activation function $\phi(\cdot)$. Unlike the ANN however, $f_s(\cdot)$ has an undefined derivative making the gradient computation for backpropagation particularly challenging. To enable backpropagation through the non-differentiable aspects of the network, the Spike Layer Error Reassignment in Time (SLAYER) algorithm is used (Shrestha and Orchard, 2018). SLAYER overcomes such difficulties by representing the derivative of a spike as a surrogate gradient and uses a temporal credit assignment policy for backpropagating error to previous layers. Although SLAYER was used in this paper, we note that any spike-derivative approximation method will work together with our methods.

2.2. Spike-Based Differentiable Plasticity

To enable differentiable plasticity we utilize the SNN dynamic equations described in (1–5), however now both the weights and the rules governing plasticity are optimized by gradient descent. This is enabled through the addition of a synaptic trace variable, $\boldsymbol{E}^{(l)}(t) \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$, which accumulates traces of the local synaptic activities between pre- and post-synaptic connections. An additional plasticity coefficient, $\boldsymbol{\alpha}^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$, is often learned which serves to element-wise scale the magnitude and direction of the synaptic traces independently from the trace dynamics. By augmenting our SNN we obtain:

$$\boldsymbol{a}^{(l)}(t) = (\varepsilon * \boldsymbol{s}^{(l-1)}(t)) \tag{6}$$

$$\boldsymbol{u}^{(l)}(t) = (\boldsymbol{W}^{(l)} + \boldsymbol{\alpha}^{(l)} \odot \boldsymbol{E}^{(l)}(t))\boldsymbol{a}^{(l)}(t) + (\nu * \boldsymbol{s}^{(l)})(t) \tag{7}$$

$$\boldsymbol{s}^{(l)}(t) = f_s(\boldsymbol{u}^{(l)}(t)). \tag{8}$$

The Hadamard product, $\odot$, is used to represent element-wise multiplication. The primary modifications from the fixed-weight SNN framework in (1–3) are in the addition of the synaptic trace $\boldsymbol{E}^{(l)}(t)$ and plasticity coefficient $\boldsymbol{\alpha}^{(l)}$ in (7). Without this modification, the underlying weight $\boldsymbol{W}^{(l)}$ remains constant from episode-to-episode in the same way as (2). However, the additional contribution of the synaptic trace $\boldsymbol{E}^{(l)}(t)$ enables each weight value to be modified through the interaction of *local* or

global activity. The differentiable plasticity and neuromodulated plasticity frameworks presented in this work are concerned with learning such local and global signals, respectively. Proceeding, we present three different methods of synaptic plasticity followed by a section describing neuromodulated plasticity. We additionally note that this framework is not limited to these particular plasticity rules and can be expanded upon to account for a wide variety of different methods.

2.2.1. Generalized

Neuronal activity can be represented by a diverse family of forms. Plasticity rules have been proposed using varying levels of abstraction, from spike rates and spike timing, all the way to modeling calcium-dependent interactions. To encapsulate this wide variety in our work, we abstractly define a vector $\boldsymbol{\rho}^{(l)}(t)$ to represent activity for a layer of neurons l at time t. In many practical instances, time t may represent continuous time, however our examples and discussions are primarily concerned with the evolution of discrete time, which is the default mode from which many models of spiking neurons operate. Likewise, the activity vector $\boldsymbol{\rho}^{(l)}(t)$ may be represented by a variety of different sets, such as $\mathbb{B}^{n^{(l)}}$ or $\mathbb{R}^{n^{(l)}}$ for spike-timing or rate-based activity; this is primarily dependent on which types of activity the experimenter desires to model. The generalized equation for differentiable plasticity is expressed as follows:

$$\boldsymbol{E}^{(l)}(t+\Delta\tau) = F(\boldsymbol{\rho}^{(l-1)}(t), \boldsymbol{\rho}^{(l)}(t), \boldsymbol{E}^{(l)}(t), L^{(l)}). \tag{9}$$

Here, $\boldsymbol{E}^{(l)}(t+\Delta\tau)$ is updated after a specified time interval $\Delta\tau \in \mathbb{N}$. In these equations, $F(\cdot)$ is a function of the pre- and post-synaptic activity, $\boldsymbol{\rho}^{(l-1)}(t)$ and $\boldsymbol{\rho}^{(l)}(t)$, as well as $\boldsymbol{E}^{(l)}(t)$ at the current time-step and $L^{(l)}$ which represents an arbitrary set of functions describing *local* neuronal activity from either pre- or post-synaptic neurons. In practice, $\boldsymbol{E}^{(l)}(t=0)$ is often set to zero at the beginning of a new temporal interaction.

2.2.2. DP-Linear Decay

Perhaps the simplest form of differentiable plasticity is the linear decay method:

$$\boldsymbol{E}^{(l)}(t+\Delta\tau) = (1-\eta^{(l)})\boldsymbol{E}^{(l)}(t) + \eta^{(l)}(\boldsymbol{\rho}^{(l)}(t))^{\intercal}\boldsymbol{\rho}^{(l-1)}(t). \tag{10}$$

Let the set $L^{(l)} = \{\eta^{(l)}\}$, with $\eta^{(l)} \in \mathbb{R}$. In this equation, $\boldsymbol{E}^{(l)}(t+\Delta\tau)$ is computed using the local layer-specific function $\eta^{(l)}$, representing the rate at which new local activity $\boldsymbol{\rho}^{(l)}(t)(\boldsymbol{\rho}^{(l-1)}(t))^{\intercal} \in \mathbb{R}^{n^{(l)}\times n^{(l-1)}}$ is incorporated into the synaptic trace, as well as the degree to which prior synaptic activity will be 'remembered' from $(1-\eta^{(l)})\boldsymbol{E}^{(l)}(t)$. While the parameters regulating $\boldsymbol{E}^{(l)}(t+\Delta\tau)$ will generally approach values that produce stable weight growth, in practice $\boldsymbol{E}^{(l)}(t)$ is often clipped to enforce stable bounds. Here, the local variable $\eta^{(l)}$ acts as a free parameter and is learned through gradient descent.

2.2.3. DP-Oja's

Among the most studied synaptic learning rules, Oja's rule simplistically provides a natural system of stability and effective correlation (Oja, 1982). This rule balances potentiation and depression directly from the synaptic activity stored in the trace, which cause a decay proportional to its magnitude. Mathematically, Oja's rule enables the neuron to perform Principal Component Analysis (PCA) which is a common method for finding unsupervised statistical trends in data (Oja, 1983). Building off of this work, we incorporate Oja's rule into our framework as Differentiable Plasticity Oja's rule (DP-Oja's). Rather than a generalized representation of activity, DP-Oja's rule uses a more specific rate-based representation, where $\boldsymbol{\rho}^{(l)}(t) = \boldsymbol{r}^{(l)}(t) \in \mathbb{R}^{n^{(l)}}$. To obtain $\boldsymbol{r}^{(l)}(t)$, spike averages are computed over the pre-defined interval $\Delta\tau$. DP-Oja's rule is defined as:

$$\boldsymbol{E}^{(l)}(t+\Delta\tau) = (1-\eta^{(l)})\boldsymbol{E}^{(l)}(t) + \eta^{(l)}(\boldsymbol{r}^{(l)} - \boldsymbol{E}^{(l)}\boldsymbol{r}^{(l-1)})(t)^{\intercal}\boldsymbol{r}^{(l-1)}(t). \tag{11}$$

Similar to (10), we let the set $L^{(l)} = \{\eta^{(l)}\}$ contain the local layer-specific value $\eta^{(l)}$ that governs the incorporation of novel synaptic activity. Differing however, $\boldsymbol{E}^{(l)}(t)$ is used twice. The term $\boldsymbol{E}^{(l)}(t)$ being multiplied by $(1-\eta^{(l)})$ on the left-side of (11) serves a similar purpose compared with (10), however on the right-hand side of the addition this value penalizes unbounded growth, acting as an unsupervised regulatory mechanism. Here, like in (10), $\eta^{(l)}$ acts as a free parameter learned through gradient descent.

