NEUROMORPHIC ENGINEERING SYSTEMS AND APPLICATIONS

EDITED BY: Chiara Bartolozzi, Emre O. Neftci and Elisabetta Chicca
PUBLISHED IN: Frontiers in Neuroscience

frontiers Research Topics

frontiers

ISSN 1664-8714
ISBN 978-2-88971-723-1
DOI 10.3389/978-2-88971-723-1

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

NEUROMORPHIC ENGINEERING SYSTEMS AND APPLICATIONS

Topic Editors:
Chiara Bartolozzi, Italian Institute of Technology (IIT), Italy
Emre O. Neftci, University of California, Irvine, United States
Elisabetta Chicca, University of Groningen, Netherlands

Citation: Bartolozzi, C., Neftci, E. O., Chicca, E., eds. (2021). Neuromorphic Engineering Systems and Applications. Lausanne: Frontiers Media SA.
doi: 10.3389/978-2-88971-723-1

Table of Contents

frontiers in Neuroscience

TECHNOLOGY AND CODE
published: 08 January 2020
doi: 10.3389/fnins.2019.01338

Sepia, Tarsier, and Chameleon: A Modular C++ Framework for Event-Based Computer Vision

Alexandre Marcireau[1], *Sio-Hoi Ieng*[1]* *and Ryad Benosman*[1,2,3]

[1] INSERM UMRI S 968, Sorbonne Universites, UPMC Univ Paris 06, UMR S 968, CNRS, UMR 7210, Institut de la Vision, Paris, France, [2] University of Pittsburgh Medical Center, Pittsburgh, PA, United States, [3] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, United States

This paper introduces an new open-source, header-only and modular C++ framework to facilitate the implementation of event-driven algorithms. The framework relies on three independent components: *sepia* (file IO), *tarsier* (algorithms), and *chameleon* (display). Our benchmarks show that algorithms implemented with *tarsier* are faster and have a lower latency than identical implementations in other state-of-the-art frameworks, thanks to static polymorphism (compile-time pipeline assembly). The *observer pattern* used throughout the framework encourages implementations that better reflect the event-driven nature of the algorithms and the way they process events, easing future translation to neuromorphic hardware. The framework integrates drivers to communicate with the *DVS*, the *DAVIS*, the *Opal Kelly ATIS*, and the *CCam ATIS*.

Keywords: silicon retinas, event-based sensing, development framework, event-based processing, asynchronous computation

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Arren Glover,
Italian Institute of Technology (IIT), Italy
Alpha Renner,
ETH Zurich, Switzerland

***Correspondence:**
Sio-Hoi Ieng
siohoi.ieng@gmail.com

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *03 July 2019*
Accepted: *27 November 2019*
Published: *08 January 2020*

Citation:
Marcireau A, Ieng S-H and Benosman R (2020) Sepia, Tarsier, and Chameleon: A Modular C++ Framework for Event-Based Computer Vision. Front. Neurosci. 13:1338. doi: 10.3389/fnins.2019.01338

1. INTRODUCTION

Event-based cameras are fundamentally different from conventional cameras (Posch et al., 2014). Conventional, frame-based cameras integrate light at fixed time intervals, and produce spatially dense frames. By contrast, the pixels of event-based sensors are asynchronous and independent. Each pixel outputs data only when the visual information in its field of view changes, mimicking biological sensing (Liu and Delbruck, 2010). Event-based cameras output their events in the order they are produced, resulting in a spatially sparse sequence with sub-millisecond precision. This fundamental difference in data nature calls for different computational strategies (Delbruck et al., 2010).

This work introduces an end-to-end framework for designing and running event-based algorithms for computer vision. The code components are written in C++, and are open-source. The presented method and implementation outperform state-of-the-art frameworks while encouraging better semantics for event-based algorithms. Special care is given to modularity and code dependencies management, in order to facilitate portability and sharing.

1.1. Event-Based Cameras

Bio-inspired cameras aim at mimicking biological retinas, as the latter greatly outperform conventional, frame-based systems (Liu and Delbruck, 2010). Many architectures have been implemented over the years, including pulse-modulation imaging (Chen et al., 2011), smart vision chips (Dudek and Hicks, 2005; Carmona-Galán et al., 2013), and event-based sensors. The framework presented in this paper primarily targets event-based sensors.

The pixels of event-based sensors contain analog circuits implementing signal processing calculations. Upon meeting a specific condition, the analog circuit emits an output transmitted to the computer. The most widespread type of calculation is brightness change detection. The pixel's photodiode output is continuously monitored to detect significant variations. When the logarithmic luminance changes beyond a fixed threshold, the pixel sends an event to the computer. This event bundles spatial and temporal information, as well as a boolean polarity encoding whether the significant change corresponds to an increase or decrease in brightness. Several sensors contain pixels implementing this behavior, including the *DVS* (Dynamic Vision Sensor) (Lichtsteiner et al., 2008), the *cDVS* (Berner and Delbruck, 2011), the *ATIS* (Asynchronous Time-based Image Sensor) (Posch et al., 2010), and the *DAVIS* (Dynamic and Active-pixel Vision Sensor) (Brandli et al., 2014). They are still under active development, with improved versions featuring lower latency (Lenero-Bardallo et al., 2011), higher sensitivity (Delbruck and Berner, 2010; Serrano-Gotarredona and Linares-Barranco, 2013; Yang et al., 2015), or more pixels (Son et al., 2017). **Figure 1** highlights the difference between a sequence of frames and a stream of polarity events recorded from the same scene.

The *cDVS* and *ATIS* differ from the *DVS* by their extended pixel circuits generating a second type of polarity events, besides change detection. The polarity bit of the second event type encodes another visual information. The *cDVS* triggers such events on wavelength changes, whereas the *ATIS* encodes absolute exposure measurements in the time difference between them. The DAVIS is a hybrid sensor: it features both a *DVS*-like circuit and a light integration circuit. The latter produces frames similar to those generated by a conventional sensor. Huang et al. (2017) present another event-based sensor, the *CeleX*, with a behavior similar to that of a *DVS*: events are triggered by brightness changes. However, output events include an absolute exposure measurement encoded on 9 bits instead of a binary spolarity.

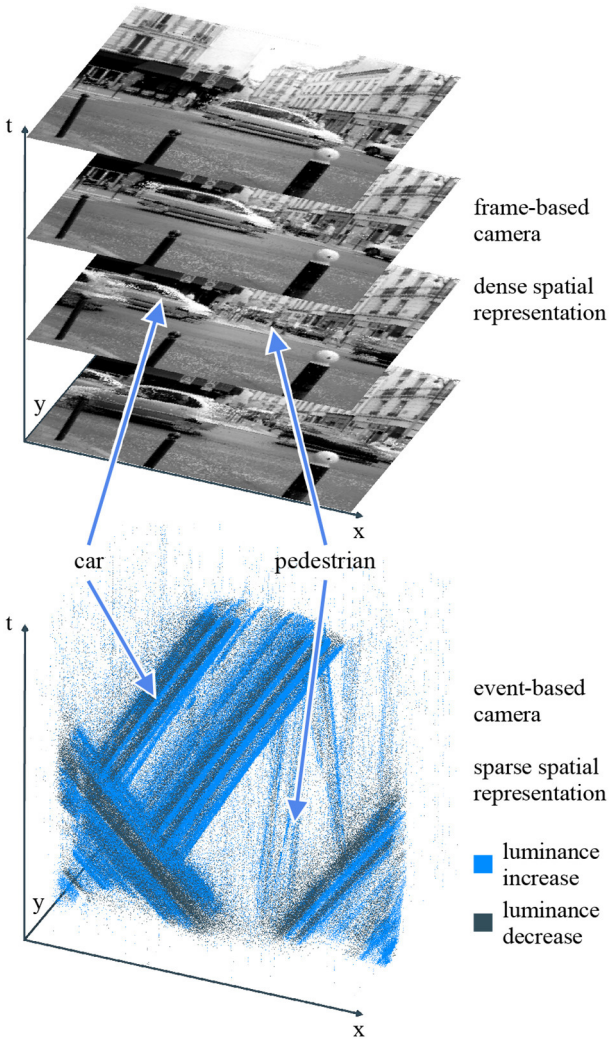


FIGURE 1 | Conventional cameras (top) capture dense frames at fixed time intervals. Event-based cameras (bottom) have independent pixels which asynchronously output information when the luminance in their individual field of view changes. This sparse representation yields a better temporal resolution and a smaller bandwidth. Some computer vision tasks, such as moving objects segmentation, become easier. The point cloud representation is a still frame of a standalone HTML widget generated by our framework.

1.2. Event-Based Computer Vision

There are three approaches to information extraction from the output of event-based cameras. The first one consists in generating spatially dense frames from the sensor output in a way that preserves temporal resolution. The frames can then be fed to conventional computer vision algorithms (Amir et al., 2017; Maqueda et al., 2018). The second approach advocates short calculations triggered by each event, and requires a rethink of computer vision from the ground up (Benosman et al., 2012; Lagorce et al., 2015; Reverter Valeiras et al., 2016; Mueggler et al., 2017). By matching the sensor data format, this approach benefits from the sensor advantages, notably resemblance to biological signals, low latency, and data compression. Spiking neural networks fit the constraints of the second approach, and several event-based computer vision algorithms were implemented on neural simulators (Galluppi et al., 2012; Orchard et al., 2015; Haessig et al., 2018; Hopkins et al., 2018). The third approach mixes frames and events, and is well-suited to hybrid sensors, such as the *DAVIS* (Barranco et al., 2014; Moeys et al., 2016; Tedaldi et al., 2016). The framework presented in this paper is designed to encourage the second approach, though it applies to the third as well.

Given the issues arising from the Von Neumman architecture of modern computers (Indiveri and Liu, 2015), dedicated hardware seems required for event-based vision systems to match the performance of their biological counterparts. Nevertheless, microprocessors remain the de facto standard to perform general-purpose computations. They benefit from years of research and development, making them cost-effective, computationally-efficient, and user-friendly. As such, they are great tools for algorithms prototyping and early applications of event-based sensors. Furber (2017) envisions heterogeneity

in future processors: general-purpose cores will work together with dedicated hardware accelerators. Under this assumption, a framework targeting CPUs is not a mere temporary solution waiting to be replaced by neural networks, but a decision support tool. It provides a baseline for algorithms power consumption and computational cost, against which implementations running on dedicated hardware can be compared. Thus, the accelerators can be chosen based on the gain they yield for tasks deemed important. A framework designed for CPUs must provide fast implementations in order to be an effective baseline. Moreover, its syntax should reflect the constrains of hardware dedicated to event-based calculations, to ease comparisons and facilitate algorithms ports from one platform to the other.

1.3. Frameworks

A software framework provides a collection of operators and a way to assemble them to build complex algorithms. We consider three types of frameworks related to event-based computer vision. First, we present frameworks for conventional computer vision and their limits when working with event-based data. Then, we examine event-based programming, showing how its concepts apply to event-based computer vision, even though existing frameworks were designed under constraints so different from event-based sensors that they cannot be used directly. Finally, we review frameworks dedicated to event-based computer vision.

The applications of linear algebra to a wide variety of science and engineering fields triggered, early in computer science history, the development of efficient libraries to compute matrix operations (Lawson et al., 1979). Conventional computer vision libraries use matrices to represent frames, allowing algorithms to be expressed as a sequence of operations on dense data (Thompson and Shure, 1995; Bradski, 2000; Jones et al., 2001). Dynamic, high-level languages can often be used to specify the operators order. The overhead incurred by the dynamic language is negligible when compared to the matrix operations. The latter are optimized by the underlying linear algebra library, yielding a development tool both efficient and user-friendly. Event-based computer vision is a different story. Small computations are carried out with each incoming event, and the cost of the glue between operators stops being negligible. Hence, the very structure of the libraries designed for conventional computer vision is incompatible with events, besides dealing with dense frames instead of sparse events.

Unlike event-based computer vision, event-driven programming languages and frameworks are not new: Visual Basic dates back to the 1990s. Among the concepts developed for event-driven programming, the event handler pattern and the observer pattern (Ferg, 2006) are natural choices to represent event-based algorithms and event-based cameras. Reactive programming (Bainomugisha et al., 2013), devised has a refinement over event-driven programming, introduced new abstractions to avoid state-full event-handlers and explicit time management. However, the neurons we aim at mimicking are state-full (the reaction to an input spike—for example, an output spike—depends on the current membrane potential), and fine control over time management is a needed feature for real-time systems. Hence, we choose to design our framework using event-driven rather than reactive concepts. Modern event-driven frameworks have notable applications in graphical user interfaces and web servers (Tilkov and Vinoski, 2010), where events represent user interactions and HTTP requests, respectively. The number of events per second reached in these applications is very small when compared to event-based cameras. On the one hand, a user clicking or typing does not generate much more than tens to hundreds of events per second (Cookie Clicker, 2013), and a large website, such as *Twitter* handles about six thousand requests per second on average[1]. On the other hand, an *ATIS* moving in a natural environment generates about one million events per second, with peaks reaching next to ten million events per second. The relatively small number of events the existing frameworks were designed to handle makes their design incompatible with event-based computer vision. For example, Javascript event handlers can be attached or detached at run-time, greatly improving flexibility at the cost of a small computational overhead whenever an event is dispatched.

All the frameworks dedicated to event-based computer vision circumvent the aforementioned problem using event buffers transmitted from operator to operator. The buffers typically contain a few thousand events spread over a few thousand microseconds. A typical operator loops over the buffer and applies some function on each event. The operator output consists in one or several new event buffers, looped over by subsequent operators. The sequence is dynamically defined at run-time, incurring a computational overhead. However, this cost is paid with every buffer instead of every event, becoming negligible as is the case with conventional computer vision frameworks. The first event-based computer vision framework, Jae (2007), was designed for the *DVS* and is written in Java. Subsequent cameras and their increased event throughput triggered the development of C and C++ frameworks: Cae (2007), recently re-factored and renamed DV (2019) (both from *iniVation*), *kAER*[2] (from *Prophesee*), and *event-driven YARP* (Glover et al., 2018a,b) (developed for the *iCub*). **Table 1** highlights the design differences between these frameworks. The table also includes *tarsier*, the computation component of the framework presented in this work. Unlike the other frameworks, it assembles operators at compile-time, suppressing the need for buffers between components, even though event buffers are still used to communicate with cameras or the file system.

1.4. Paper Structure

This paper presents the frameworks components in the order they intervene in an actual pipeline, starting with an overall view (section 2). We introduce event-driven programming concepts and shows how they apply to event-based computer

[1] *CBS News Uses Twitter as Part of Its News Investigating and Reporting* (2015). Available online at: https://developer.twitter.com/en/case-studies/cbs-news

[2] *kAER 0.6*, used in this work, is the latest version developed by our laboratory and licensed to *Prophesee*. Newer versions are now developed and maintained by *Prophesee* and the source codes are for internal use only, hence their performances are not assessed in this work.

TABLE 1 | Various C/C++ frameworks provide tools to build event-based algorithms.

Name	Open source	Operators connection	Dependencies	Communication and execution	Event types
tarsier (this work)	Yes	Compile-time, C++ templates	–	Event-wise function calls, single thread	Template event types, contiguous memory
cAER	Yes	Run-time, XML	Boost, libpng, libusb, libuv	Event buffers, single thread	Hard-coded event types, contiguous memory
kAER	No	Run-time, C++/Python	Boost, OpenCV, Python, Qt	Event buffers, constant time intervals, single thread	Hard-coded event types, contiguous memory
Event-driven YARP	Yes	Run-time, C++/XML	Libace	IP packets, multiple programs	Hard-coded event types, contiguous memory or polymorphic event types, non-contiguous memory
Dynamic vision system	Yes	Run-time, XML	Boost, libusb, OpenCV, OpenSSL	Event buffers, multiple threads	Hard-coded event types, contiguous memory

Despite an identical goal and programming language, they are build upon very different design decisions. Differences impact users' interaction with the framework and the performance of algorithms implementations. YARP uses IP packets, which makes it possible to run an algorithm in parallel on several machines, but adds overhead when running on a single computer.

vision (section 3), followed by a brief description of *sepia*, the component implementing functions to read and write event files. Section 4 presents the design and implementation of *tarsier*, a collection of event-based algorithms. Benchmarks are used to compare its performance with existing event-based computer vision frameworks (section 5). Section 6 describes *chameleon*, a collection of Qt components to display events on a conventional screen. The implementation of drivers to communicate with event-based cameras, non-feed-forward architectures and considerations on parallelism are exposed (section 7), before discussing future work and our conclusions (section 8).