2.2.4. DP-BCM

Another well-studied example of plasticity is the BCM rule (Bienenstock et al., 1982). The BCM rule has been shown to exhibit similar behavior to STDP under certain conditions (Izhikevich and Desai, 2003), as well as to successfully describe the development of receptive fields (Shouval et al., 1970; Law and Cooper, 1994). BCM differs from Oja's rule in that it has more direct control over potentiation and depression through the use of a dynamic threshold which often represents the average spike rate of each neuron. In this example of differentiable plasticity, we describe a model of BCM, where the dynamics governing the plasticity as well as the stability-providing sliding threshold are learned through backpropagation, which we refer to as Differentiable Plasticity BCM (DP-BCM). This rule can be described as follows:

$$\boldsymbol{E}^{(l)}(t+\Delta\tau) = (1-\eta^{(l)})\boldsymbol{E}^{(l)}(t) + \eta^{(l)}(\boldsymbol{r}^{(l)}(t))^{\intercal}\boldsymbol{r}_{\beta}^{(l)}(t) \tag{12}$$

$$\boldsymbol{r}_{\beta}^{(l)}(t) = \boldsymbol{r}^{(l-1)}(t) \odot (\boldsymbol{r}^{(l-1)}(t) - (\boldsymbol{\phi}^{(l)}(t) + \boldsymbol{\psi}^{(l)})) \tag{13}$$

$$\boldsymbol{\phi}^{(l)}(t+\Delta\tau) = (1-\eta_{\phi}^{(l)})\boldsymbol{\phi}^{(l)}(t) + \eta_{\phi}^{(l)}\omega(\boldsymbol{r}^{(l-1)}(t)). \tag{14}$$

As in (11), the DP-BCM uses a rate-based representation of synaptic activity, where $\boldsymbol{\rho}^{(l)}(t) = \boldsymbol{r}^{(l)}(t)$. Here, we let the set of local functions $L^{(l)} = \{\boldsymbol{\psi}^{(l)}, \boldsymbol{\phi}^{(l)}, \eta_{\phi}^{(l)}, \eta^{(l)}\}$. To begin, $\boldsymbol{\psi}^{(l)} \in \mathbb{R}^{n^{(l-1)}}$ is a bias vector that remains static during interaction time, and the parameter $\boldsymbol{\phi}^{(l)}(t) \in \mathbb{R}^{n^{(l-1)}}$ is its dynamic counterpart. These parameters enable the addition of a sliding-boundary, $\boldsymbol{\phi}^{(l)}(t) + \boldsymbol{\psi}^{(l)}$, which determines whether activity results in potentiation vs. depression. The dynamics of this boundary are described in (14). Otherwise, $\omega(\cdot)$ serves as an arbitrary function

of the pre-synaptic activity $\boldsymbol{r}^{(l-1)}(t)$, and $\eta_{\phi}^{(l)} \in \mathbb{R}$ determines the rate at which new information is incorporated into the $\boldsymbol{\phi}^{(l)}(t)$ trace. For our experiments we let $\omega(\cdot) = I(\cdot)$, which is the identity function. This altogether has the effect of slowly incorporating the observed rate of pre-synaptic activity $\boldsymbol{r}^{(l-1)}(t)$ into $\boldsymbol{\phi}^{(l)}(t)$. Finally, $\eta^{(l)} \in \mathbb{R}$ provides the same utility as in (10). Comparatively, the BCM rule is more naturally suited for regulating potentiation and depression than Oja's, which primarily regulates depression through synaptic weight decay. Among the local functions, $\boldsymbol{\psi}^{(l)}$, $\eta_{\phi}^{(l)}$, and $\eta^{(l)}$ are free parameters learned through gradient descent.

2.3. Spike-Based Differentiable Neuromodulation

In addition to differentiable plasticity, a framework for differentiable neuromodulation is also presented. Neuromodulation, or neuromodulated plasticity, allows the use of both learned local signals, as well as learned *global* signals. These signals modulate the effects of plasticity by scaling the magnitude and direction of synaptic modifications based on situational neuronal activity. This adds an additional layer of learned supervision, enabling the potential for learning-to-learn, or *meta-learning*. Adding neuromodulation provides an analogy to the neuromodulatory signals observed in biological neural networks, which provide a rich set of biological processes to take inspiration from.

As before, we present the equation for generalized neuromodulated plasticity and further describe a series of more specific neuromodulation rules.

2.3.1. Generalized Neuromodulated Plasticity

The generalized equation for neuromodulated plasticity is as follows:

$$\boldsymbol{E}^{(l)}(t+\Delta\tau) = G(\boldsymbol{\rho}^{(l-1)}(t), \boldsymbol{\rho}^{(l)}(t), \boldsymbol{E}^{(l)}(t), L^{(l)}, M). \tag{15}$$

In this equation, $G(\cdot)$ has the same functionality as $F(\cdot)$ in (9) except for the addition of neuromodulatory signals M. Here, the values contained in M may be represented by a wide variety of functions, however it differs from $L^{(l)}$ in that the elements may express global signals. Global signals may be computed at any part of the network, or by a separate network all together. Additionally, global signals may be incorporated that are learned independent of the modulatory reaction, such as dopamine-inspired TD-error from an independent value-prediction network, or a function computing predictive feedback-error signals as is observed in the cerebellum (Popa and Ebner, 2019).

2.3.2. NDP-Oja's

Building off of (11), Oja's synaptic update rule is augmented with a neuromodulatory signal that linearly weights the neuronal activity of post-synaptic neurons. This neuromodulated variant of Oja's rule (NDP-Oja's) is described as follows:

$$\begin{aligned}\boldsymbol{E}^{(l)}(t+\Delta\tau) = (1-\eta^{(l)})\boldsymbol{E}^{(l)}(t) + \eta^{(l)}(\boldsymbol{M}^{(l)}(t) \odot \boldsymbol{r}^{(l)} \\ - \boldsymbol{E}^{(l)}\boldsymbol{r}^{(l-1)})(t))^{\intercal}\boldsymbol{r}^{(l-1)}(t)\end{aligned} \tag{16}$$

$$\boldsymbol{M}^{(l)}(t) = \boldsymbol{W}_m^{(l)}\boldsymbol{r}^{(l)}(t). \tag{17}$$

Where the parameter $\boldsymbol{W}_m^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l)}}$ weights the post-synaptic activity $\boldsymbol{r}^{(l)}(t)$, which modulates the right-hand trace dynamics in (16). Importantly, the effect of the gradient in learning $\boldsymbol{M}^{(l)}(t) \in \mathbb{R}^{n^{(l)}}$ also contributes toward modifying the parameters producing the post-synaptic activity $\boldsymbol{r}^{(l)}(t)$ rather than simply having a passive relationship. This enables more deliberate and effective control of the neuromodulatory signal. In this equation, both $\boldsymbol{W}_m^{(l)}$ and $\eta^{(l)}$ are free parameters learned through gradient descent. Otherwise, the role of each parameter is identical to (11).