2. FRAMEWORK OVERVIEW

The framework presented in this work supports Linux, macOS, and Windows. It is organized in independent components, named after animals with unusual eyes. They work together by following the same conventions, even though they have no explicit link. This structure, illustrated in **Figure 2**, reduces to a minimum the external dependencies of each component, and promotes modularity. In particular, several components solely require a C++ compiler, facilitating code sharing between various machines and operating systems, and usage with other libraries. The framework's three major components are *sepia* (file I/O), *tarsier* (algorithms), and *chameleon* (display). Since these components are independent, one may use any of them without the others. For example, *sepia* can be used to read and write event files on an operating system lacking Qt support.

The framework's libraries are header-only: they require neither pre-compilation nor system-wide installation, and several versions of the library can co-exist on the same machine without interfering. Bundling dependencies with algorithms makes projects more likely to keep working over long periods of time without active support, which we believe is a critical factor for research. Moreover, an algorithm and all its dependencies can be shipped in a single zip file, making code easy to share as the supplementary material of a publication (as illustrated by this paper's **Supplementary Material**). Header-only libraries also simplify MSVC support for Windows (Barrett, 2014), removing the need for GCC ports, such as MinGW.

All the code is open-source, and hosted on our GitHub page (section 8). Each framework component is hosted on a distinct repository, and documented in the associated Wiki page. More importantly, the *tutorials* repository provides step-by-step tutorials and commented examples to build event-driven applications with the framework.

3. EVENT-DRIVEN PROGRAMMING

3.1. A Generic Event-Based Algorithm

The object-oriented *observer* pattern consist in two constructs: an observable and an event handler. The former dispatches events at arbitrary times, whereas the latter responds to each event with an action. This pattern provides a natural model for an event-based camera (observable) and an algorithm (event handler). It extends to neuron models (for example, integrate-and-fire), though implementing complex networks with feedback and delays—which can change the events order in time—is not straightforward (section 7 provides considerations on this topic). Algorithm 1 gives a generic expression of an event-based algorithm under this paradigm.

A framework reflecting this theoretical expression facilitates algorithms implementation. A function (in the programming sense) which takes an event as sole parameter and returns nothing has a syntax close to Algorithm 1. Such a function has to mutate a state to do something useful, thus it is not a function in the mathematical sense (it is non-pure).

Algorithm 1: A generic event-based algorithm, or event handler.

```
initialize the state
on event do
    instructions mutating the state
end
```

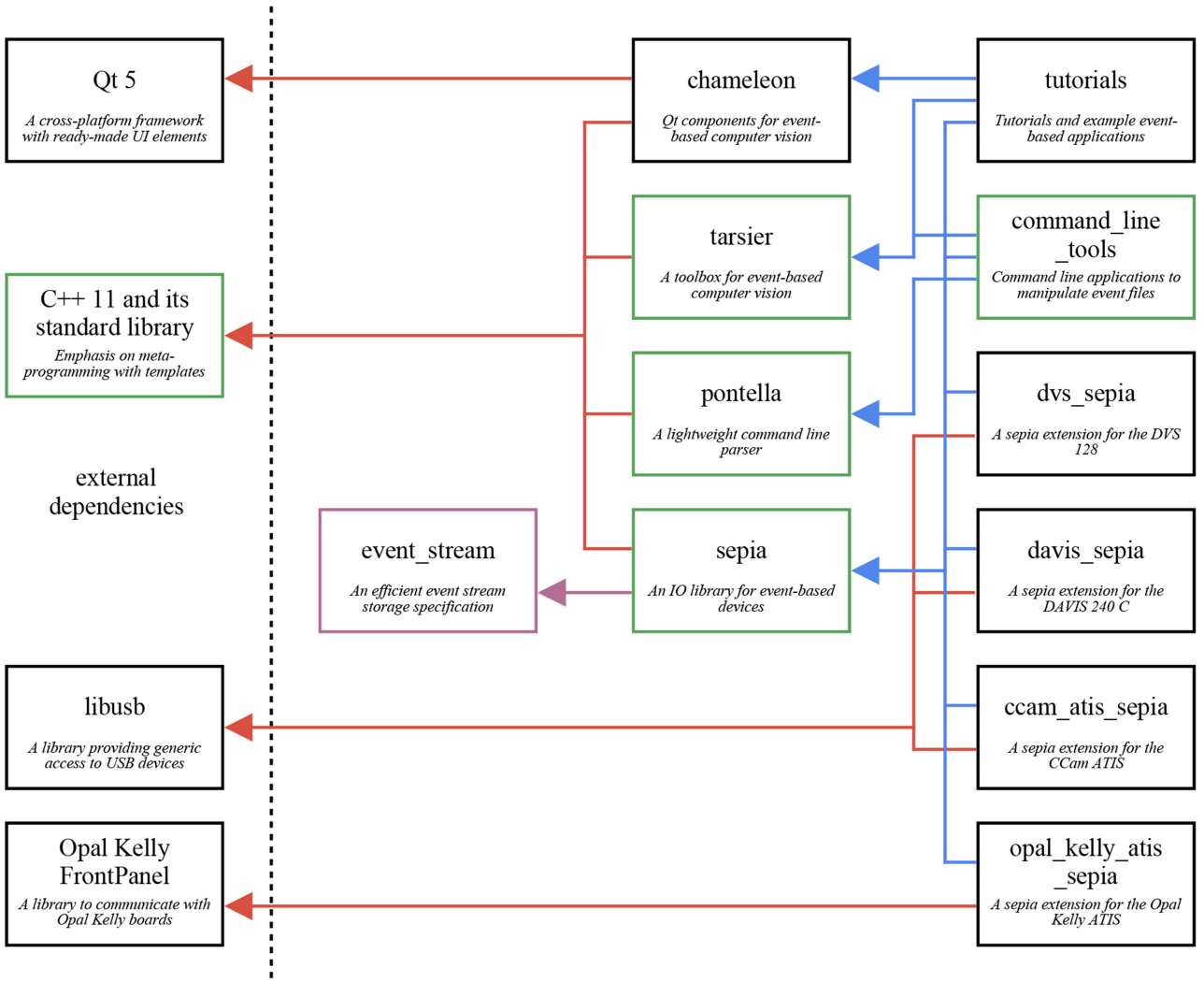


FIGURE 2 | The framework presented in this paper is a collection of three independent components: *sepia* (file IO), *tarsier* (event-based algorithms), and *chameleon* (displays). Each component is hosted on its own repository, and serves a specific goal. This graph shows the three components, their external dependencies, and other repositories dependent on the framework. The *event_stream* component (purple) is not a library but a file format specification, detailed in the **Appendix**. The components shown in green have no external dependencies but the C++ Standard Template Library.

3.2. C++ Implementation

The typical C++ implementation of the observer pattern relies on dynamic polymorphism: the event handler inherits a generic class, and the observable holds a pointer to an instance of this class. This approach creates overhead for two reasons. On the one hand, every call to an event handler requires a vtable lookup and an extra dereferencing. On the other, the compiler is given less information to optimize the program.

Existing frameworks (cAER, kAER, event-driven YARP, and Dynamic Vision System) solve this issue using buffers: events are handled thousands at a time, reducing overhead proportionally. In return, user-written handlers (called *modules* in cAER, Dynamic Vision System and event-driven YARP, and *filters* in kAER) have to loop over buffers of events. Manipulating buffers, though good for performance, may foster practices that deepen the gap with neuromorphic hardware: using events ahead in the buffer to improve performance, as they are "already there," and viewing the events as pieces of data rather than function calls. The former makes the conversion to neuromorphic hardware harder (the algorithm uses future events, increasing memory usage and latency waiting for them), while the latter strips away the event meaning (a model of a hardware spike).

The presented framework relies on static polymorphism, using templates (Veldhuizen, 2000): the event handler is bound to the observable during compilation. This approach does not incur an overhead with every event, therefore buffers are not needed. The algorithm is specified by a loop-free function, illustrated in **Figure 3**. We want to emphasize that the code presented in this figure is a complete program, which can be compiled without prior libraries installation. The function

`handle_event` modifies the state of the `std::cout` object, captured implicitly as a global variable. Events are read from the file `"input.es"`, which uses the *Event Stream* encoding (see the **Appendix**).

The *sepia* header used in this example implements file IO in the framework, and can be extended to communicate with cameras (section 7). Even though it relies on buffers, similarly to the other C++ frameworks, the event loop is hidden from the user. This is meant to reconcile two somewhat paradoxical objectives: provide a fast implementation on CPUs, which work best with bulk data, and encourage an algorithm design likely to translate to highly distributed neuromorphic systems with fine-grained calculations.

Static polymorphism is implemented in sepia using the same approach as the C++ Standard Template Library (see, for example, the *compare* function of the *std::sort* algorithm). Besides being efficient, it allows compile-type, type-safe "duck typing": the code will compile as long as the syntax `handle_event(event)` is valid. Notably, `handle_event` can be a function, a lambda function or an object with an overloaded call operator. Lambda functions are great to quickly prototype an event-driven algorithm, as shown in **Figure 4**. This second example is a standalone, dependency-free program as well. The state variables `previous_t` and `activity` are captured by reference in the lambda function. The latter implements a sensor-wide "leaky integrate" neuron to estimate the activity, printed after processing all the input file's events.

```
#include "sepia.hpp"
#include <iostream>

void handle_event(sepia::dvs_event dvs_event) {
    std::cout << (dvs_event.is_increase ? "+" : "-");
}

int main() {
    sepia::join_observable<sepia::type::dvs>(
        sepia::filename_to_ifstream("input.es"),
        handle_event);
    return 0;
}
```

FIGURE 3 | This code snippet is the "hello world" program of the *sepia* library. The function `handle_event` prints a plus sign in the terminal on luminance increase events, and a minus sign on luminance decrease events. The main program creates an observable from a file, with the `handle_event` function as event handler. This program, provided in **Supplementary Material**, only needs the *sepia* library in its directory to be compiled on any machine.

The `sepia::join_observable` function blocks until all the events are processed, preventing other routines (notably Graphical User Interfaces) from running. Under the hood, it uses the GUI-compatible `sepia::make_observable` function, which dispatches events on another thread. In turn, this function constructs a `sepia::observable` object. The latter's constructor cannot be called directly, because C++ does not allow class template deduction from a constructor (until C++ 17). Thanks to the *make* function, the event handler type does not have to be explicitly specified. However, the event handler must be statically specified—not unlike connections in a neural network. Changing the event handler at run-time requires an explicit if-else block within the handler.

Both the `sepia::join_observable` and `sepia::make_observable` functions require a template parameter: the expected event type. The event handlers signature is check at compile-time, whereas the file events type is checked at run-time (each *Event Stream* file contains a single type of events).

The event handlers presented thus far have several shortcomings: they use global variables, can be used only with specific event types, and cannot be easily used from other algorithms. The *tarsier* library tackles these issues.

4. BUILDING BLOCKS

Basic blocks that can be assembled into complex algorithms are the central feature of a framework for computer vision. They reduce development time and foster code reuse: components debugged and optimized by an individual benefit the community.

```
#include "sepia.hpp"
#include <iostream>

int main() {
    auto previous_t = 0ull;
    auto activity = 0.0f;
    sepia::join_observable<sepia::type::dvs>(
        sepia::filename_to_ifstream("input.es"),
        [&](sepia::dvs_event dvs_event) {
            activity *= std::exp((dvs_event.t - previous_t) / -1000.0f);
            activity += 1.0f;
            previous_t = dvs_event.t;
        });
    std::cout << activity << std::endl;
    return 0;
}
```

FIGURE 4 | Unlike **Figure 3**, this program uses a lambda function to implement an event handler. Lambda functions can be declared inside the main function, keeping the global scope clean. This event handler implements a leaky neuron to compute the activity. The latter is printed once all the event from the source file `"input.es"` have been processed.

4.1. Partial Event Handlers

In order to represent a building block for event-based algorithms, we introduce the concept of partial event handler, illustrated by the Algorithm 2. A partial event handler is triggered by each event, similarly to the complete event handler defined in subsection 3.1. However, instead of consuming the event, the partial event handler performs a calculation, then conditionally triggers a second handler.

Algorithm 2: A partial event handler.

```
initialize the state
on event do
    instructions mutating the state
    if condition then
        trigger another event handler with a new event
    end
end
```

Using functions to represent handlers, we denote f_* a partial event handler. Since f_* generates events, it is an observable for a complete event handler g. Binding g to f_* yields the complete event handler f_g. When called, it performs the calculations associated with f_*, then calls g. Any number of partial event handlers can be chained to build an algorithm, as long as the last handler is complete. For example, with g_* now a partial event handler, and h a complete event handler, one can build the pipeline f_{g_h}. For each child, its direct parent is an observable generating events. For each parent, its child is a complete event handler (g_h is a complete event handler and a child for f_*). The syntax can be extended to partial event handlers generating multiple event types: $f_{*,*}$ is a partial event handler with two observable types.

A more common approach to defining algorithms consists in specifying inputs and outputs for each block. However, since a partial event handler conditionally generates (possibly) multiple event types, a generic output is a list of pairs {event, boolean} representing optional objects[3]. Each boolean indicates whether the event was generated. The program assembling the pipeline would contain a complex sequence of function calls and nested if-else statements to propagate only events that were actually generated. Nested observables yield a syntax both easier to read and more closely related to the event-driven nature of the algorithm.

f_{g_h} is written $f \rightarrow g \rightarrow h$ in figures to avoid nested indices. Complex pipelines, including merging and feedback, are discussed in section 7.

4.2. *tarsier* Implementation

The framework's *tarsier* library is a collection of partial event handlers implemented in C++. Each handler is declared and defined in a single header file: only the included ones are compiled with the program. This organization makes the code resilient to compatibility errors in unused handlers.

The partial handlers are implemented as classes with an overloaded call operator. The children handlers types are templated. In order to allow type deduction, each class is associated with a *make* function: the partial event handler f_* is associated with *make_f*. For any complete event handler g, $make\_f(g) := f_g$. Pipelines are built by nesting *make* functions: $make\_f(make\_g(h)) = f_{g_h}$. Unlike event handlers, the high-order *make* functions are pure. Most of them take extra parameters to customize partial event handlers. For example, `tarsier::make_mask_isolated`, which builds a partial event handler propagating only events with spatio-temporal neighbors, takes a sensor width and height and a time window as parameters. **Figure 5** shows a simple *tarsier* pipeline, bound to a *sepia* observable.

The *tarsier* and *sepia* libraries are compatible even though they are not explicitly related. Every partial event handler provided by *tarsier* uses template event types, besides template event handlers parameters. The event type has to be specified explicitly (`sepia::dvs_event` in **Figure 5**), and must have a minimal set of public members which depends on the event handler (often x, y and t). A C++ struct with at least these three fields meets the requirements of most *tarsier* handlers. Users can define custom types to best represent the events output by their algorithms (flow events, activity events, line events, time surfaces…), or to customize the events payload (with a camera index for stereo-vision, sparse-coding labels…).