2.3.3. NDP-BCM

In a similar manner, Equations (12–14) for DP-BCM are augmented with a neuromodulatory signal that linearly weights the neuronal activity of post-synaptic neurons, which is referred to as Neuromodulated Differentiable Plasticity BCM (NDP-BCM). This rule can be described as follows:

$$\boldsymbol{E}^{(l)}(t+\Delta\tau) = (1-\eta^{(l)})\boldsymbol{E}^{(l)}(t) + \eta^{(l)}(\boldsymbol{M}^{(l)}(t) \odot \boldsymbol{r}^{(l)}(t))^{\intercal}\boldsymbol{r}_{\beta}^{(l)}(t) \tag{18}$$

$$\boldsymbol{r}_{\beta}^{(l)}(t) = \boldsymbol{r}^{(l-1)}(t) \odot (\boldsymbol{r}^{(l-1)}(t) - (\boldsymbol{\phi}^{(l)}(t) + \boldsymbol{\psi}^{(l)})) \tag{19}$$

$$\boldsymbol{\phi}^{(l)}(t+\Delta\tau) = (1-\eta_{\phi}^{(l)})\boldsymbol{\phi}^{(l)}(t) + \eta_{\phi}^{(l)}\omega(\boldsymbol{r}^{(l-1)}(t)). \tag{20}$$

$$\boldsymbol{M}^{(l)}(t) = \boldsymbol{W}_m^{(l)}\boldsymbol{r}^{(l)}(t). \tag{21}$$

As above, the learned parameter $\boldsymbol{W}_m^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l)}}$ weights the post-synaptic activity $\boldsymbol{r}^{(l)}(t)$, which is then distributed to modulate the right-hand trace dynamics in (18). Otherwise, each parameter follows from (12–14). Similarly, $\boldsymbol{\psi}^{(l)}$, $\eta_{\phi}^{(l)}$, and $\eta^{(l)}$ as well as $\boldsymbol{W}_m^{(l)}$ are free parameters learned through gradient descent.

3. RESULTS

The results of this work demonstrate the improvements in performance that differentiable plasticity provides over fixed-weight SNNs, as well as the unique behavioral patterns that emerge as a result of differentiable plasticity. Presented here are two distinct environments which require challenging credit assignment.

3.1. Noisy Cue-Association: Temporal Credit-Assignment Task

Experience-dependent synaptic changes provide critical functionality for both short- and long-term memory. Importantly, such a mechanism should be able to disentangle the correlations between complex sensory cues with *delayed* rewards, where the learning agent often has to wait a variable amount or time before an action is made and a reward is received. A common learning experiment in neuroscience analyzes the performance of rodents in a similar context through the use of a T-maze training environment (Kuśmierz et al., 2017; Engelhard et al., 2019). Here, a rodent moves down a straight corridor

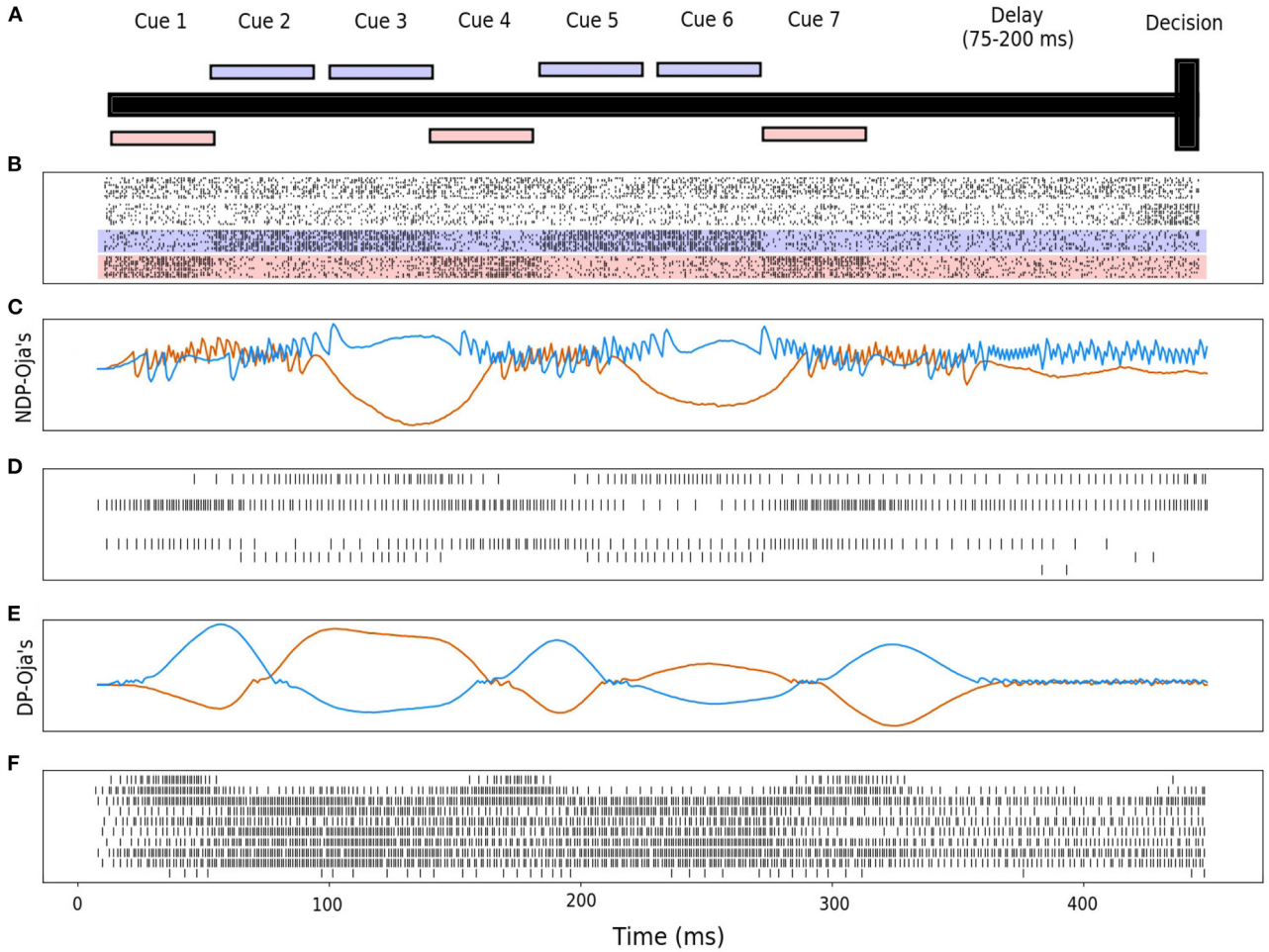


FIGURE 1 | (A) A graphical visualization of the cue-association T-Maze, where the cues are sequentially presented followed by a delay and decision period. The right cue is shown in red, the left in blue. **(B)** Displayed is the sensory neuron input activity. From bottom to top, the activity of ten neurons each represent left and right cues, followed by ten neurons for an action decision cue, and finally ten neurons that have activity with no relationship to the task. Membrane potential for the output neurons, and the spiking activity of a random subset of 10 hidden neurons for neuromodulated Oja's rule **(C,D)** and non-modulated Oja's rule **(E,F)**. The blue and red curves here correspond to the neuron representing the decision for choosing left or right, respectively, also corresponding to the left and right cue colors in **(A,B)**. In **(D)**, only 5 of the 10 neurons were actively spiking, whereas in **(F)** all 10 were.

where a series of sensory visual cues are arranged randomly on the left and right of the rodent as it walks toward the end of the maze. After the sensory cues are displayed, there is a delay interval between the cues and the decision period. Finally, at the T-junction, the rodent is faced with the decision of turning either left of right. A positive reward is given if the rodent chooses the side with the highest number of visual cues. This environment poses unique challenges representative of a natural temporal learning problem, as the decision-making agent is required to learn that reward is independent of both the temporal order of each cue as well as the side of the final cue.