This implementation has several benefits. Since the pipeline is assembled statically, type checks are performed by the compiler. Missing event fields and incompatible observable/event handler bindings are detected during compilation, and meaningful errors are returned (in contrast with run-time segfaults). Moreover, an event loaded from disk or sent by a camera, with a specific type, can be used directly without an extra copy to a buffer holding events with another type. Since the compiler manipulates a completely specified pipeline, it can perform more powerful code optimizations. Finally, since static event handler calls have no run-time overhead, events buffers can be traversed depth-first instead of breadth-first (**Figure 6**). This operation ordering reduces the pipeline latency, as observed in section 5.

5. COMPARATIVE BENCHMARKS

Event-based computer vision shows promise for real-time sensing on robots (Blum et al., 2017). If a CPU is used to run computer vision algorithms on a robot, the code efficiency can make the difference between a real time and non-real time system. Performance is also essential to make realistic comparisons of conventional hardware and neuromorphic hardware, or to compare two event-based CPU algorithms. Even though the average number of operations per event gives an estimation of an algorithm complexity, it does not account for compiler optimizations, memory IO or processor optimizations (branch predicting, cache…).

[3]Using C++ STL primitives, the output's type would be `std::tuple<std::pair<event_type_0, bool>, std::pair<event_type_1, bool>, ...>`. With the C++ 17 standard, `std::optional<event_type>` can be used instead of pairs.

```cpp
#include "mask_isolated.hpp"
#include "mirror_x.hpp"
#include "sepia.hpp"
#include "shift_y.hpp"
#include <iostream>

int main() {
    const auto header = sepia::read_header(sepia::filename_to_ifstream("input.es"));
    sepia::join_observable<sepia::type::dvs>(
        sepia::filename_to_ifstream("input.es"),
        tarsier::make_mask_isolated<sepia::dvs_event>(
            header.width,
            header.height,
            1e3,
            tarsier::make_mirror_x<sepia::dvs_event>(
                header.width,
                tarsier::make_shift_y<sepia::dvs_event>(
                    header.height,
                    10,
                    [](sepia::dvs_event dvs_event) {
                        std::cout << (dvs_event.is_increase ? "+" : "-");
                    })))));
    return 0;
}
```

FIGURE 5 | This program uses both *sepia* and *tarsier*. It can be compiled on any computer without installing external libraries. The pipeline is implemented as a sequence of nested partial event handlers. `tarsier::mask_isolated` removes noisy events, `tarsier::mirror_x` inverts the *x* coordinate and `tarsier::shift_y` shifts the *y* coordinate by a fixed offset. Events outside the original window after shifting are not propagated.

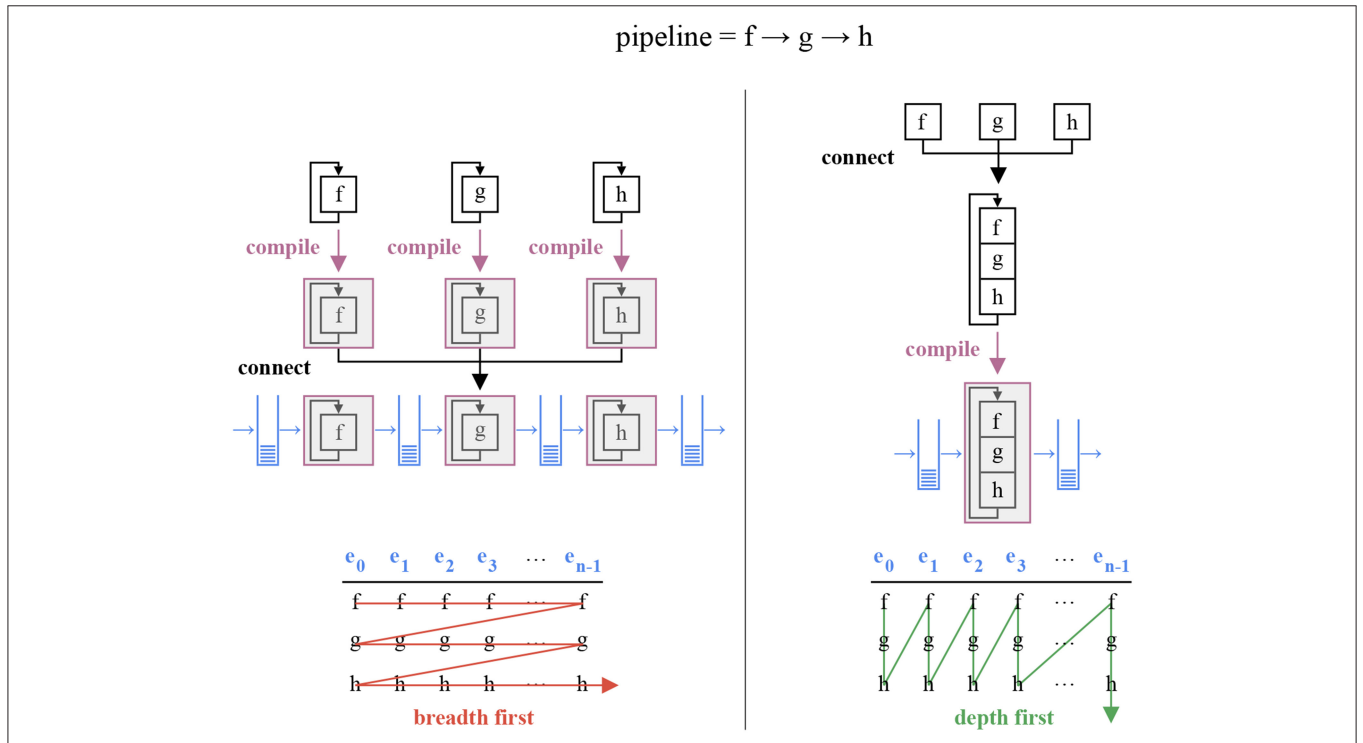


FIGURE 6 | The $3 \times n$ operations associated with a sequence of three event handlers *f*, *g*, and *h* and a buffer of *n* events $e_i, i \in [0 \dots n-1]$ can be performed in two orders: breadth first and depth first. An implementation relying on dynamic polymorphic incurs an overhead for every distinct function call, and must therefore use the breadth first approach (left). Depth first yields lower latencies, but requires static polymorphism: the pipeline must be assembled during compilation (right).

Hence, accurate speed comparisons require a comparison of implementations, whose result depends on the quality of the implementations.

The efficiency of an implementation depends on many parameters, including the algorithm itself, the choice of programming language, the use of suitable programming primitives, and the properties of the framework. We aim to compare the contribution of the latter among frameworks designed for event-based computer vision. We restrict this comparison to frameworks written in C/C++, to avoid

comparing languages rather than frameworks. The compared algorithms are given the same implementation in each framework, thus observed differences can only be attributed to frameworks properties.

The present benchmarks focus on event processing: the *tarsier* library is compared to its counterparts in *cAER*, *kAER*, and *event-driven YARP*. The other frameworks components (file IO, camera drivers and display) are not considered. Moreover, we were not able to include *Dynamic Vision Systems* in the benchmarks: its current implementation uses multiple threads and circular FIFOs between modules. Modules running faster than their children overflow the FIFO, resulting in silent event loss. Though not critical for real-time applications, this loss biases benchmark results and prevents graceful program termination, which depends on exact event counting. Nevertheless, since the structural design choices of *Dynamic Vision Systems* are similar to those of *cAER*, we expect comparable results. *Event-driven YARP* offers two implementations for event buffers: vectors and *vQueues*. Vectors leverage contiguous memory, whereas *vQueues*, which are double-ended queues of pointers to polymorphic events, support arbitrary types. We evaluate the performance of both options. The results associated with the vector (respectively *vQueue*) implementation are labeled *YARP* (respectively *YARP vQueue*). The code used to run the benchmarks is available online (section 8). This resource also illustrates the implementation of the same algorithms in various frameworks.

Before each benchmark, we load a specific stream of events in memory. The events are organized in packets of up to 5,000

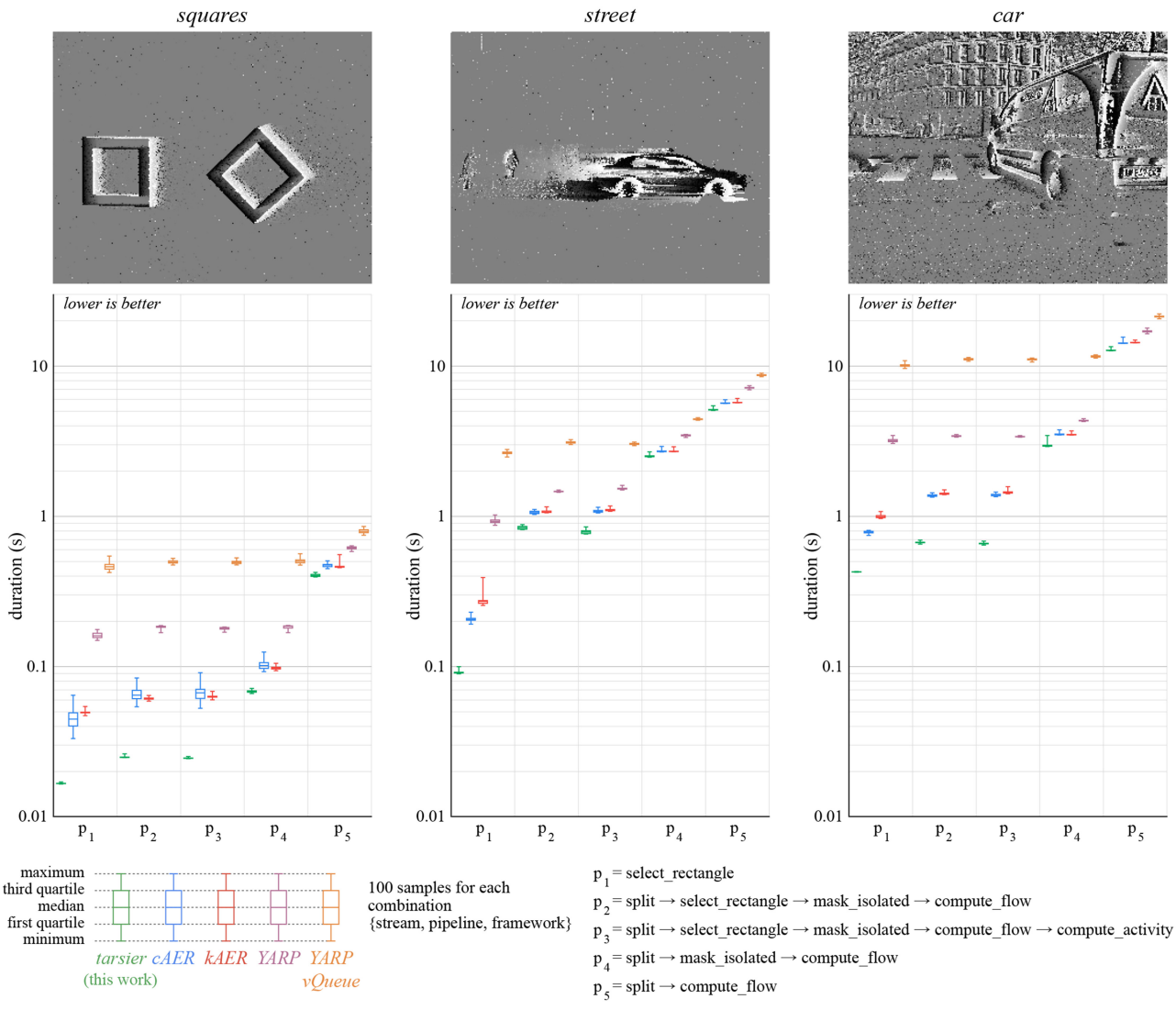


FIGURE 7 | We implement the same partial event handlers in each framework in order to compare them. We consider five pipelines and three event streams. The total time it takes to handle every event from the input stream is measured 100 times for each condition. We attribute the better performance of *tarsier* to static polymorphism, which yields a program with fewer memory operations.

events and up to 10 ms (a new packet is created as soon as either condition is met), as to mimic the typical output of a camera. We consider two performance indicators. The *duration* experiment measures the total time it takes to read the packets from memory, run an algorithm and write the result back to memory. It indicates how complex a real-time algorithm can be. The *latency* experiment measures the time elapsed between the moment a packet is available and the moment results are written to memory. A packet is made available when the wall clock time goes past the timestamp of its last event. A busy-wait loop is used to wait for the wall clock time if the framework is ready to handle a packet before the latter is available. This mechanism simulates the output of an actual event-based camera while avoiding putting processes to sleep, which is a source of non-deterministic variations in the measured latency. The packets contain `sepia::dvs_event` objects, chosen as a neutral type for all the frameworks. Event type conversions, if needed, are taken into account in the performance measurement. This choice is not an unfair advantage to *tarsier*, since its handlers are compatible with any event type (including the types provided by *sepia*). The events dispatched from one partial event handler to the next are framework-dependent. However, to avoid uneven memory writes, the output events are converted to a common type before being pushed to a pre-allocated vector. To make sure

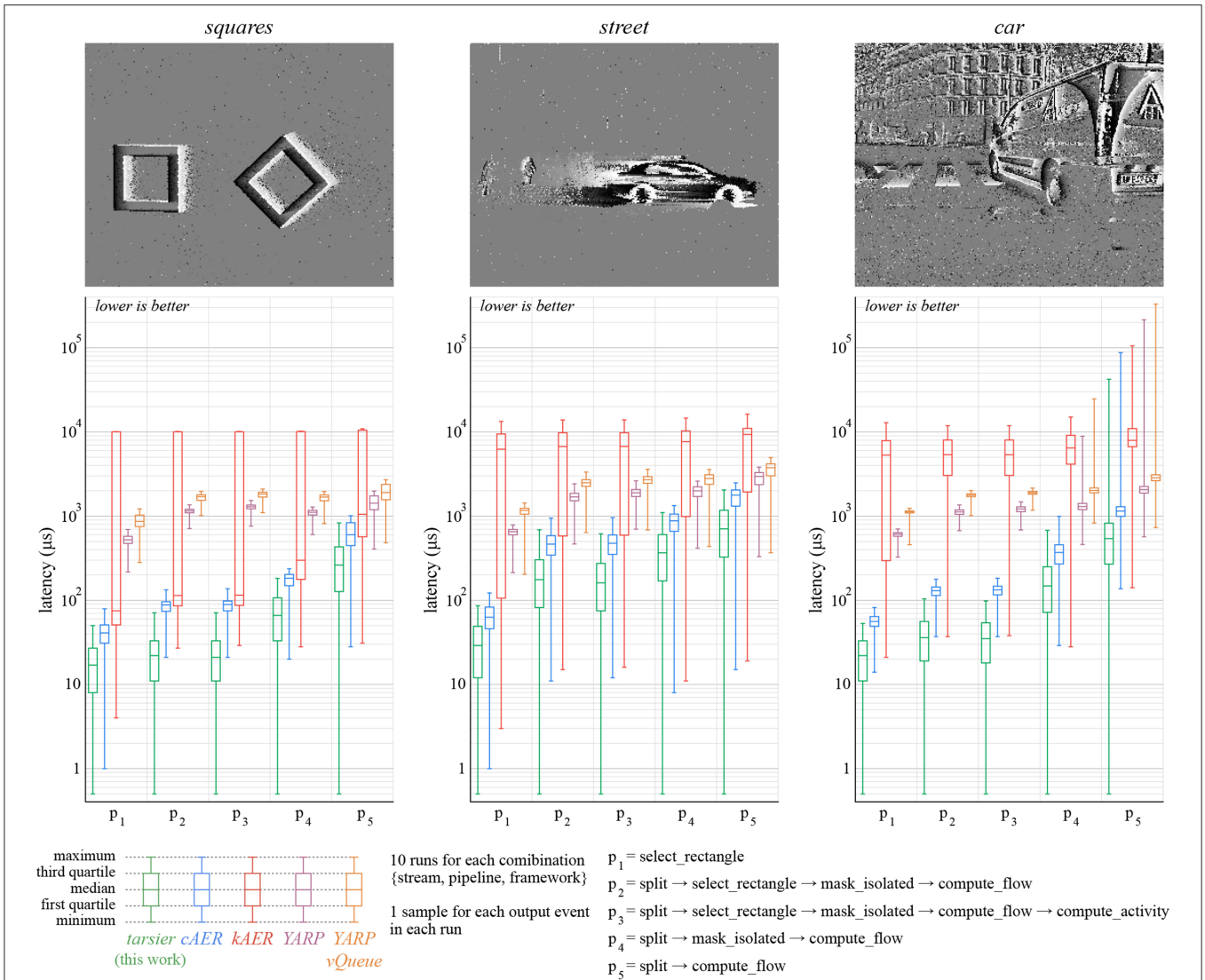


FIGURE 8 | Low-latency is an important feature of event-based cameras, and therefore event-based frameworks. We measure the time elapsed between the moment a buffer is available and the moment associated output events are produced by the pipeline. Events that are not propagated by the pipeline (for example, removed noise) are not taken into account. For each condition, latency is measured for each output event over 10 runs of the whole stream. We attribute the better performance of *tarsier* to depth-first traversal. *kAER* under-performs in this benchmark since it constrains buffers duration, unlike the camera model assumed in the benchmarks: the resulting buffer reorganization increases delays. This benchmark's relative variations are larger than the *duration* benchmark's variations. The same time measurement functions are used, however durations are order of magnitude larger than latencies.

TABLE 2 | We use three event streams recorded by an ATIS to perform benchmarks.

Stream name	Description	Duration (s)	Event rate (s^{-1})
Squares	Artificial scene, moving geometric shapes, fixed sensor	9.50	2.83e5
Street	Natural scene, moving pedestrians and cars, fixed sensor	50.6	3.17e5
Car	Natural scene, sensor inside a moving car	69.6	9.56e5

The streams were chosen for their different conditions (artificial and natural scenes, fixed and moving sensor) and average event rates.

that the output is not skipped by the compiler as an optimization, we calculate the MurmurHash3 (Appleby, 2014) of each output field once the algorithm completed. The resulting values are controlled for each benchmark run, and guarantee that each implementation calculates the same thing.

The benchmarks use five distinct algorithms (p_1 to p_5) described (**Figures 7**, **8**). Each pipeline is assembled from one or several of the following partial event handlers:

- `select_rectangle` only propagates events within a centered 100×100 pixels window.
- `split` only propagates events representing a luminance increase.
- `mask_isolated` only propagates event with spatio-temporal neighbors.
- `compute_flow` calculates the optical flow.
- `compute_activity` calculate the pixel-wise activity. The activity decays exponentially over time, and increases with each event.

We use three event streams, listed in **Table 2** and available in the benchmarks' repository. These streams contain polarity events recorded by an ATIS, in both controlled and natural environments. The *duration* experiment is run one hundred times for each combination {stream, pipeline, framework}, and the *delay* experiment ten times. Each *delay* task generates many samples, whereas each *duration* task yields a single value. All 6,600 tasks are shuffled, to avoid possible biases, and run sequentially on a computer running Ubuntu 16.04 LTS with an Intel Core i7-6700 CPU @ 3.40GHz CPU and a 16 GB Hynix/Hyundai DDR4 RAM @ 2.4 GHz. The code is compiled with GCC 5.5, C++11 and the -O3 optimization level.

5.1. Duration

The *duration* benchmark results are illustrated in **Figure 7**. The approach presented in this paper yields the smallest duration on all the pipelines and event streams considered. This improvement over state-of-the-art frameworks can notably be attributed to a reduced number of memory reads and writes, thanks to the template event types.

Event-driven YARP yields longer durations than the other frameworks. The difference is most likely related to the use of IP packets to communicate between filters. The alternative implementation *event-driven YARP vQueue* is substantially worse with respect to the considered benchmarks. We attribute the performance loss to the non-contiguous memory allocation of events in *vQueues*. The other frameworks use either multiple hard-coded event types (*cAER*, *kAER*, *event-driven YARP*), or template event types (*tarsier*) to leverage contiguous memory.

The pipeline p_3 contains more operations than p_2. Yet, the p_3 *tarsier* implementations has a smaller duration than p_2 (the effect is most visible with the *street* stream). The `compute_activity` event handler does not utilize the visual speed calculated by `compute_flow`, only the flow events' timestamp and position. Therefore, the flow computation can be skipped without changing the algorithm outcome. In the case of frameworks with modules assembled at run-time, the compiler cannot make this simplifying assumption. We believe this behavior can improve the performance of complex pipelines, where finding redundant or unused calculations manually can prove difficult.

5.2. Latency

The *latency* benchmark results are illustrated (**Figure 8**). Wall clock time is measured with microsecond precision for each input packet and each output event. Latency samples are calculated by subtracting the wall clock time of output events and that of their input packet. In some cases, the latency is zero, meaning that the actual elapsed wall clock time is smaller than the measurements' precision. To allow representation on a log-scale, we round up null latency samples to 0.5 µs.

The relative standard deviation is much higher for the *latency* benchmark than the *duration* one. As a matter of fact, measured values are much smaller: durations are in the order of seconds, whereas latencies are on the order of microseconds. Thus, every small non-time-deterministic operation (memory operations, CPU prediction, kernel preemption…) has, relatively, more impact.

The *kAER* framework yields substantially larger latencies than the other frameworks. Since it enforces buffers with a constant duration, latency increases when the buffers provided by the camera use a different, possibly variable, duration.

The framework presented in this paper outperforms the others in this benchmark as well. Low-latency can be a major benefit for robots or closed-loop systems. The performance gain is a consequence of buffer depth-first traversal and the reduced number of memory operations, since inter-handler communication is not implemented with buffers. The latency reduction improves with the duration of the algorithm when comparing *tarsier* and *cAER*, as illustrated in **Figure 9** (top graph).

However, the latency variance is larger for *tarsier* than *cAER*, and increases with the pipeline duration as well. This is another consequence of depth-first traversal: the first event in the input buffer is handled as soon as the packet is available, and therefore has a small latency. In contrast, the last event in the buffer waits for all the other events to be handled, resulting in a much larger latency. This phenomenon does not exist with *cAER* since the whole packet is processed by each module sequentially:

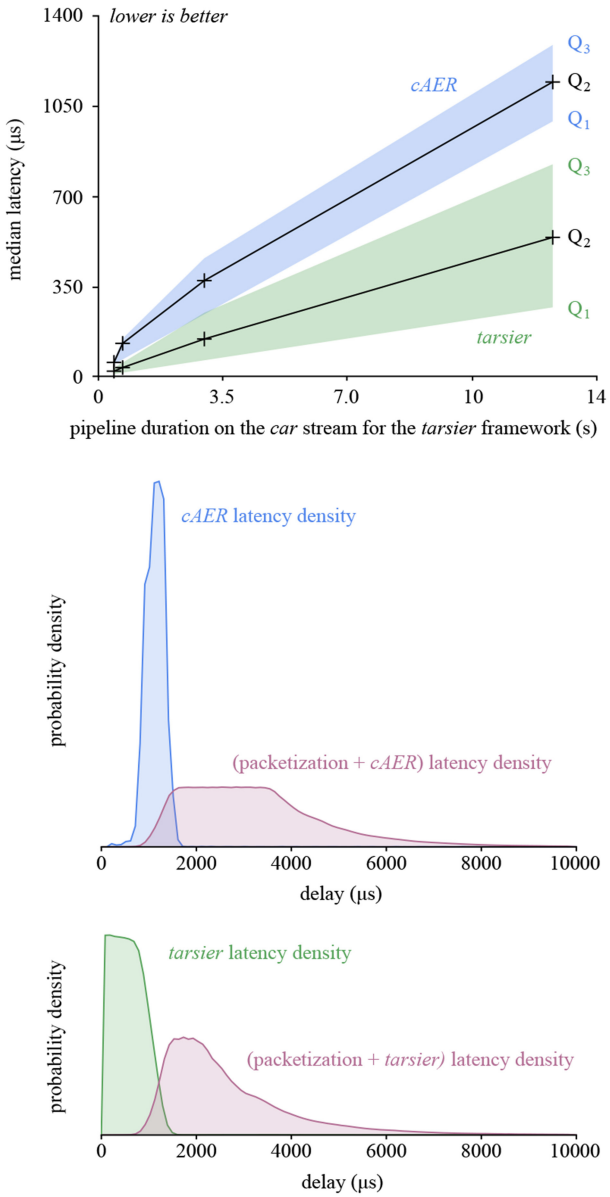


FIGURE 9 | The graphs presented in this figure take a closer look at the latency created by *tarsier* and *cAER* for the *car* stream. In the top graph, latency is plotted as a function of pipeline duration when run with *tarsier* (arbitrarily chosen as a complexity indicator). *tarsier* has a smaller median density, but a larger variance. The density probability for the most complex pipeline is plotted in the middle and bottom graphs (blue and green). It accounts only for framework latency (as does the first graph). Adding the latency caused by packetization in the camera (before the USB transfer) yields the total latency. The depth-first traversal leveraged by *tarsier* better counterbalances packetization, resulting in both a lower total latency and a smaller variance.

events with the same input packet exit the pipeline at the same time.

The latency used so far takes only the framework into account. The first event of each buffer is also the one that waited the most in the camera while the input buffer was being filled. If we neglect the USB transfer duration, we can define the total latency associated with an event as the sum of the framework latency and the timestamp difference between the last event in the packet and the considered event. The total latency as well as its variance are both smaller for *tarsier* when compared with *cAER*, since the packetization effect is counterbalanced by the depth-first traversal. Both the framework latency and total latency densities are illustrated in **Figure 9** (bottom graphs).

6. EVENT DISPLAYS

Conventional screens display frames at fixed time intervals[4]. In order to display events, one has to perform a conversion. Most frameworks rely on fixed time windows: a frame pixel is colored in white if it was the source of a luminance increase event during the associated time interval, in black if the luminance decreased, and in gray if nothing happened. This approach does not account for the high temporal resolution of the signal. Another method relies on time decays (Cohen, 2016; Lagorce et al., 2017): the frame pixel i is given the color $c_i = \frac{1}{2}\left(1 + \delta_i \cdot exp\left(-\frac{t-t_i}{\tau}\right)\right)$. t is the current timestamp. t_i is the timestamp of the most recent event generated by the pixel i. $\delta_i = 1$ if the last event generated by i corresponds to a luminance increase, and -1 otherwise. τ is a fixed decay. **Figure 10** illustrates the difference between the two methods, highlighting the benefits of exponential decays.

The full-frame decay rule requires an exponential calculation on every event for every pixel (for an ATIS, 72,960 pixels a million times per second), which is both unrealistic and unnecessary, since the typical display features a 50 Hz refresh rate. Instead, one can calculate the decays only when a frame is about to be rendered, and use the GPU available on most machines to do so. GPUs are designed to run massively parallel calculations with every frame, thus are well-suited to this task.

The *chameleon* library provides Qt (1995) components to build event displays. The components are independent and header-only. Unlike *sepia* and *tarsier*, *chameleon* cannot be used without *Qt 5*. In return, the event displays can easily be integrated into complex user interfaces. The `chameleon::dvs_display` implements the full-frame decay method mentioned previously. This component assumes two threads: an event loop (for example, a *sepia* observable followed by a *tarsier* pipeline) and a render loop. The loops communicate using a shared memory with one cell per pixel, where the last timestamp and polarity of each event is stored. When a new frame is about to be rendered, the shared memory is sent to an OpenGL program to compute each pixel's time decay. The shared memory is accessed millions of times per second by the event loop. Usual mutexes can cause non-negligible overhead, since they rely on system calls. The *chameleon* implementation uses spin-lock mutexes instead (essentially busy-wait loops with atomic variables), at the cost of increased CPU usage. To minimize the strain on the event

[4]Recent screens compatible with Nvidia's G-Sync technology can display frames at varying time intervals, narrowing the gap between frames and events. Exponential decays can be used to convert events to frames compatible with such screens.

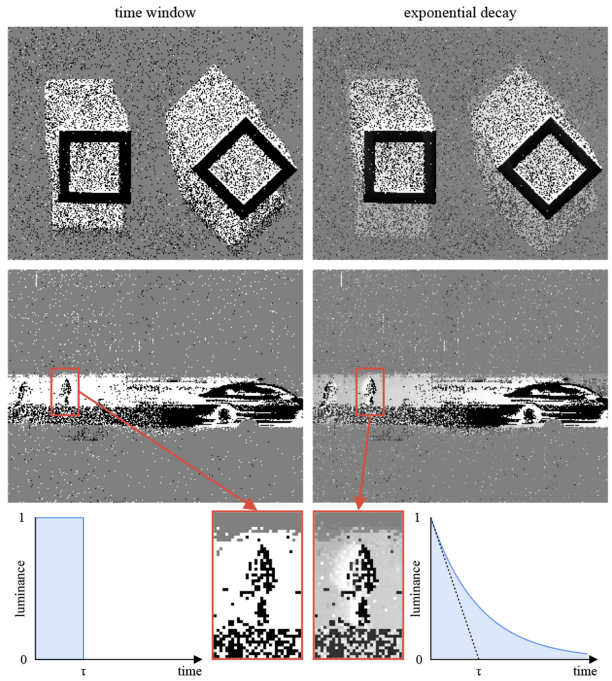


FIGURE 10 | This figures compares two strategies to convert events to frames for display. The time window approach (left) degrades temporal information: the still frames do not hold enough information to determine the geometric shapes motions (top row) or the relative speed of the car and the pedestrian (bottom row). The exponential decay approach (right) represents temporal information with gray levels. It is computationally more expensive than the time window approach, but can be easily implemented on a GPU to relieve the CPU.

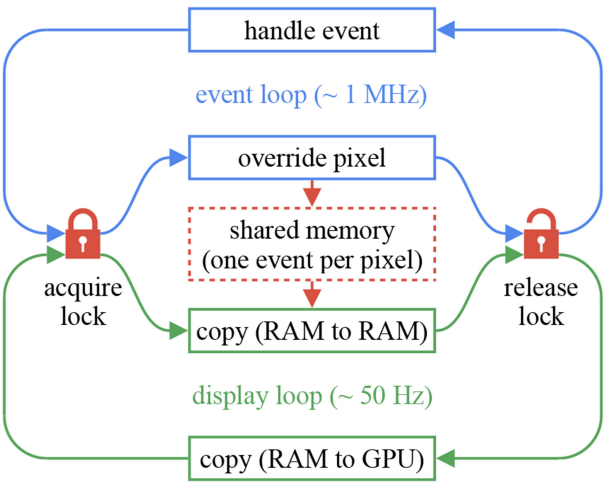


FIGURE 11 | In order to convert events to frames, one has to reconcile the very different rates of the event loop (about 1 MHz) and the display loop (often 50 Hz). We use a shared memory the size of the sensor, protected by thread-safe locks. On each event, the first thread (blue) overwrites former events with the same spatial coordinates. Every time a frame is about to be rendered, the display loop (green) copies the shared memory to RAM and releases the lock, then communicates with the GPU. The memory-to-memory copy minimizes lock ownership, to avoid blocking the event loop. The lock, acquired with every event, is implemented as a spin-lock mutex.

loop, the render loop first creates a local copy of the shared memory, then releases the mutex, and finally communicates with the GPU. This mechanism is illustrated in **Figure 11**. **Figure 12** gives an overview of an application build with the three major components of the framework, with a focus on thread management. This application's code is available in the *tutorials* repository.

The proposed approach does not rely on pre-defined frame boundaries: the frame-rate matches the display rate regardless the event loop speed. Consequently, the visual animation remains smooth even if the event pipeline is slower than real time. A smooth slow-motion display can be created by artificially slowing down the event loop.