Rather than visual cues specifically, the cues in our experiment produce a time period of high-spiking activity that is input into a distinct set of neurons for each cue (**Figure 1**). Additionally, a similarly-sized set of neurons begins producing spikes near the T-junction indicating a decision period, from which the agent is expected to produce a decision to go left or right. Finally, the last set of neurons produce noise to make the task more challenging. The cue-association task has been shown to be solvable in simulation with the use of recurrent spiking neural networks for both Backpropagation Through Time (BPTT) and eligibility propagation (E-Prop) algorithms (Bellec et al., 2020). However, using the same training methods, a feedforward spiking neural network without recurrent connections is not able to solve this task. In the Miconi et al. (2019) experiments, results were shown for ANNs with plastic and neuromodulated synapses on only the recurrent weights. In this experiment, we determine whether non-recurrent feedforward networks with plastic synapses are sufficient for solving this same task. Here, we consider both DP-BCM and DP-Oja's rules as well as the respective neuromodulatory variants described in sections 2.2, 2.3. We also collect results from an additional environment where

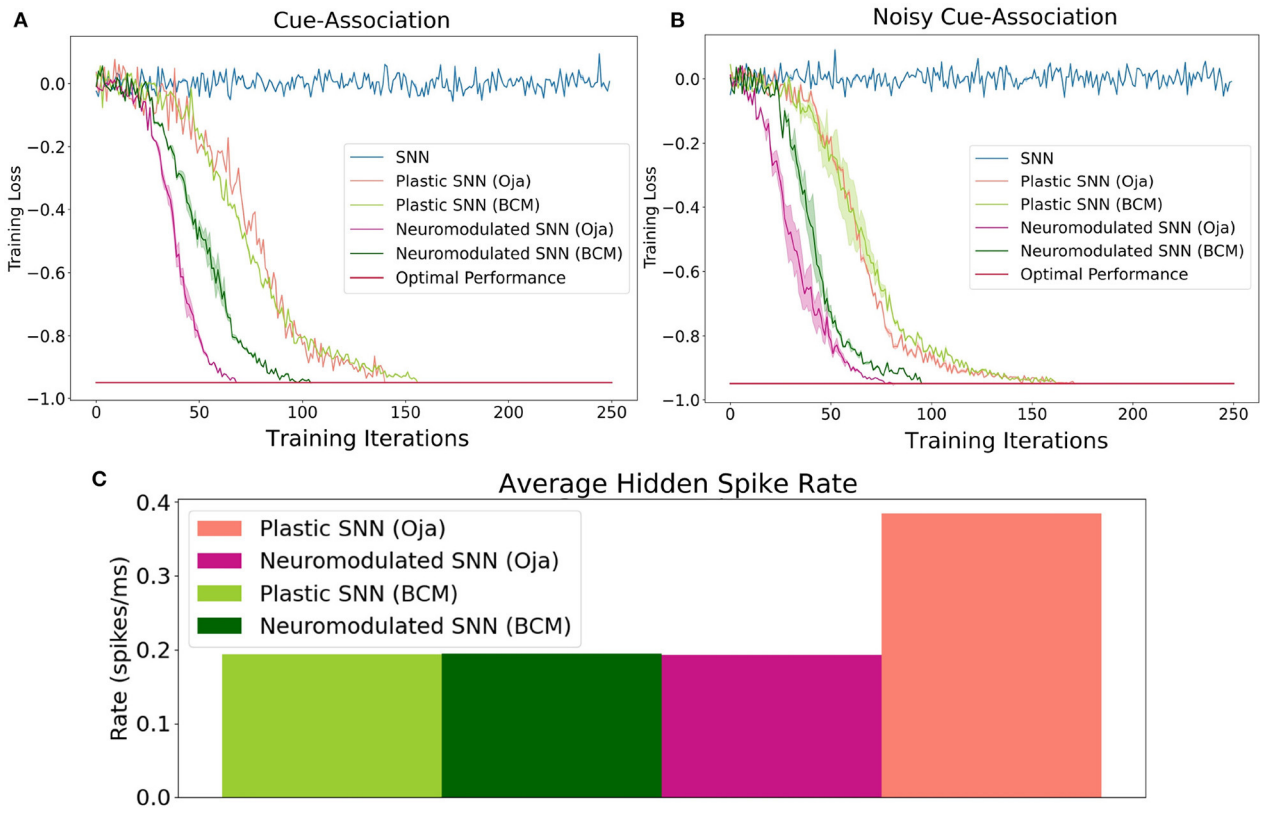


FIGURE 2 | The performance of Oja's and BCM plastic and neuromodulated learning rules on the low-noise **(A)** and high-noise **(B)** environments. **(C)** The average spike firing rate of hidden layer neurons in the high-noise environments for plastic and neuromodulated Oja's and BCM. Once the average network training loss is less than −0.97, which here is consider optimal, the training is halted. Training accuracy is defined as the ratio of correct cues averaged over a series of trials. To perform gradient descent, training loss is this value multiplied by −1.

each population of neurons has a significantly higher probability of spiking randomly, as well as a reduced probability of spiking during the actual cue interval (**Figure 2**).

The input layer is comprised of 40 neurons: 10 for the right-sided cues, 10 for the left-sided cues, 10 neurons which display activity during the decision period, and 10 neurons which produce spike noise (**Figure 1B**). The hidden layer is comprised of 64 neurons, where each neuron is synaptically connected to every neuron in the input layer. Finally, the output layer is similarly fully-connected with two output neurons (**Figure 3**).

Output activity is collected over the decision interval averaging the number of spikes over each distinct output neuron. To decide which action is taken at the end of the decision interval, the output activity is used as the 2-dimensional log-odds input for a binomial distribution from which an action is then sampled. To compute the parameter gradients, the policy gradient algorithm is employed together with BPTT and the Adam optimization method (Kingma and Ba, 2014). To compute the policy gradient, a reward of one is given for successfully solving the task, where otherwise a reward of negative one is given. A more in-depth description of the training details is reserved for the section 5.4 of the **Appendix**.

Accuracy on this task is defined as the ratio of correct cue-decisions at the end of the maze averaged over 100 trials. Training loss is defined as accuracy multiplied by −1, hence the optimal performance is −1. The training results for both the high- and low-noise environments are shown in **Figure 2**. For both environments, the NDP-BCM and NDP-Oja's consistently outperform both DP-BCM and DP-Oja's, whereas the non-plastic SNN fails to solve the task. The neuromodulated networks learn to solve the task most efficiently, despite having more complex dynamics as well as a larger set of parameters to learn. In comparing the non-modulated plasticity variants, there is minimal difference in learning efficiency for either environment. Interestingly, there was minimal degradation in training performance when transitioning from a low- to high-noise environment as shown in **Figures 2A,B**.