The colors used by the DVS display can be customized: the c_i value is then used as a weight parameter for mixing the colors. Transparent colors can be used, enabling display overlays for cameras generating multiple stream types (such as the ATIS or the DAVIS). Other notable components provided by *chameleon* include a vector field display (well-suited to flow events), a blob display, a time delta display (to represent the absolute exposure measurements of an ATIS), and a screen-shot component to easily create frame-based videos. These components use template event types, similarly to *tarsier* event handlers, and the type requirements follow the same conventions.

The displays coordinates system follows the usual mathematical convention, with the origin located at the screen's lower-left pixel. The usual computer vision convention (origin at the upper-left pixel) is not used as it is a result of the matrix representation of frames, which event-based algorithms aim to avoid.

7. FRAMEWORK EXTENSIONS

7.1. Camera Drivers

Since most event-based cameras feature a USB interface, their drivers can be devised as user-space programs atop a third-party library overseeing the USB communication. To keep the codebase modular and minimize dependencies, each camera interface is held in a distinct repository extending the *sepia* library.

As of now, the following cameras are supported:

- DVS 128. We re-implemented the *libcaer* interface to provide out-of-the-box MSVC support.
- ATIS (Opal Kelly board). This extension depends on the non-free Opal Kelly Front Panel library.
- ATIS (CCam 3). This camera has the same pixels and arbiter as the Opal Kelly ATIS, however it features a custom FPGA and a USB 3 interface. It was designed by Prophesee.
- DAVIS 240C. We re-implemented the *libcaer* interface for this sensor as well.

Event-based cameras have internal buffers to store events while waiting for a USB transfer. A camera generating events at a

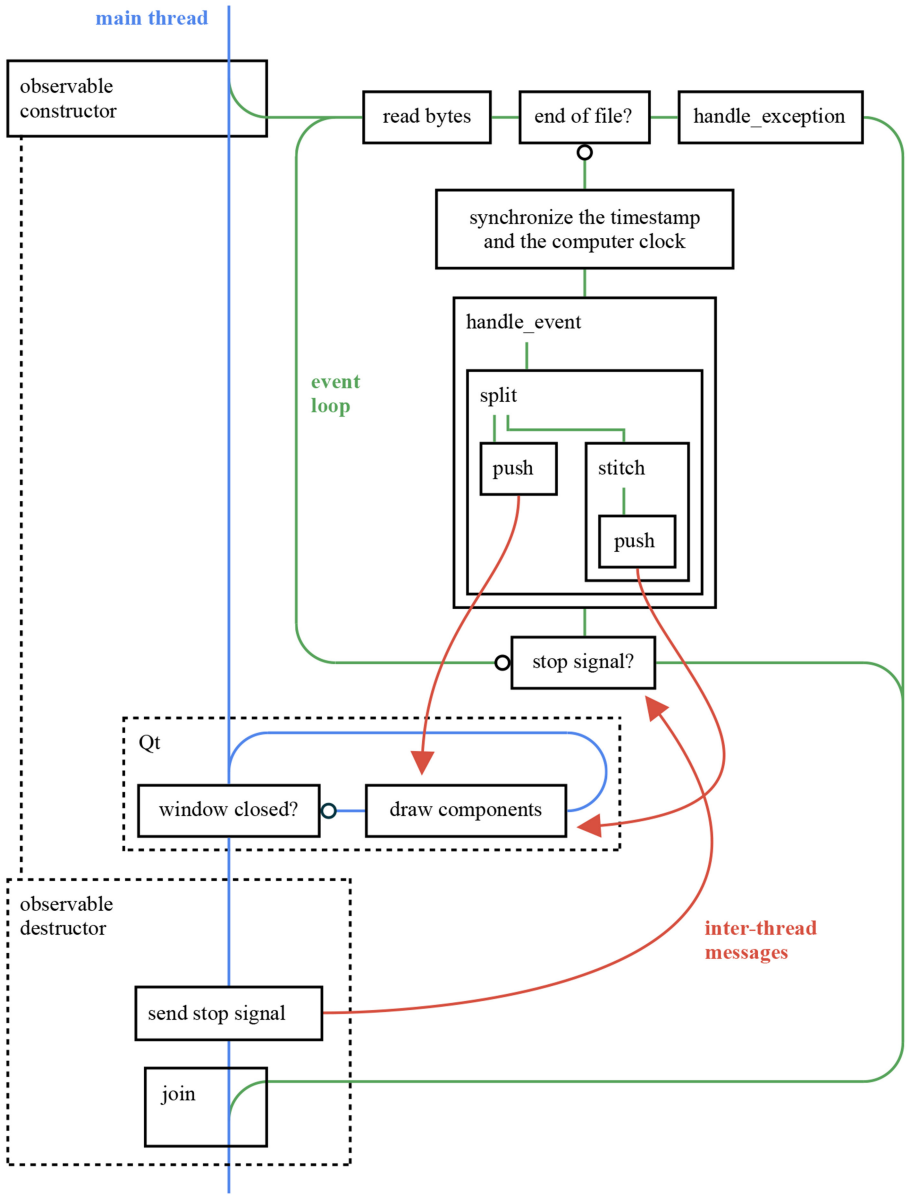


FIGURE 12 | This figure provides an overall view of the threads in an ATIS event stream viewer application using *sepia*, *tarsier*, and *chameleon*. The application reads an *Event Stream* file and displays it as frames. The observable constructor (respectively destructor) creates (respectively joins) the event loop thread, in accordance with the RAII (Resource acquisition is initialization) philosophy of C++. The *push* inter-thread messages rely on the mechanism illustrated in **Figure 11**, whereas the *stop* signal is implemented as an atomic boolean. The code for this application can be found in the *tutorials* repository.

faster rate than what the computer program can handle ends up filling its internal buffers to capacity. At this points, cameras either drop new events or shuts down. To circumvent this issue, each *sepia* extension uses an extra thread to communicate with the camera, independently of the event loop executing the algorithm. The two threads communicate with a thread-safe circular FIFO. An overall view of the threads of an application using a *sepia* extension, *tarsier* and *chameleon* is given in **Figure 13**. The circular FIFO implementation is provided by *sepia*.

Multiple parameters can be specified to configure an event-based camera, such as the operating mode or the current biases. JSON files are used by *sepia* extensions to specify the configuration. The *sepia* header implements a JSON parser and validator to load configuration files and warn users in case of syntax errors or unknown parameters.

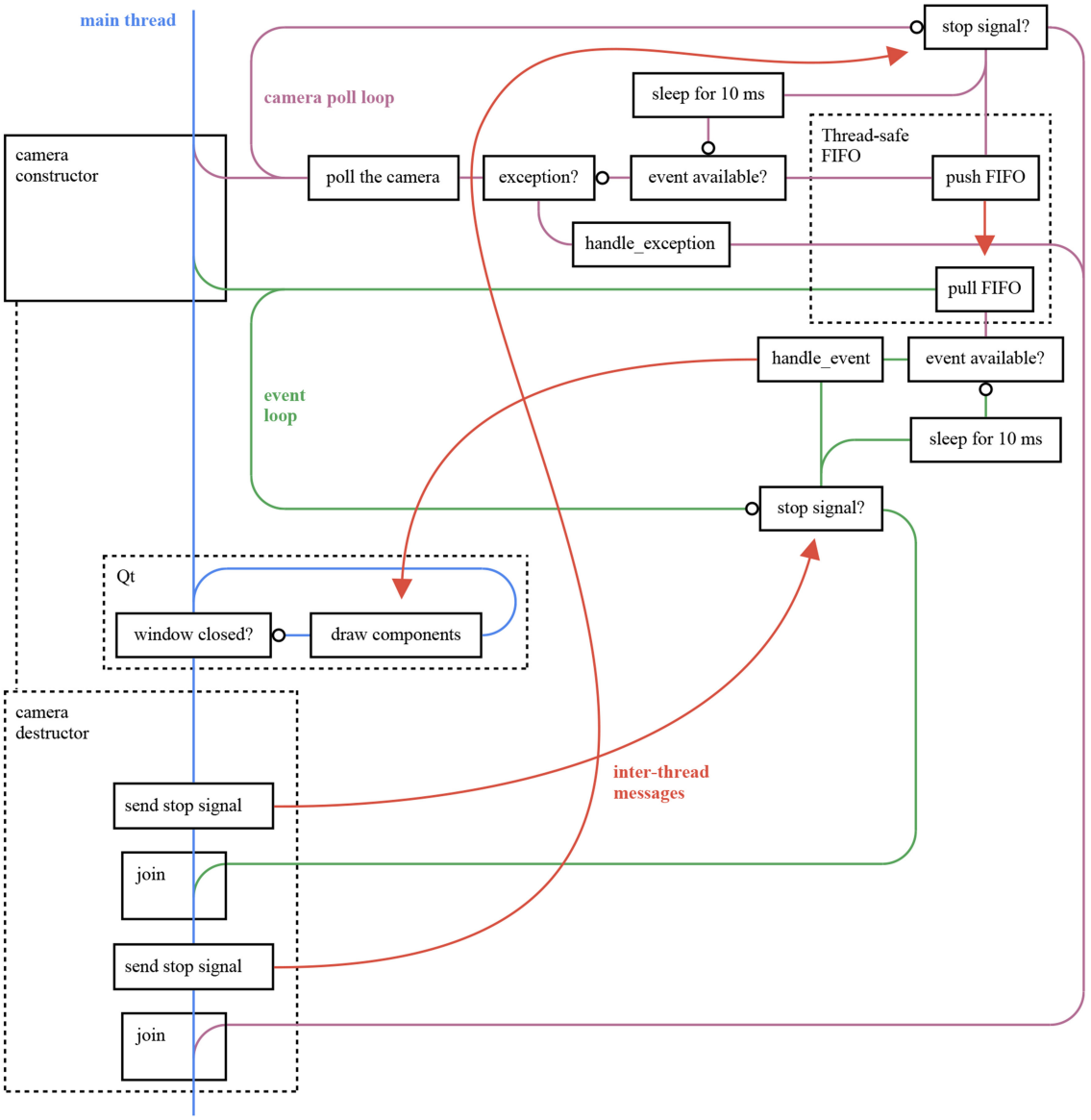


FIGURE 13 | This figure provides an overall view of the threads in an ATIS camera viewer application using *opal_kelly_atis_sepia*, *tarsier*, and *chameleon*. The application listens to a camera and displays the generated events as frames. It encompasses the application illustrated in **Figure 12**. The extra thread is used to communicate with the camera as fast as possible even when the event loop is busy. The two threads communicate through a thread-safe FIFO buffer implemented in *sepia*. The Opal Kelly Front Panel library does not provide a *poll* function, hence the explicit *sleep* step in the graph. However, this function is used by *sepia* extensions based on libusb, resulting in reduced CPU usage.

7.2. Complex Pipelines

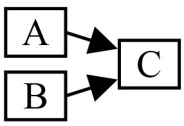
The present framework is designed to implement feed-forward pipelines, with optional splits. Most partial event handlers can be represented with populations of neurons, as they perform small calculations with each input. Thus, event-based pipelines can be translated to neuromorphic hardware, though a method to actually perform the conversion has yet to be devised.

However, not all neural networks can be represented with event handlers. Notably, neurons with second order dynamics and synapses with delays dispatch events that are not an immediate response to an input spike. The present framework, and more generally, purely event-based algorithms—cannot implement such models. To use complex neurons to process the output of a camera, one needs to leverage frameworks designed to implement neural networks. The present framework can, in this case, be used to communicate with sensors, perform low-level processing and send events to the neural network.

Nevertheless, two types of architectures more complex than feed-forward pipelines can be implemented in our framework: streams merging and feedback loops. Even though they still

impose more constraints than generic spiking neural networks, they allow for the efficient implementation of algorithms on a CPU without the need for another framework.

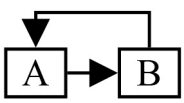
Streams merging has the following generic structure:



A, **B** are partial event handlers, and **C** is a complete event handler. This structure appears when merging the results of several calculations with a common origin. For example, one may split a stream of polarity events to compute two optical flows (one per polarity) and merge them to calculate an overall flow. **A** and **B** run sequentially in this scenario, therefore events are dispatched to **C** in the order of their timestamps. This scenario can be implemented by constructing **C** before the pipeline. The partial event handlers **A** and **B** are both given a reference to **C** as complete event handler. The `std::reference_wrapper` class can be used to prevent template deduction to a non-reference type, which would trigger a copy.

The merge operation can also arise from the use of multiple sensors, for example for stereo-vision or audio-video fusion. In this case **A** and **B** run in parallel, on different threads. Given the non-deterministic scheduling of most operating systems, **C** must re-order the events dispatched by its observables before handling them. This operation is implemented by the partial event handler `tarsier::merge`, compatible with an arbitrary number of observables.

A simple feedback loop can be modeled as:



A and **B** are both partial event handlers. This structure can be useful for flow control or learning. The feedback operation can be executed at various moments in the lifecycle of an algorithm: after processing a batch of data, immediately after each event and after each event with a delay. The implementation of the second and third approaches is not straightforward with existing packet-based frameworks. The whole packet has already been processed by **A** when the first event is processed by **B**, preventing the associated feedback from affecting the next events. The second approach can be implemented in *tarsier* using variables shared between **A** and **B**. Before handling an event, **A** reads from the variables and processes the event accordingly. After handling an event, **B** writes to the variables. Since an event is completely handled before the next is considered, modifications of the shared variables caused by the event n will be available to event $n+1$. The third approach—adding delays to the feedback—can be implemented by combining the second approach and a merge structure.

7.3. Parallelism

The application illustrated in **Figure 13** relies on multiple threads, and can take advantage of CPUs with a few cores. However, the sequential strategy presented so far does not harness the full potential of many-cores architectures.

The creation of parallel tasks and inter-task communication have a cost. An application using multiple tasks must reach a compromise on grain size (Acar et al., 2018). A large grain size yields less overhead, whereas a small grain size fully utilizes the CPU capabilities. The atomic tasks of an event-based pipeline are its partial event handlers. Larger grain sizes can be obtained by combining several partial handlers into a single task. The tasks represented in **Figure 6** can be combined either vertically (one thread per event) or horizontally. The former requires inter-thread communication with every partial handler to ensure sequentiality, canceling the benefits of parallelism, whereas the latter corresponds to the buffer-based approach of *event-driven YARP* and *Dynamic Vision System*. Consequently, latency increases with the grain size.

Parallelism can be beneficial when high latency is not critical and a high throughput is required. However, implementing parallelism efficiently is not straightforward: to avoid FIFO overflows between modules, possibly complex flow control algorithms must be implemented. High-quality libraries provide high-level tools to build parallel algorithms, such as Intel Threading Building Block's flow graph (Tovinkere and Voss, 2014). The partial event handlers provided by *tarsier* can be integrated with such tools. Thus, one can implement an algorithm once and use it with either a low-latency *tarsier* pipeline or a high-throughput flow graph. An example integration of a partial event handler in a class manipulating buffers is given in the *tutorials* repository. This approach can also help integrating *tarsier* with other event-based frameworks, in order to use existing drivers and viewers.

8. CONCLUSION AND DISCUSSION

We have presented a modular framework for event-based computer vision with three major components: *sepia*, *tarsier*, and *chameleon*. The components, though designed to work together, have no explicit relationship, thus minimizing the external dependencies of each component. Moreover, each component can easily be replaced with other libraries.

The presented framework hides buffers from the user, serving our goal: encouraging functional, event-based semantics likely to translate to neuromorphic hardware while providing an efficient implementation on CPUs. Benchmarks show an increased throughput and a reduced latency compared to state-of-the-art frameworks for event-based computer vision. Using contiguous memory to store events is crucial to performance. Moreover, assembling pipelines before compilation reduces latency and improves throughput, thanks to better compiler optimizations and fewer memory operations. The common practice of hard-coding simple operations (mirroring the stream, removing noise…) in file readers to reduce latency is no longer required with static polymorphism, yielding a cleaner, more generic codebase.