One observed difference between the activity of the neuromodulated and non-modulated variants of BCM and Oja's rule is their average hidden spike rates. The average spike-firing rate is relatively consistent between NDP-Oja's and NDP-BCM, as well as the DP-BCM, however the DP-Oja's network has almost twice the spike-firing rate of the former three networks (**Figure 2**). This is thought to be due to Oja's rule primarily controlling synaptic depression through weight decay, which tends to produce larger weight values in active networks. This differs from the BCM rule, which produces a sliding boundary based on the average neuronal spike-firing rate to control

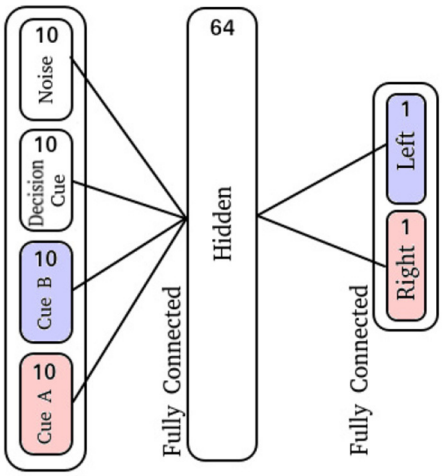


FIGURE 3 | Cue-Association network diagram.

potentiation and depression. The neuromodulated variant of Oja's rule however, can control potentiation and depression through the learned modulatory signal, bypassing the weight-value associated decay. This effect is also seen in **Figures 1D,F** where the spiking activity of 10 randomly-sampled neurons is shown. Again, the NDP-Oja's rule shows drastically lower average spiking activity. This demonstrates that neuromodulatory signals may provide critical information in synaptic learning rules where depression is not actively controlled. Reduced activity is desirable in neuromorphic hardware, as it enables lower energy consumption in practical applications.

To better understand the role of neuromodulation in this experiment, four unqiue cue-association cases are considered (Figures A1–A4 in **Appendix**). It is observed that the modulatory signals on the output neurons exhibit loose symmetry, whereas the activity on hidden neurons follow a similar dynamic pattern for different cues. Additionally, the plastic-weights are shown to behave differently when deprived of sensory-cues. During this deprivation period, the characteristic potentiation and depression seen in all other cue patterns is absent. It is evident from these experiments that the plasticity and neuromodulation have a significant effect on self-organization and behavior. A more in-depth analysis of the neuronal activity is provided in the **Appendix** (section 5.6).

3.2. High-Dimensional Robotic Locomotion Task

Locomotion is among the most impressive capabilities of the brain. The utility of such a capability for a learning agent extends beyond biological organisms, and has received a long history of attention in the robotics field for its many practical applications. Importantly, among the most desirable properties of a locomotive robot are *adaptiveness*, *robustness*, as well as energy efficiency. However, it is worth noting that the importance of having adaptive capabilities for a locomotive agent primarily serves to enable robust performance in response to noise, varying environments, and novel situations. Since plasticity in networks largely serves as a mechanism toward adapting appropriately to new stimuli, we test the adaptive capabilities of our differentiable plasticity networks in a locomotive robotic learning setting (**Figure 4**).

We thus begin by considering a modified version of the Half-Cheetah environment. Half-Cheetah is a common benchmark used to examine the efficacy of RL algorithms. This environment begins with a robot which loosely has the form of a cheetah, controlling six actuated joints equally divided among the two limbs. Additionally, the robot is restricted to motion in 2-dimensions, hence the name 'Half-Cheetah'. In total, the half-cheetah is a 9-DOF system, with 3 unactuated floating body DOFs and 6 actuated-DOFs for the joints. The objective of this environment is to maximize forward velocity, while retaining energy efficiency. The sensory input for this environment is comprised of the relative angle and angular velocity of each joint for a total of 12 individual inputs. Originally, the environmental measurements are represented as floating point values. These measurements are then numerically clipped, converted into a binary spike representation, and sent as input into the network. The binary spike representation utilized is a probabilistic population representation based on place coding. Similar to the sensory representation, the action outputted by the network is represented by a population of spiking neurons. In each population there exists spiking neurons with equal sized positive and negative sub-populations. The total sum of spikes for each population is then individually collected and averaged over the pre-defined integration interval $T \in \mathbb{N}$. Both the equations describing the spike observation and action representation are further discussed in sections 5.2 and 5.3 of the **Appendix**.

To introduce action variance for this experiment, the output floating point value $\mathbf{A}(t)$ is used as the mean for a multivariate Gaussian with zero co-variance, $\mathbf{A}_e(t) = \mathcal{N}(\mathbf{A}(t), \exp(\boldsymbol{\sigma}_{log})^2)$. The log standard deviation, $\boldsymbol{\sigma}_{log}$, is a fixed vector that is learned along with the network parameters. To produce an action, the integration interval T was chosen to be 50 time-steps and the action sub-population size to be 100 neurons. With the action floating point dimensionality having been 6, this produced a spike-output dimensionality of 600 neurons. Additionally, using a population size of 50 neurons for each state input and a 12-dimensional input, the spike-input dimensionality was also 600 neurons. Each network model in this experiment is comprised of 2 fully-connected feed-forward hidden layers with 64 neurons each (**Figure 5**).

To compute the gradients for the network parameters we used BPTT with the surrogate gradient method Proximal Policy Optimization, altogether with the Adam optimization method (Kingma and Ba, 2014; Schulman et al., 2017). For the policy gradient, the reinforcement signal is given for each action output proportional to forward velocity, with an energy penalty on movement. This signal is backpropagated through the non-differentiable spiking neurons using SLAYER, with modifications described in section 5.1 of the **Appendix** (Shrestha and Orchard, 2018).

Five unique network types are evaluated on the locomotion task: a traditional SNN, the DP-BCM, DP-Oja's, NDP-BCM,

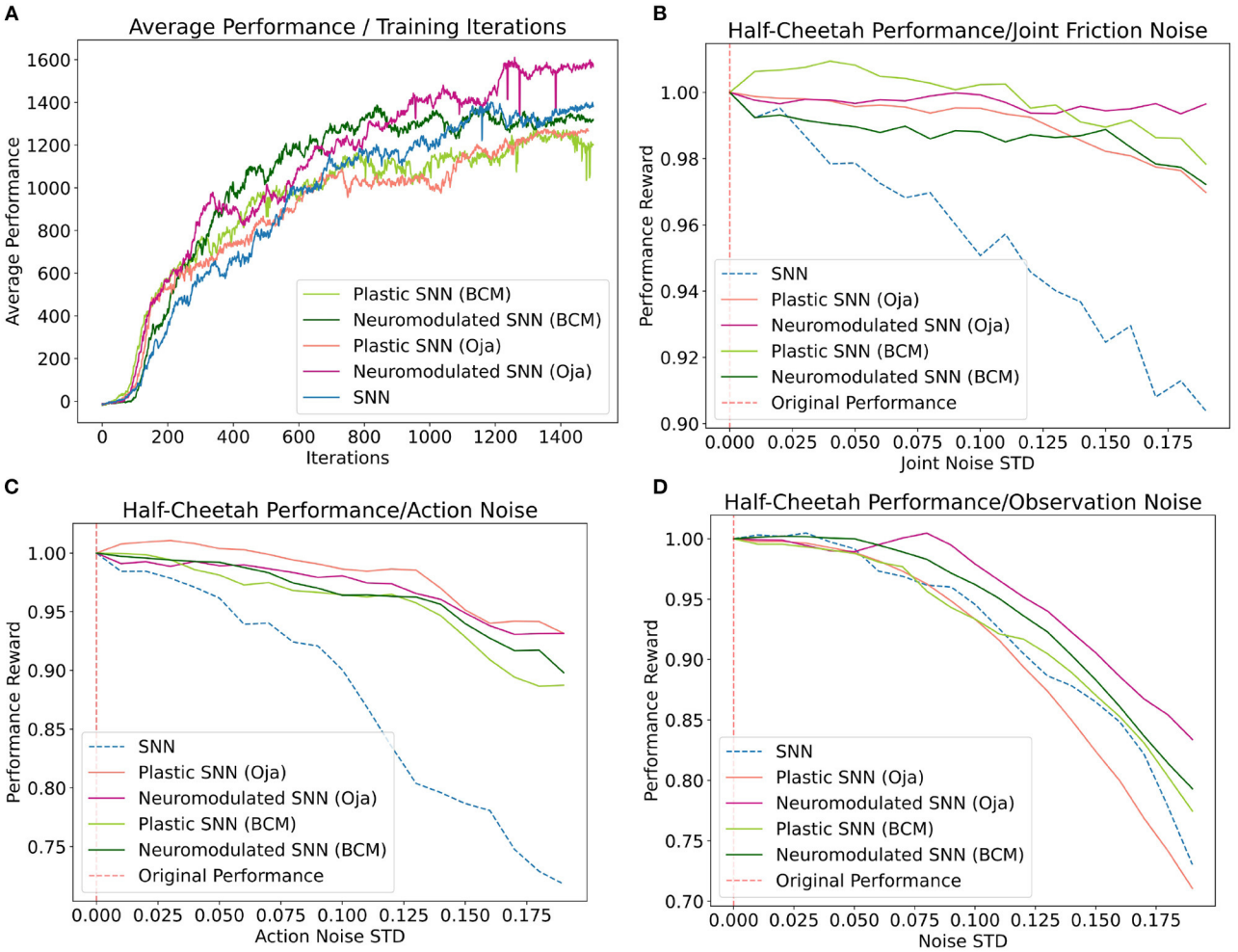


FIGURE 4 | (A) Average performance of each network at each iteration of the training process. **(B–D)** Average loss in performance as a ratio of the observed performance in response to noise [x-axis] and the original network performance in the absence of noise at x=0. Averages for each noise standard deviation were collected over 100 training episodes.

and NDP-Oja's. Three unique categories of noise are measured on the trained networks: joint friction noise, action noise, and observation noise. For each of these categories, a noise vector is sampled from a Gaussian distribution with a mean of zero and a specified standard deviation. Then, to obtain an accurate measurement, the performance for an individual network is collected and the objective is averaged over 100 performance evaluations to represent the average performance response for that network at the specified noise standard deviation (**Figures 4B–D**). Both the noise vector dimensionality and the way in which it is utilized is uniquely defined by the nature of each task. To represent observation noise, a 12-dimensional noise vector $\mathbf{z}(t) \sim \mathcal{N}(\mathbf{0}, \mathbf{1}\sigma^2)$, with a standard deviation σ which remains fixed starting at the beginning of an episode, is sampled for each observation at time t, and further summed with the respective observation before conversion to spike representation, $\mathbf{X}_z(t) = \mathbf{X}(t) + \mathbf{z}(t)$. This observation noise is in addition to the spike-firing probability noise, ϑ_{min}, described in section 5.2 of the **Appendix**. For the action noise, the same sampling process is repeated for a 6-dimensional vector, however here the noise vector plus one is element-wise multiplied with the output action, $\mathbf{A}_z(t) = \mathbf{A}(t) \odot (\mathbf{1} + \mathbf{z}(t))$. The joint friction noise rather is sampled at the beginning of a performance episode $t = 0$ and held constant throughout that episode, $\mathbf{z}(t) = \mathbf{z}(t = 0)$. The sampled noise is then further summed as a percentage of the originally specified joint friction constants $\mathbf{f}_z(t)$ at the beginning of the performance evaluation episode, $\mathbf{f}_z(t) = \mathbf{f}(t) \odot (\mathbf{1} + \mathbf{z}(t))$.

Despite each network having been trained in the absence of these noise types, the post-training performance response of the networks vary (**Figure 4**). Overall, the networks augmented with differentiable plasticity are shown to provide more effective adaptive capabilities, where minimal loss in performance was observed during the joint friction and action noise experiment for plastic networks (**Figure 4**). However, while these networks displayed improvements in robustness over the joint friction and action noise tasks,

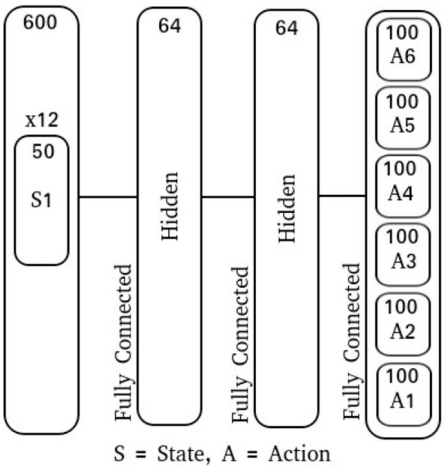


FIGURE 5 | Locomotion network diagram.

they did not display improvements on the observation noise task compared with the fixed-weight SNN. In general, the performance of NDP-Oja's was better than NDP-BCM, which may be a result of NDP-BCM's added complexity, hyper-parameter choices, network initialization, or network size. Policy gradient methods tend to favor the model which works best with the particular implementation details (Engstrom et al., 2020).

The activity of the modulatory signals in this task seem to noisily oscillate within a consistent range after the initial timestep (Figure A6 in **Appendix**). This behavior is observed to be consistent across various noise perturbations, and in the absence of them. However, when deprived of sensory input, as in the case where the robot flips on its back, modulatory oscillations cease almost completely (Figure A5 in **Appendix**). This differs from the modulatory behavior in the cue-association task where the hidden signals potentiate and decay significantly during the sensory cue sequence. A more in-depth analysis for this task is provided in the **Appendix** (section 5.8).

Training performance results for both the NDP- and DP-network variants are relatively consistent for each experiment included in **Figure 4A**. This differs from the cue-association experiment, where NDP-networks converged on the training task around 50 iterations faster than DP-networks. This experiment differs fundamentally in that the locomotion task is inherently solvable without temporal learning capabilities (Schulman et al., 2017), hence deciphering the role and benefit of plasticity and neuromodulation is not trivial. Additionally, neuromodulatory signals in biological networks do not act solely on modifying synaptic efficacy, rather have a whole host of effects depending on the signal, concentration, and region. Perhaps advances in the capabilities of NDP-networks will be a result of introducing these biologically inspired modulations (Zaninetti et al., 1999; Hosp et al., 2011; Dabney et al., 2020).