The benchmarks compare performance with pipelines of varying complexity. However, all the considered experiments use simple pipelines (without merges or loops), focus solely on the algorithm performance (the performance of IO and display operations is not evaluated), and run in real-time on the test machine. In a future work, we plan to devise new benchmarks to cover more use-cases. Moreover, adding more measurements, such as power consumption, will enable comparisons with neuromorphic hardware.

Assembling a pipeline before compiling requires meta-programming, i.e., another programming language to generate the actual code. The framework presented in this work uses C++ template meta-programming, since this language is supported by every standard-compliant compiler. Nevertheless, it can be unsettling to new users, and makes the creation of wrappers in high-level languages, such as Python, difficult. A high-level language or graphical user interface must bundle a C++ compiler to generate *tarsier* pipelines. Nevertheless, the framework modular structure and its independence from third-party libraries make it a good candidate for a common low-level library to multiple high-level interfaces. It can notably be integrated with native Android applications, or used to speed up Python modules.

The observer pattern used by the framework naturally models event-based cameras and algorithms. However, this pattern can lead to the problem known as callback hell: deeply nested statements make the code hard to read. Languages, such as Javascript have solved this problem with the async/await construct. This construct is available in C++, but is not compatible with the template deduction mechanism leveraged by the framework.

The current implementation of *partial event handlers* relies on *make* functions. These functions wrap the handlers constructors to enable template deduction. The C++17 standard allows template deduction from the constructor of a class, making the *make* functions unnecessary. The upcoming Debian 10 and macOS 10.15 operating systems will provide full support for this standard with their default libraries, allowing a major framework update.

DATA AVAILABILITY STATEMENT

The code repositories mentioned in this study:

- framework tutorials https://github.com/neuromorphic-paris/tutorials
- frameworks benchmarks https://github.com/neuromorphic-paris/frameworks_benchmarks
- *sepia* repository https://github.com/neuromorphic-paris/sepia
- *tarsier* repository https://github.com/neuromorphic-paris/tarsier
- *chameleon* repository https://github.com/neuromorphic-paris/chameleon
- *Event Stream* specification https://github.com/neuromorphic-paris/event_stream.

AUTHOR CONTRIBUTIONS

AM, S-HI, and RB contributed the conception and design of the study. AM devised the theoretical model, implemented it, carried out the experiments, analyzed the results, and wrote the first draft of the manuscript. All authors contributed to the manuscript revision, read, and approved the submitted version.

FUNDING

This work received the support from the French Medical Research Foundation via the program of Bioinformatics Analysis in Biology Research [DBI20141231328]. This work also received the support from LABEX LIFESENSES [ANR-10-LABX-65], managed by the French state funds (ANR) within the Investissements d'Avenir program [ANR-11-IDEX-0004-02].

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2019.01338/full#supplementary-material

REFERENCES

Acar, U. A., Aksenov, V., Charguéraud, A., and Rainey, M. (2018). "Performance challenges in modular parallel programs," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming—PPoPP '18* (Vienna).

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Nolfo, C. D., et al. (2017). "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Honolulu, HI).

Appleby, A. (2014). *Smhasher*. Available online at: https://github.com/aappleby/smhasher

Bainomugisha, E., Carreton, A. L., Cutsem, T. V., Mostinckx, S., and Meuter, W. D. (2013). A survey on reactive programming. *ACM Comput. Surveys* 45, 1–34. doi: 10.1145/2501654.2501666

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

Barrett, S. (2014). *STB Github Page*. Available online at: https://github.com/nothings/stb

Benosman, R., Ieng, S.-H., Clercq, C., Bartolozzi, C., and Srinivasan, M. (2012). Asynchronous frameless event-based optical flow. *Neural Netw.* 27, 32–37. doi: 10.1016/j.neunet.2011.11.001

Berner, R., and Delbruck, T. (2011). Event-based pixel sensitive to changes of color and brightness. *IEEE Trans. Circuits Syst. I Reg. Papers* 58, 1581–1590. doi: 10.1109/tcsi.2011.2157770

Blum, H., Dietmüller, A., Milde, M., Conradt, J., Indiveri, G., and Sandamirskaya, Y. (2017). A neuromorphic controller for a robotic vehicle equipped with a dynamic vision sensor. *Robot. Sci. Syst. XIII*. doi: 10.15607/RSS.2017.XIII.035

Bradski, G. (2000). The OpenCV Library. *Dobbs J. Softw. Tools* 120, 122–125.

Brandli, C., Berner, R., Yang, M., Liu, S.-C., and Delbruck, T. (2014). A 240 × 180 130 db 3μs latency global shutter spatiotemporal vision sensor. *IEEE J. Solid State Circuits* 49, 2333–2341. doi: 10.1109/jssc.2014.2342715

Caer. (2007). Available online at: https://gitlab.com/inivation/dv-runtime/tags/caer-1.1.2

Carmona-Galán, R., Zarándy, A., Rekeczky, C., Földesy, P., Rodríguez-Pérez, A., Domínguez-Matas, C., et al. (2013). A hierarchical vision processing architecture oriented to 3d integration of smart camera chips. *J. Syst. Archit.* 59, 908–919. doi: 10.1016/j.sysarc.2013.03.002

Chen, D. G., Matolin, D., Bermak, A., and Posch, C. (2011). Pulse-modulation imaging—review and performance analysis. *IEEE Trans. Biomed. Circuits Syst.* 5, 64–82. doi: 10.1109/TBCAS.2010.2075929

Cohen, G. K. (2016). *Event-based feature detection, recognition and classification* (Ph.D. thesis), Université Pierre et Marie Curie-Paris VI, Western Sydney University.

Cookie Clicker. (2013). Available online at: http://orteil.dashnet.org/cookieclicker

Delbruck, T., and Berner, R. (2010). "Temporal contrast aer pixel with 0.3%-contrast event threshold," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris).

Delbruck, T., Linares-Barranco, B., Culurciello, E., and Posch, C. (2010). "Activity-driven, event-based vision sensors," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris).

Dudek, P., and Hicks, P. (2005). A general-purpose processor-per-pixel analog simd vision chip. *IEEE Trans. Circuits Syst. I Reg. Papers* 52, 13–20. doi: 10.1109/tcsi.2004.840093

DV (2019). *Dynamic Vision System*. Available online at: https://gitlab.com/inivation/dv-runtime

Ferg, S. (2006). *Event-Driven Programming: Introduction, Tutorial, History*. Available online at: http://eventdrivenpgm.sourceforge.net

Furber, S. (2017). Microprocessors: the engines of the digital age. *Proc. R. Soc. A Math. Phys. Eng. Sci.* 473:20160893. doi: 10.1098/rspa.2016.0893

Galluppi, F., Brohan, K., Davidson, S., Serrano-Gotarredona, T., Carrasco, J.-A. P., Linares-Barranco, B., et al. (2012). "A real-time, event-driven neuromorphic system for goal-directed attentional selection," in *Lecture Notes in Computer Science* (Doha), 226–233.

Glover, A., Vasco, V., and Bartolozzi, C. (2018a). "A controlled-delay event camera framework for on-line robotics," in *2018 IEEE International Conference on Robotics and Automation (ICRA)* (Brisbane, QLD).

Glover, A., Vasco, V., Iacono, M., and Bartolozzi, C. (2018b). The event-driven software library for yarp—with algorithms and icub applications. *Front. Robot. AI* 4:73. doi: 10.3389/frobt.2017.00073

Haessig, G., Cassidy, A., Alvarez, R., Benosman, R., and Orchard, G. (2018). Spiking optical flow for event-based sensors using IBM's truenorth neurosynaptic system. *IEEE Trans. Biomed. Circuits Syst.* 12, 860–870. doi: 10.1109/TBCAS.2018.2834558

Hopkins, M., Pineda-García, G., Bogdan, P. A., and Furber, S. B. (2018). Spiking neural networks for computer vision. *Interface Focus* 8:20180007. doi: 10.1098/rsfs.2018.0007

Huang, J., Guo, M., and Chen, S. (2017). "A dynamic vision sensor with direct logarithmic output and full-frame picture-on-demand," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (Baltimore, MD).

Indiveri, G., and Liu, S.-C. (2015). Memory and information processing in neuromorphic systems. *Proc. IEEE* 103, 1379–1397. doi: 10.1109/jproc.2015.2444094

Jaer. (2007). Available online at: http://jaerproject.org

Jones, E., Oliphant, T., and Peterson, P. (2001). *SciPy: Open Source Scientific Tools for Python*. Available online at: http://www.scipy.org/

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

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2017). Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.* 5, 308–323. doi: 10.1145/355841.355847

Lenero-Bardallo, J. A., Serrano-Gotarredona, T., and Linares-Barranco, B. (2011). A 3.6 μs latency asynchronous frame-free event-driven dynamic-vision-sensor. *IEEE J. Solid State Circuits* 46, 1443–1455. doi: 10.1109/jssc.2011.2118490

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

Liu, S.-C., and Delbruck, T. (2010). Neuromorphic sensory systems. *Curr. Opin. Neurobiol.* 20, 288–295. doi: 10.1016/j.conb.2010.03.007

Maqueda, A. I., Loquercio, A., Gallego, G., Garcia, N., and Scaramuzza, D. (2018). "Event-based vision meets deep learning on steering prediction for self-driving cars," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT).

Moeys, D. P., Corradi, F., Kerr, E., Vance, P., Das, G., Neil, D., et al. (2016). "Steering a predator robot using a mixed frame/event-driven convolutional neural network," in *2016 Second International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)* (Krakow).

Mueggler, E., Bartolozzi, C., and Scaramuzza, D. (2017). "Fast event-based corner detection," in *Procedings of the British Machine Vision Conference 2017* (London).

Orchard, G., Lagorce, X., Posch, C., Furber, S. B., Benosman, R., and Galluppi, F. (2015). "Real-time event-driven spiking neural network object recognition on the spinnaker platform," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)* (Lisbon).

Posch, C., Matolin, D., and Wohlgenannt, R. (2010). "A QVGA 143db dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression," in *2010 IEEE International Solid-State Circuits Conference-(ISSCC)* (San Francisco, CA).

Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B., and Delbruck, T. (2014). Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. *Proc. IEEE* 102, 1470–1484. doi: 10.1109/jproc.2014.2346153

Qt. (1995). Available online at: https://www.qt.io

Reverter Valeiras, D., Kime, S., Ieng, S.-H., and Benosman, R. B. (2016). An event-based solution to the perspective-n-point problem. *Front. Neurosci.* 10:208. doi: 10.3389/fnins.2016.00208

Serrano-Gotarredona, T., and Linares-Barranco, B. (2013). A 128 ×128 1.5% contrast sensitivity 0.9% fpn 3 μs latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *IEEE J. Solid State Circuits* 48, 827–838. doi: 10.1109/jssc.2012.2230553

Son, B., Suh, Y., Kim, S., Jung, H., Kim, J.-S., Shin, C., et al. (2017). "4.1 a 640 × 480 dynamic vision sensor with a 9μm pixel and 300 meps address-event representation," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)* (San Francisco, CA).

Tedaldi, D., Gallego, G., Mueggler, E., and Scaramuzza, D. (2016). "Feature detection and tracking with the dynamic and active-pixel vision sensor (DAVIS)," in *2016 Second International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)* (Krakow).

Thompson, C. M., and Shure, L. (1995). *Image Processing Toolbox [For Use With Matlab]*. Avialable online at: http://infoscience.epfl.ch/record/24814

Tilkov, S., and Vinoski, S. (2010). Node.js: using javascript to build high-performance network programs. *IEEE Intern. Comput.* 14, 80–83. doi: 10.1109/mic.2010.145

Tovinkere, V., and Voss, M. (2014). "Flow graph designer: a tool for designing and analyzing intel threading building blocks flow graphs," in *2014 43rd International Conference on Parallel Processing Workshops* (Minneapolis, MN).

Veldhuizen, T. (2000). Techniques for scientific c++. *Comput. Sci. Tech. Rep.* 542:60.

Yang, M., Liu, S.-C., and Delbruck, T. (2015). A dynamic vision sensor with 1% temporal contrast sensitivity and in-pixel asynchronous delta modulator for event encoding. *IEEE J. Solid State Circuits* 50, 2149–2160. doi: 10.1109/jssc.2015.2425886

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 Neuroscience

ORIGINAL RESEARCH
published: 20 February 2020
doi: 10.3389/fnins.2020.00135

Low-Power Dynamic Object Detection and Classification With Freely Moving Event Cameras

Bharath Ramesh[1,2]*, *Andrés Ussa*[1,2], *Luca Della Vedova*[2], *Hong Yang*[2] and *Garrick Orchard*[1,2]

[1] Life Science Institute, The N.1 Institute for Health, National University of Singapore, Singapore, Singapore, [2] Temasek Laboratories, National University of Singapore, Singapore, Singapore

We present the first purely event-based, energy-efficient approach for dynamic object detection and categorization with a freely moving event camera. Compared to traditional cameras, event-based object recognition systems are considerably behind in terms of accuracy and algorithmic maturity. To this end, this paper presents an event-based feature extraction method devised by accumulating local activity across the image frame and then applying principal component analysis (PCA) to the normalized neighborhood region. Subsequently, we propose a backtracking-free *k*-d tree mechanism for efficient feature matching by taking advantage of the low-dimensionality of the feature representation. Additionally, the proposed *k*-d tree mechanism allows for feature selection to obtain a lower-dimensional object representation when hardware resources are limited to implement PCA. Consequently, the proposed system can be realized on a field-programmable gate array (FPGA) device leading to high performance over resource ratio. The proposed system is tested on real-world event-based datasets for object categorization, showing superior classification performance compared to state-of-the-art algorithms. Additionally, we verified the real-time FPGA performance of the proposed object detection method, trained with limited data as opposed to deep learning methods, under a closed-loop aerial vehicle flight mode. We also compare the proposed object categorization framework to pre-trained convolutional neural networks using transfer learning and highlight the drawbacks of using frame-based sensors under dynamic camera motion. Finally, we provide critical insights about the feature extraction method and the classification parameters on the system performance, which aids in understanding the framework to suit various low-power (less than a few watts) application scenarios.

Keywords: object recognition, neuromorphic vision, low-power FPGA, closed-loop control, object detection, event-based descriptor, rectangular grid, FIFO processing

OPEN ACCESS

Edited by:
Chiara Bartolozzi,
Italian Institute of Technology, Italy

Reviewed by:
Federico Corradi,
Imec, Netherlands
Arren Glover,
Italian Institute of Technology, Italy

***Correspondence:**
Bharath Ramesh
bharath.ramesh03@u.nus.edu

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *16 October 2019*
Accepted: *03 February 2020*
Published: *20 February 2020*

Citation:
Ramesh B, Ussa A, Della Vedova L, Yang H and Orchard G (2020) Low-Power Dynamic Object Detection and Classification With Freely Moving Event Cameras. Front. Neurosci. 14:135. doi: 10.3389/fnins.2020.00135

1. INTRODUCTION

Through these fruitful decades of computer vision research, we have taken huge strides in solving specific object recognition tasks, such as classification systems for automated assembly line inspection, hand-written character recognition in mail sorting machines, bill inspection in automated teller machines, to name a few. Despite these successful applications, generalizing object

appearance, even under moderately controlled sensing environments, for robust and practical solutions for industrial challenges like robot navigation and sense-making is a major challenge. This paper focuses on the industrially relevant problem of real-time, low-power object detection using an asynchronous event-based camera (Brandli et al., 2014) with limited training data under unconstrained lighting conditions. Compared to traditional frame-based cameras, event cameras do not have a global shutter or a clock that determines its output. Instead, each pixel responds independently to temporal changes with a latency ranging from a low of tens of microseconds to a high of few milliseconds. This local sensing paradigm naturally results in a wider dynamic range (120 dB), as opposed to the usual 60 dB for frame-based cameras.