4. DISCUSSION

We have proposed a framework for learning the rules governing plasticity and neuromodulated plasticity, in addition to fixed network weights, through gradient descent on SNNs, providing a mechanism for online learning. Additionally, we have provided formulations for a variety of plasticity rules inspired by neuroscience literature, as well as general equations from which new plasticity rules may be defined. Using these rules, we demonstrated that synaptic plasticity is sufficient for solving a noisy and complex cue-association environment where a fixed-weight SNN fails. These networks also display an increased robustness to noise on a high-dimensional locomotion task. We also showed that the average spike-firing rate for DP-Oja's rule is reduced to the same observed rates seen in DP-BCM and NDP-BCM in the presence of a neuromodulatory signal, and hence more energy efficient. One potential limitation of this work is that, while gradients provide a strong and precise mechanism for learning in feed-forward and self-recurrent SNNs, there is no straightforward mechanism for backpropagating the gradients of feedback weights which are often incorporated in biologically-inspired network architectures. Another limitation is the computation cost associated with BPTT, which has a non-linear complexity with respect to weights and time. This limitation may be alleviated with truncated BPTT (Tallec and Ollivier, 2017), however this reduces gradient accuracy and hence often performance as well.

The incorporation of synaptic plasticity rules together with SNNs has a history that spans almost the same duration as SNNs themselves. The implications of a framework for learning these rules using the power of gradient descent may prove to showcase the inherent advantages that SNNs provide over ANNs on certain learning tasks. One task that may naturally benefit from this framework is in the domain of Sim2Real, where the behavior of policies learned in simulation are transferred to hardware. Often small discrepancies between a simulated environment and the real world prove too challenging for a reinforcement-trained ANN, especially on fine-motor control tasks. The improved response to noise displayed in our experimental results for DP-SNNs and NDP-SNNs on the robotic locomotion task may benefit the transfer from simulation to real hardware. Additionally, the inherent online learning capabilities of differentiable plasticity may provide a natural mechanism for on-chip learning in neurorobotic systems.

While our framework leverages the work of Miconi et al. (2019) to enter the SNN domain, this work also introduces novel results and further innovations. In the Miconi et al. (2019) experiments, results were shown for networks with plastic and neuromodulated synapses on only the recurrent weights. In their experiment, the cue association process was iterated for 200 time-steps without introducing any noise. They show that only modulatory variants of ANNs with fixed-feedforward weights and neuromodulated self-connecting recurrent weights are capable of solving this task. In our experiment, we extend a similar task to the spike domain and introduce a significant amount of sensory spike-noise. Additionally, the time dependency is more than doubled. We show that not only are

neuromodulatory feedforward weights without recurrent self-connections capable of solving this task, but also that feedforward plastic weights are. We also show that the introduction of spike-noise does not decrease training convergence. On the cue association task, we show that with Oja's rule, neuromodulatory signals drastically reduce spike-firing rates compared with the non-modulatatory variant. This reduction in activity does not apply to BCM, which has a natural mechanism for both potentiation and depression. Finally, our experiments showcase a meta-learning capability to adapt beyond what the network had encountered during its training period on a high-dimensional robotic learning task.

While our experiments showcase the performance of BCM and Oja's plasticity rules, our proposed framework can be applied to a wide variety of plasticity rules described in both the AI and neuroscience literature. Our framework may also be used to experimentally validate biological theories regarding the function of plasticity rules or neuromodulatory signals. Furthermore, the modeling of neuromodulatory signals need not be learned directly through gradient descent. Our method can be extended to explicitly model neuromodulatory signals through a pre-defined global signal. Such signals might include: an online reward signal emulating dopaminergetic neurons [Hosp et al., 2011, Hosp et al. (2011)], error signals from a control system (Popa and Ebner, 2019), or a novelty signal for exploration (DeYoung, 2013). In addition, evidence toward biological theories regarding the function of plasticity rules or neuromodulatory signals may be experimentally validated using this framework. Finally, the addition of neural processes such as homeostasis may provide further learning capabilities when interacting with differentiable synaptic plasticity. The fruitful marriage between the power of gradient descent and the adaptability of synaptic plasticity for SNNs will likely enable many interesting research opportunities for a diversity of fields. The authors see a particular enabling potential in the field of neurorobotics.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/**Supplementary Material**, further inquiries can be directed to the corresponding author/s.

AUTHOR CONTRIBUTIONS

SS designed and performed the experiments as well as the analysis and wrote the paper with JH and JA being active contributors toward editing and revising the paper. WL also provided helpful editing of the manuscript. JH had the initial conception of the presented idea as well as having supervised the project. All authors contributed to the article and approved the submitted version.

FUNDING

This work was performed at the US Naval Research Laboratory under the Base Program's Safe Lifelong Motor Learning (SLLML) work unit, WU1R36.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2021.629210/full#supplementary-material

REFERENCES

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., et al. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature* 557, 429–433. doi: 10.1038/s41586-018-0102-6

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Dhruva, T. B., et al. (2018). Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*.

Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., et al. (2020). A solution to the learning dilemma for recurrent networks of spiking neurons. *Nat. Commun.* 11:3625. doi: 10.1038/s41467-020-17236-y

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. (2015). Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*.

Bengio, Y., Mesnard, T., Fischer, A., Zhang, S., and Wu, Y. (2017). STDP-compatible approximation of backpropagation in an energy-based model. *Neural Comput.* 29, 555–577. doi: 10.1162/NECO_a_00934

Bienenstock, E., Cooper, L., and Munro, P. (1982). Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. *J. Neurosci.* 2, 32–48. doi: 10.1523/JNEUROSCI.02-01-00032.1982

Bohte, S. M., Kok, J. N., and La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing* 48, 17–37. doi: 10.1016/S0925-2312(01)00658-0

Carlson, K., Nageswaran, J., Dutt, N., and Krichmar, J. (2014). An efficient automated parameter tuning framework for spiking neural networks. *Front. Neurosci.* 8:10. doi: 10.3389/fnins.2014.00010

Cueva, C. J., and Wei, X.-X. (2018). "Emergence of grid-like representations by training recurrent neural networks to perform spatial localization," in *International Conference on Learning Representations*.

Dabney, W., Kurth-Nelson, Z., Uchida, N., Starkweather, C. K., Hassabis, D., Munos, R., et al. (2020). A distributional code for value in dopamine-based reinforcement learning. *Nature* 577, 671–675. doi: 10.1038/s41586-019-1924-6

Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., et al. (2018). Loihi: a neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38, 82–99. doi: 10.1109/MM.2018.112130359

DeYoung, C. (2013). The neuromodulator of exploration: a unifying theory of the role of dopamine in personality. *Front. Hum. Neurosci.* 7:762. doi: 10.3389/fnhum.2013.00762

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S., and Pfeiffer, M. (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)* (Killarney: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280696

Doya, K. (1999). What are the computations of the cerebellum, the basal ganglia and the cerebral cortex? *Neural Netw.* 12, 961–974.

Doya, K. (2002). Metalearning and neuromodulation. *Neural Netw.* 15, 495–506. doi: 10.1016/S0893-6080(02)00044-8

Engelhard, B., Finkelstein, J., Cox, J., Fleming, W., Jang, H. J., Ornelas, S., et al. (2019). Specialized coding of sensory, motor and cognitive variables in VTA dopamine neurons. *Nature* 570, 509–513. doi: 10.1038/s41586-019-1261-9

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., et al. (2020). Implementation matters in deep policy gradients: a case study on PPO and TRPO. *arXiv preprint arXiv:2005.12729*.

Eskandari, E., Ahmadi, A., Gomar, S., Ahmadi, M., and Saif, M. (2016). "Evolving spiking neural networks of artificial creatures using genetic algorithm," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 411–418. doi: 10.1109/IJCNN.2016.7727228

Frank, M. J., Seeberger, L. C., and O,Reilly, R. C. (2004). By carrot or by stick: cognitive reinforcement learning in parkinsonism. *Science* 306, 1940–1943. doi: 10.1126/science.1102941

Gerstner, W., Lehmann, M., Liakoni, V., Corneil, D., and Brea, J. (2018). Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Front. Neural Circ.* 12:53. doi: 10.3389/fncir.2018.00053

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.

Hosp, J., Pekanovic, A., Rioult-Pedotti, M.-S., and Luft, A. (2011). Dopaminergic projections from midbrain to primary motor cortex mediate motor skill learning. *J Neurosci.* 31, 2481–2487. doi: 10.1523/JNEUROSCI.5411-10.2011

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2020). Meta-learning in neural networks: a survey. *arXiv preprint arXiv:2004.05439*.

Hu, Y., Tang, H., Wang, Y., and Pan, G. (2018). Spiking deep residual network. *arXiv preprint arXiv:1805.01352*.

Izhikevich, E., and Desai, N. (2003). Relating STDP to BCM. *Neural Comput.* 15, 1511–1523. doi: 10.1162/089976603321891783

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. (2018). STDP-based spiking deep convolutional neural networks for object recognition. *Neural Netw.* 99, 56–67. doi: 10.1016/j.neunet.2017.12.005

Kingma, D. P., and Ba, J. (2014). ADAM: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kuśmierz, Ł., Isomura, T., and Toyoizumi, T. (2017). Learning with three factors: modulating Hebbian plasticity with errors. *Curr. Opin. Neurobiol.* 46, 170–177. doi: 10.1016/j.conb.2017.08.020

Law, C. C., and Cooper, L. N. (1994). Formation of receptive fields in realistic visual environments according to the Bienenstock, Cooper, and Munro (BCM) theory. *Proc. Natl Acad. Sci. U.S.A.* 91, 7797–7801. doi: 10.1073/pnas.91.16.7797

Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Front. Neurosci.* 10:508. doi: 10.3389/fnins.2016.00508

Liu, X., Ramirez, S., Pang, P. T., Puryear, C. B., Govindarajan, A., Deisseroth, K., et al. (2012). Optogenetic stimulation of a hippocampal engram activates fear memory recall. *Nature* 484, 381–385. doi: 10.1038/nature11028

Martin, S. J., Grimwood, P. D., and Morris, R. G. M. (2000). Synaptic plasticity and memory: an evaluation of the hypothesis. *Annu. Rev. Neurosci.* 23, 649–711.

Masquelier, T., Guyonneau, R., and Thorpe, S. J. (2009). Competitive STDP-based spike pattern learning. *Neural Comput.* 21, 1259–1276. doi: 10.1162/neco.2008.06-08-804

Miconi, T., Clune, J., and Stanley, K. O. (2018). Differentiable plasticity: training plastic neural networks with backpropagation. *arXiv [Preprint]*. arXiv:1804.02464.

Miconi, T., Rawal, A., Clune, J., and Stanley, K. O. (2019). "Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity," in *International Conference on Learning Representations*.

Mikaitis, M., Pineda Garcia, G., Knight, J. C., and Furber, S. B. (2018). Neuromodulated synaptic plasticity on the spinnaker neuromorphic system. *Front. Neurosci.* 12:105. doi: 10.3389/fnins.2018.00105

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Montague, P. R., Dayan, P., and Sejnowski, T. J. (1996). A framework for mesencephalic dopamine systems based on predictive hebbian learning. *J. Neurosci.* 16, 1936–1947. doi: 10.1523/JNEUROSCI.16-05-01936.1996

Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Thorpe, S. J., and Masquelier, T. (2018). Combining STDP and reward-modulated STDP in deep convolutional spiking neural networks for digit recognition. *arXiv preprint arXiv:1804.00227*. doi: 10.1016/j.patcog.2019.05.015

Niv, Y., Duff, M. O., and Dayan, P. (2005). Dopamine, uncertainty and TD learning. *Behav. Brain Funct.* 1, 6. doi: 10.1186/1744-9081-1-6

Oja, E. (1982). Simplified neuron model as a principal component analyzer. *J. Math. Biol.* 15, 267–273. doi: 10.1007/BF00275687

Oja, E. (1983). *Subspace Methods of Pattern Recognition*, Vol. 6. New York, NY: John Wiley & Sons.

Pavlidis, N. G., Tasoulis, O. K., Plagianakos, V. P., Nikiforidis, G., and Vrahatis, M. N. (2005). "Spiking neural network training using evolutionary algorithms," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005* (Montreal, QC: IEEE), 2190–2194. doi: 10.1109/IJCNN.2005.1556240

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774

Popa, L. S., and Ebner, T. J. (2019). Cerebellum, predictions and errors. *Front. Cell. Neurosci.* 12:524. doi: 10.3389/fncel.2018.00524

Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. (2017). Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Front. Neurosci.* 11:682. doi: 10.3389/fnins.2017.00682

Schmidgall, S. (2020). "Adaptive reinforcement learning through evolving self-modifying neural networks," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20* (New York, NY: Association for Computing Machinery), 89–90. doi: 10.1145/3377929.3389901

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science* 275, 1593–1599. doi: 10.1126/science.275.5306.1593

Shouval, H., Intrator, N., Law, C., and Cooper, L. (1970). Effect of binocular cortical misalignment on ocular dominance and orientation selectivity. *Neural Comput.* 8, 1021–1040. doi: 10.1162/neco.1996.8.5.1021

Shrestha, S. B., and Orchard, G. (2018). "SLAYER: spike layer error reassignment in time," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 1419–1428.

Sporea, I., and Grüning, A. (2012). Supervised learning in multilayer spiking neural networks. *arXiv preprint arXiv:1202.2249*. doi: 10.1162/NECO_a_00396

Tallec, C., and Ollivier, Y. (2017). Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*.

van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., et al. (2018). Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model. *Front. Neurosci.* 12:291. doi: 10.3389/fnins.2018.00291

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., et al. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nat. Neurosci.* 21, 860–868. doi: 10.1038/s41593-018-0147-8

Zaninetti, M., Tribollet, E., Bertrand, D., and Raggenbass, M. (1999). Presence of functional neuronal nicotinic acetylcholine receptors in brainstem motoneurons of the rat. *Eur. J. Neurosci.* 11, 2737–2748.

Zucker, R. S., and Regehr, W. G. (2002). Short-term synaptic plasticity. *Annu. Rev. Physiol.* 64, 355–405. doi: 10.1146/annurev.physiol.64.092501.114547

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Advantages of publishing in Frontiers

OPEN ACCESS
Articles are free to read for greatest visibility and readership

FAST PUBLICATION
Around 90 days from submission to decision

HIGH QUALITY PEER-REVIEW
Rigorous, collaborative, and constructive peer-review

TRANSPARENT PEER-REVIEW
Editors and reviewers acknowledged by name on published articles

Frontiers
Avenue du Tribunal-Fédéral 34
1005 Lausanne | Switzerland

Visit us: www.frontiersin.org
Contact us: frontiersin.org/about/contact

REPRODUCIBILITY OF RESEARCH
Support open data and methods to enhance research reproducibility

DIGITAL PUBLISHING
Articles designed for optimal readership across devices

FOLLOW US
@frontiersin

IMPACT METRICS
Advanced article metrics track visibility across digital media

EXTENSIVE PROMOTION
Marketing and promotion of impactful research

LOOP RESEARCH NETWORK
Our network increases your article's readership