Most significantly, event cameras do not output pixel intensities, but only a spike output with a precise timestamp, also termed an event, that signifies a sufficient change in log-intensity of the pixel. As a result, event cameras require lower transmission bandwidth and consume only a few hundred mW vs. a few W by standard cameras (Posch et al., 2014). In summary, event-based cameras offer a fundamentally different perspective to visual imaging while having a strong emphasis on low-latency and low-power algorithms (Conradt et al., 2009; Ni et al., 2012; Delbruck and Lang, 2013; Kueng et al., 2016).

Despite the notable advantages of event cameras, there still remains a significant performance gap between event camera algorithms and frame-based counterparts for various vision problems. This is partly due to a requirement of totally new event-by-event processing paradigms. However, the burgeoning interest in event-based classification/detection is focused on closing the gap using deep spiking neural networks (O'Connor et al., 2013; Lee et al., 2016), something that again entails dependence on powerful hardware like its frame-based counterpart. On the other hand, a succession of frames captured at a constant rate (say 30 Hz), regardless of the scene dynamics and ego-motion, works well with controlled scene condition and camera motion. Frame-based computer vision algorithms have benefited immensely from sophisticated methodologies that reduce the computational burden by selecting and processing only informative regions/keypoints within an image (Lowe, 2004; Galleguillos et al., 2008; Vikram et al., 2012; Ramesh et al., 2017a). In addition, frame-based sensing has led to high hardware complexity, such as powerful GPU requirements for efficiently re-training and deploying state-of-the-art object detection frameworks using deep neural networks (Ren et al., 2017; Redmon and Farhadi, 2018).

Since event-based vision is relatively new, only a limited amount of work addresses object detection using these devices (Liu et al., 2016; Iacono et al., 2018; Lenz et al., 2018). Liu et al. (2016) focuses on combining a frame-based CNN detector to facilitate the event-based module. We argue that using intensity images, either reconstructed from the event stream (Scheerlinck et al., 2018) or captured simultaneously (Liu et al., 2016; Iacono et al., 2018), with deep neural networks for event-based object detection may achieve good performance with lots of training data and computing power, but they go against the idea of low-latency, low-power event-based vision. In contrast, Lenz et al. (2018) presents a practical event-based approach to face detection by looking for pairs of blinking eyes. While Lenz et al. (2018) is applicable to human faces in the presence of activity, we develop a general purpose event-based, object detection method using a simple feature representation based on local event aggregation. Thus, this paper is similar in spirit to the recently spawned ideas of generating event-based descriptors, such as histogram of averaged time surfaces (Sironi et al., 2018) and log-polar grids (Ramesh et al., 2017b, 2019b), or low-level corner detectors as features (Manderscheid et al., 2019). Moreover, the proposed object detection and categorization method was accommodated on FPGA to demonstrate energy-efficient low-power vision.

In contrast to the above works, this paper introduces a simple, energy-efficient approach for object detection and categorization. **Figure 1** illustrates the local event-based feature extraction pipeline that is used for classification using a codebook-based method. Accordingly, efficient feature matching with the codebook is required, which is handled by a backtracking-free branch-and-bound k-d tree. This proposed system was ported to a field programmable gate array (FPGA) with certain critical design decisions, one of which demanded a virtual dimensionality reduction method based on the k-d tree, to accommodate very low-power computational needs.

This paper is an extended version of the work initially published in ACCV Workshops 2018 (Ramesh et al., 2019a). Novel contributions over (Ramesh et al., 2019a) include closed-loop aerial flight, tested for the first time using event-based sensors to the best of our knowledge, and robustness analysis using hand-held experiments (section 3.3) with critical insights into the system performance for various hyper-parameters (section 3.1.1). Additionally, this work includes a comprehensive comparison to deep learning methods (section 3.4) and further provides full implementation details in section 2, including a free-running mode implementation capable of a classification output at any point in time, as opposed to a classifier periodically operating on a set of events like (Ramesh et al., 2019a).

2. MATERIALS AND METHODS

We follow the event-based classification framework proposed in Ramesh et al. (2017b), with the following crucial changes: a new descriptor (PCA-RECT), a virtual dimensionality reduction technique using k-d trees (vPCA) and a simplified feature matching mechanism to account for hardware limitations. The framework consists of four main stages: feature extraction, feature matching with a codebook, creating an object representation, which is lastly fed to a linear classifier. Additionally, we incorporate an object detector in the framework as explained in the following subsections.

2.1. PCA-RECT

Each incoming event, $\mathbf{e}_i = (x_i, y_i, t_i, p_i)^T$ with pixel location x_i and y_i, timestamp t_i, polarity p_i, is encoded as a feature vector $\mathbf{x}_i$. To deal with hardware-level noise from the event camera, two main steps are used: (1) nearest neighbor filtering and

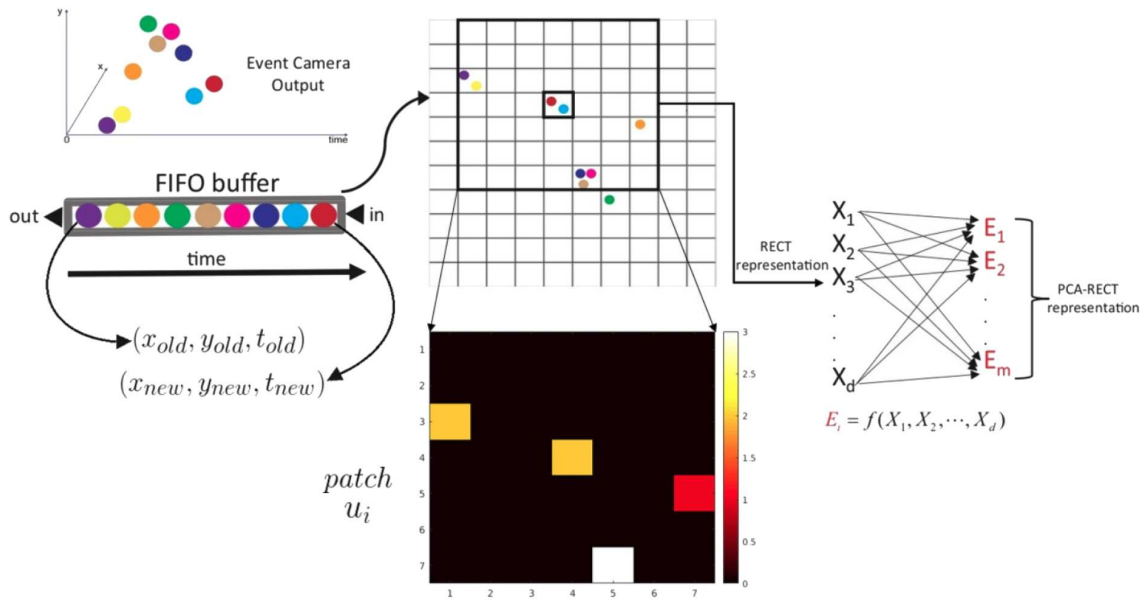


FIGURE 1 | PCA-RECT representation (best viewed on monitor). Useful events are sub-sampled and filtered after applying nearest-neighbor temporal filtering and refractory filtering, termed as rectangular event context transform (RECT). The sparser RECT event representation is updated dynamically using a first in, first out (FIFO) buffer. Subsequent feature extraction is carried out by applying principal component analysis (PCA) to project RECT onto a lower-dimensional subspace to obtain the final PCA-RECT feature representation.

(2) refractory filtering. We define a spatial Euclidean distance between events as,

$$D_{i,j} = \left|\left|\begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_j \\ y_j \end{pmatrix}\right|\right| . \tag{1}$$

Using the above distance measure, for any event we can define a set of previous events within a spatial neighborhood, $N(\mathbf{e}_i, \gamma) = \{\mathbf{e}_i \mid j < i,\ D_{i,j} < \gamma\}$, where $\gamma = \sqrt{2}$ for an eight-connected pixel neighborhood. When the time difference between the current event and the most recent neighboring event is less than a threshold, Θ_{noise}, the filter can be written as

$$F_{noise}(\mathbf{e}) = \{\mathbf{e}_i |\ N(\mathbf{e}_i,\ \sqrt{2}) \backslash N(\mathbf{e}_i,\ 0)\ \ni\ \mathbf{e}_j \mid t_i - t_j < \Theta_{noise}\} . \tag{2}$$

When the neighborhood is only the current pixel, $\gamma = 0$, the set of events getting through the refractory filter F_{ref} are those such that,

$$F_{ref}(\mathbf{e}) = \{\mathbf{e}_j |\ \ t_i - t_j > \Theta_{ref}\ \forall\, j \mid \mathbf{e}_j \in N(\mathbf{e}_j,\ 0)\} . \tag{3}$$

Cascading the filters, we can write the filtered incoming events as,

$$\{\hat{\mathbf{e}}\} = F_{noise}\left(F_{ref}(\mathbf{e})\right) . \tag{4}$$

As shown in **Figure 1**, the incoming events $\hat{\mathbf{e}}_i$ are first pushed into a FIFO buffer. The FIFO queue is then used to update an event-count matrix $C \in \mathbb{R}^{m \times n}$, where m and n denote the number of rows and columns of the event camera output.

$$C(x_i, y_i) = C(x_i, y_i) + 1 . \tag{5}$$

Before pushing the latest event, the FIFO buffer of size $\mathcal{S}$ is popped to make space and simultaneously update the count matrix C,

$$C(x_{i-s}, y_{i-s}) = C(x_{i-s}, y_{i-s}) - 1 . \tag{6}$$

The event-count C is pooled to build local representations, which are further aggregated to obtain the RECT representation of each event. In particular, let A be a $p \times q$ rectangular grid filter, the 2-D convolution is defined as,

$$R(j,k) = \sum_p \sum_q A(p,q) C(j-p+1, k-q+1) , \tag{7}$$

where p run over all values that lead to legal subscripts of $A(p,q)$ and $C(j-p+1, k-q+1)$. In this work, we consider a filter containing equal weights (commonly known as an averaging filter) for simplicity, while it is worth exploring Gaussian-type filters that can suppress noisy events. The resultant 2-D representation is termed as filtered matrix $R \in \mathbb{R}^{(m/p)\times(n/p)}$, where the filter dimensions are chosen to be give integer values for m/p and n/q or conversely C is zero-padded sufficiently. Subsequently, the RECT representation for $\hat{\mathbf{e}}_i$ is obtained as a patch $\mathbf{u}_i$ (see **Figure 1**) of dimension d centered at $R(y/p, x/q)$. Subsequently, the filtered event-count patch is projected on-to a lower-dimensional subspace using principal component analysis (PCA) for eliminating noisy dimensions and improving classifier accuracy. For the sake of completion, the details of extracting the principal components (PCs) are given below.

For a set of n mean-centered feature vectors $\mathbf{u}_i \in \mathbb{R}^d$ with N_i samples in the subset D_i belonging to class ω_i, $(i = 1, \cdots, C)$,

principal component analysis (PCA) seeks a projection W that minimizes the error function:

$$J_{PCA}(W) = \sum_{k=1}^{N} ||\mathbf{u}_k - \mathbf{v}_k||^2 . \quad (8)$$

where $\mathbf{v}_k$ is obtained after projection of $\mathbf{u}_k$ by W as $\mathbf{v}_k = WW^T\mathbf{u}_k$. The minimization is equivalent to finding the eigenvectors of the total scatter matrix defined as:

$$S_T = \sum_{k=1}^{N} (\mathbf{u}_k - \mu)(\mathbf{u}_k - \mu)^T . \quad (9)$$

where μ is the mean of all training samples:

$$\mu = \frac{1}{N} \sum_{k=1}^{N} \mathbf{u}_k . \quad (10)$$

The columns of W associated with non-trivial eigenvalues are the PCs and those with negligible eigenvalues are regarded as arising from noise. After projecting the RECT representation using the PCs, each filtered event is thus encoded as a feature vector $\mathbf{x}_i \in \mathbb{R}^{d'}$ where $d' < d$.

2.2. Feature Selection and Matching Using *K*-d Trees

The PCA-RECT feature representation for each event is matched to a codebook for creating the object representation. However, exhaustive search is too costly for nearest neighbor matching with a codebook, and approximate algorithms can be orders of magnitude faster than exact search, while almost achieving par accuracy.

In the vision community, k-d tree nearest-neighbor search is popular (Silpa-Anan and Hartley, 2008; Muja and Lowe, 2009), as a means of searching for feature vectors in a large training database. Given n feature vectors $\mathbf{x}_i \in \mathbb{R}^{d'}$, the k-d tree construction algorithm recursively partitions the d'-dimensional Euclidean space into hyper-rectangles along the dimension of maximum variance. However, for high dimensional data, backtracking through the tree to find the optimal solution still takes a lot of time.

Research in the vision community has therefore aimed at increasing the probability of success while keeping backtracking within reasonable limits. Two similar and successfully applied approximated search methods are the best-bin-first search (Beis and Lowe, 1997) and priority search (Arya and Mount, 1993). Backtracking is accomplished in such methods by maintaining an estimate of the distance from the query point to any of the nodes down all of the branches. In the best-bin-first search, a parameter specifies the number of data points that can be checked before terminating and returning the closest point traversed up to that point. This process however still requires the computationally expensive Euclidean distance calculation to a subset of the data points in the codebook or training database.

This paper proposes a simple, backtracking-free branch-and-bound search for matching (Algorithm 1), taking advantage of the low-dimensionality of the PCA-RECT representation. The hypothesis is that, in general, the point recovered from the leaf node is a good approximation to the nearest neighbor in low-dimensional spaces, and performance degrades rapidly with increase in dimensionality, as inferred from the intermediate results in Beis and Lowe (1997). In other words, with $(\log_2 n) - 1$ scalar comparisons, nearest neighbor matching is accomplished without an explicit distance calculation. While the PCA-RECT representation is useful for software implementations, an extra PCA projection step can be computationally demanding on FPGA devices. To this end, we propose a virtual PCA-RECT representation based on the k-d tree, termed as vPCA-RECT.

2.2.1. vPCA-RECT

A key insight is that only a fraction of the data dimensions are used to partition the k-d tree, especially when the codebook size is only a few times more than the feature dimension. Therefore, instead of using the PCA-RECT representation, an alternative dimensionality reduction scheme can be implemented by discarding the unused dimensions in the k-d tree structure. In other words, the RECT representation is first used to build a k-d tree that selects the important dimensions (projection π), which are then utilized for codebook learning and classification. It is worth noting that exactly the same k-d tree will be obtained if the RECT data is first projected by π onto a subspace that is aligned with the coordinate axes. Since no actual projection takes place, we refer to this as a virtual projection—the irrelevant dimensions chosen by the k-d tree are discarded to obtain a lower-dimensional feature representation.

2.3. Event-Based Object Categorization and Detection

The learning stage: Using the PCA-RECT event representation, the learning process corresponds to creating a set of K features denoted as $M = \{1, 2, \cdots, K\}$ to form the codebook (also termed

Algorithm 1: KDSEARCHFAST(node, **y**): Fast k-d tree search for PCA-RECT

Input: The root of the *k*dtree structure, and a query point **y**
Output: *k*dtree leaf node data-point j

```
if node.type ≠ leaf then
    if y(node.splitDimension) ≤ node.splitThreshold then
        node ← node.left
        KDSEARCHFAST(node, y)
    else
        node ← node.right
        KDSEARCHFAST(node, y)
    end if
else
    return node.dataIndex
end if
```

as dictionary). The clustering is done by randomly selecting a sufficiently large pool of event representations of various categories from the training samples. It is worth noting that the dictionary features are learned from the training set using unsupervised clustering for all the objects jointly.

In contrast to the above single-shot learning for PCA, the vPCA-RECT representation requires two codebook learning steps. First, using the RECT representation, an initial codebook of size K_v, where $K_v << K$, is built to spot the unused feature dimensions in the k-d tree. The unused dimensions simply correspond to those which were not used by the k-d tree construction algorithm to recursively partition the d'-dimensional Euclidean space into hyper-rectangles. Subsequently, the projection π is used to obtain a lower-dimensional representation for the training data and then the final codebook of size K is generated. The initial, smaller codebook helps to partition the RECT feature space with much higher entropy, and thus is an essential step for the virtual PCA-RECT representation.

The learning stage for detection builds on top of the categorization module, in such a way that the learning process corresponds to selecting a subset of features from the codebook for each object. In contrast to the learning phase of the categorization module, the detector features are selected from the whole training set in a supervised one-vs-all manner.

We propose to evaluate the balanced matches Y_+^k to each codeword f_k from the target events against the matches Y_-^k for all the other events to the respective feature. Mathematically, the ratio

$$D(k) = \frac{\beta_+^k Y_+^k}{\beta_-^k Y_-^k}, \text{ where } \beta_+^k = \frac{|Y_+^k|}{\sum_{k=1}^{K} |Y_+^k|}, \text{ and } \beta_-^k = \frac{|Y_-^k|}{\sum_{k=1}^{K} |Y_-^k|}, \tag{11}$$

is to be maximized. The balancing component β_+^k denotes the percentage of target events matched to the codeword f_k. Similarly, β_-^k denotes the percentage of non-target events matched to the codeword f_k. Thus, choosing the detector features with the D–largest ratios completes the learning phase.

The classification/detection stage: At runtime, the event representations are propagated through the k-d tree. On the one hand, the distribution of the codewords are then extracted and further passed to a simple linear classifier (we experimented with both linear SVM and Kernel Methods). On the other hand, the event representations propagated through the k-d tree are matched with the detector features. Those matched events are used to update a location map for the target object and the region with the highest activation is considered to be the final detection result.

2.4. FPGA Implementation

An overview of the whole system is shown in **Figure 2**. Learning is performed in software first and the relevant data is transferred to the FPGA, which corresponds to the SVM weights and the information regarding the nodes of the k-d tree. This process does not require fine tuning of the FPGA code, except for the case of a different codebook size, in which only the new size has to be updated in the FPGA side. The memory initialization for the block memories is automated from the files generated during training in software.

2.4.1. Categorization Pipeline

In order to showcase energy-efficient event-based object recognition, the FPGA implementation of the algorithm is designed as a series of four independent hardware units: event sub-sampling, vPCA-RECT generation, a recursive k-d tree and a SVM classifier output on an event-by-event basis, each of which has an independent block design. Generally, these hardware counterparts are not a direct application of the algorithm

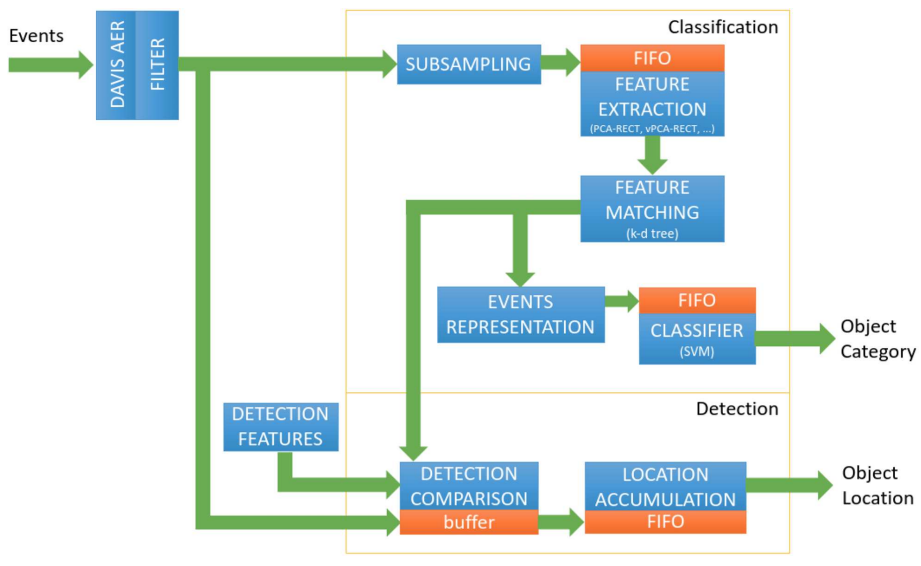


FIGURE 2 | Pipeline of the proposed system.

presented in the earlier section, i.e., certain design decisions were taken for this task, among them, to desist the use of an extra PCA projection along the pipeline.

The sub-sampling block receives the filtered event locations as input values x and y, each 8-bit in size, which are used to update the zero-padded count matrix $C \in \mathbb{R}^{m \times n}$ (Equation 5, 6). The sub-sampling behavior can be achieved in hardware through a combinatorial module that performs the division by shifting the inputs by one bit, and subsequently adding p and q to that value to obtain the sub-sampled representation (Equation 7). This results in two 7-bit values which are then concatenated to output a single memory address (**Figure 3A**).

The next block uses the cell-count matrix $R \in \mathbb{R}^{(m/p)\times(n/q)}$, created by a block of distributed RAM of depth $((m/p) \times (n/q))$ and $log(s)$-bits width, corresponding to the FIFO buffer size s, initialized to zero for generating the vPCA-RECT representation. To generate a descriptor with respect to the last event received would add a considerable overhead, since each element of the descriptor would have to be read sequentially from the block RAM while being stored by the next module. Instead, the address corresponding to the center of the descriptor is provided, i.e., the input address of the count matrix is passed over to the k-d tree module. This allows to trigger the k-d tree in one clock cycle once the count matrix is updated and later read the descriptor values based on this single coordinate. However, a new issue arises, the count matrix then can not be modified while the k-d tree exploration is being performed. Hence a buffering element is added between the sub-sampling and count matrix modules that will only provide the next address once there is a valid output from the tree.

The k-d tree nodes are represented in a 49-bit number stored in a previously initialized single port ROM of depth equal to the number of nodes. This number is conformed by the elements of a node: type, left node, right node, index output, split value and

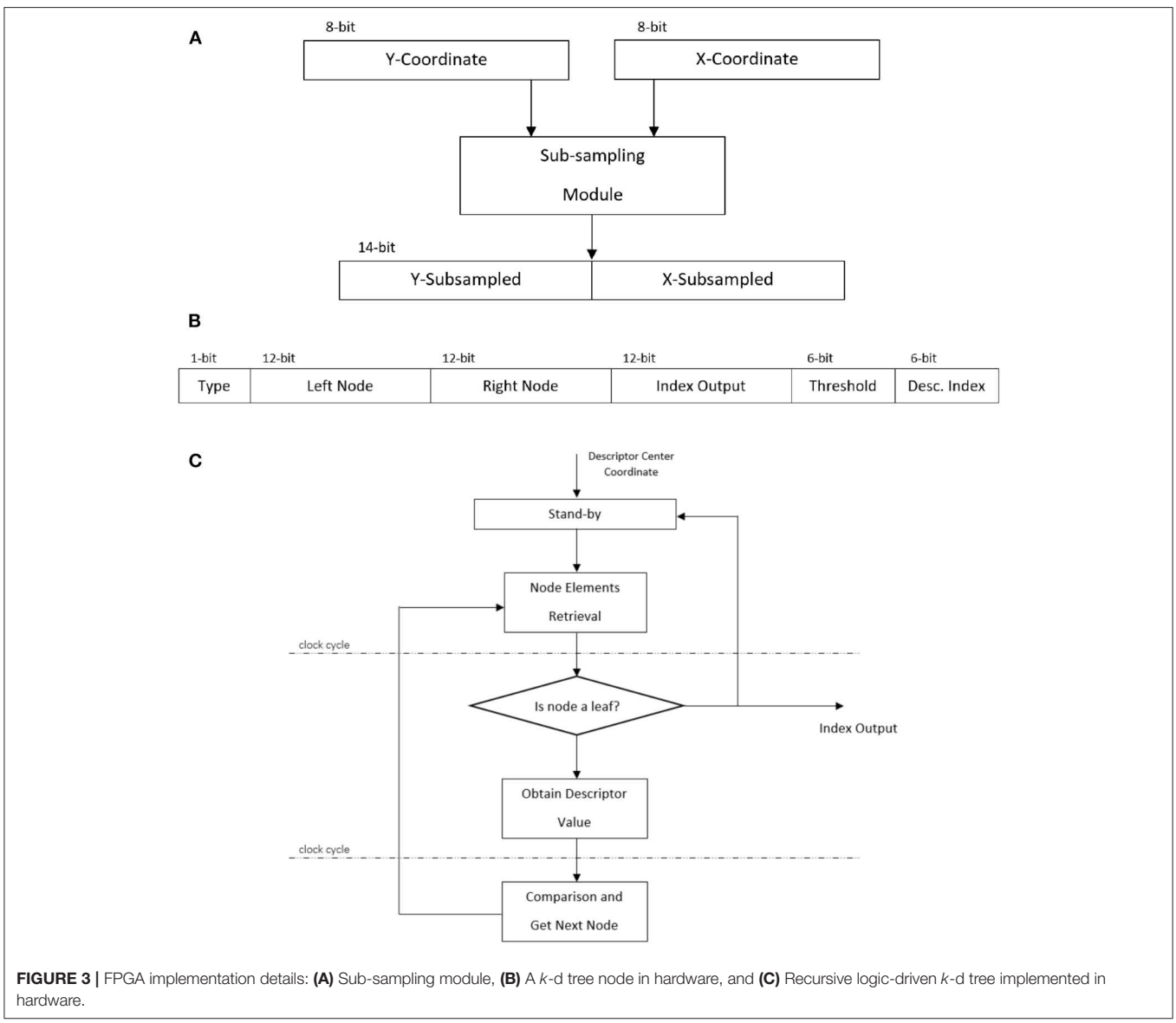


FIGURE 3 | FPGA implementation details: **(A)** Sub-sampling module, **(B)** A k-d tree node in hardware, and **(C)** Recursive logic-driven k-d tree implemented in hardware.

split dimension; these are concatenated and their width is shown in **Figure 3B**.

The k-d tree module follows a three steps cycle (**Figure 3C**). The split dimension of a k-d tree node provides the address that needs to be read from the cell-count matrix block RAM to get the relevant descriptor value. Next, the descriptor value is compared to the previously stored split value from the node, taking a path down the tree, left or right, depending on the boolean condition. The corresponding node to get is then retrieved from the respective left or right address element acquired in the retrieval step. This cycle repeats until the node type belongs to a leaf, then the leaf node output is made available for the classifier module. It is worth mentioning that in the software implementation of this algorithm, once the descriptor is formed, it is then normalized before being passed to the k-d tree. A normalization step in hardware would add a big overhead to the pipeline, disturbing its throughput, and it was removed from the FPGA implementation after verifying that the overall performance was not affected harshly. The "distribution of the codewords" normalization that is input to the SVM is an important step (Equation 12). It is implicitly performed by limiting the number of events quantized to that of the FIFO buffer size. This is more important than normalizing the descriptors. Using the N-SOD, we noticed only a small drop in test accuracy (from 98 to 93%) without normalization, which is acceptable for real-time applications.

At runtime in a software implementation, the classification is performed by a linear combination of the weights and a feature vector created by the k-d tree after a buffer time of S events. To achieve this in a hardware implementation, the depth of the feature vector would have to be transversed while performing several multiplications which would require a considerable amount of multiplier elements from the FPGA, and would affect the speed of the module. Thus, it was desired to avoid this solution and the following was proposed.

The elements of the linear combination mentioned would be acquired as readily available and would be added to an overall sum vector of length equal to the number of classes to classify, hence performing the dot product operation as one addition per event. Then, after S_c events, a resulting vector is formed, which is equal to the result of the same linear combination first mentioned in the software implementation. Thus, the final module to perform the classification receives the output index from the k-d tree and adds its corresponding classifier parameter to a sum vector of length equal to the number of classes. In parallel, this index value is stored in another FIFO element. When the queue is full, the oldest value would be passed to the module to be subtracted from the sum. This allows to have a classification output at any point in time, corresponding to the last S_c events.

Let k_i be the output of the k-d tree, which also corresponds to the codeword index of the dictionary, and k_{old} be the k-d tree index corresponding to the oldest event being removed from the FIFO buffer. Let the SVM weights and bias be $W_{SVM} \in \mathbb{R}^{K\times C}$ and $B_{SVM} \in \mathbb{R}^{1\times C}$, respectively for C object categories with K features. Thus, the former element k_i contributes to the SVM representation whereas k_{old} must be removed from the SVM representation. A classification output for an event i is computed as $S_i^o = W_{SVM} \cdot H_i + B_{SVM}$, considering a dictionary representation denoted by $H_i \in \mathbb{R}^{K\times 1}$. The equivalent free-running SVM update can be represented using Equation (12).

$$S_i^o = S_{i-1}^o + W_{SVM}\left(k_i, 1:C\right) - W_{SVM}\left(k_{old}, 1:C\right) \tag{12}$$

Note that the number of input events used to form the feature representation H_i is always constant, and corresponds to the last $\mathcal{S}$ events that creates the PCA-RECT representation using a FIFO (Equation 6). Hence, it is not a single event that is being classified. As the queue is updated on an event-by-event basis, the classification output corresponds to the entire block, although the classification output is triggered by every event.

2.4.2. Detection Pipeline

Parallel to the modules performing the classification pipeline, the aim of the detection process is to find the coordinates corresponding to "landmarks" with the highest activation after S_d events, and then find the most probable location for the object. Again, the algorithm was divided into multiple coherent hardware modules that would produce the same results as the original software version. The designed blocks are: landmarks detector, detection heat map and mean calculation.

First, the codewords corresponding to the landmarks that were calculated offline are loaded into a binary memory block. This module receives as input the codeword index provided by the k-d tree for the current event. If the feature is found as one of the landmarks, the respective event coordinates x and y are passed as a concatenated address to the next module in the pipeline. Next, a stage corresponding to the heat map is utilized. This module holds a matrix represented as a block RAM of depth $m \times n$, since the coordinates are not sub-sampled and have the ranges $1 \leq x \leq m$ and $1 \leq y \leq n$. For each new input address, its value in memory is incremented.

Since the aim of the detection algorithm is to calculate the average of the coordinates with the highest activation, it would be inefficient to find these event addresses after S_d events. Therefore, the coordinates with the highest count are stored in a FIFO element while the counting is performed. At the end, this will contain all the x and y coordinates needed for the average calculation. Once the classification flag is triggered, all the coordinates stored in the previous step (which belong to the highest activation) are acquired for calculating the total activation (the divisor). Subsequently, it will calculate the sum of the respective x and y values, and pass these as dividends to hardware dividers that will provide the final coordinates of the detected object. Algorithm 2 summarizes the above object detection hardware pipeline clearly.

2.5. Experiment Setup

We validated our approach on two benchmark datasets (Orchard et al., 2015a), namely the N-MNIST and N-Caltech101, that have become de-facto standards for testing event-based categorization algorithms. **Figure 4** shows some representative samples from N-MNIST and N-Caltech101.

The above datasets are good for only evaluating the categorization module. In addition, as the benchmark datasets

Algorithm 2: Event-based FPGA Object Detection

Input: Filtered event stream $\{\hat{\mathbf{e}}\}$, detector landmarks l, number of events S

Output: Mean object location (x_{obj}, y_{obj})

Initialize detector count $D(y, x) = 0_{m,n}$, detector cut-off $threshold = 0$
for $t = 1 : S$ **do**
  For each incoming event $\hat{\mathbf{e}}_t = (x_t, y_t, t_t, p_t, \mathbf{x}_t^T)^T$
  For $\mathbf{x}_t$ get leaf node index l_t using k-d tree
  if $l_t \in l$ **then**
    $D(y_t, x_t) = D(y_t, x_t) + 1$
    if $D(y_t, x_t) > threshold$ **then**
      $threshold = threshold + 1$
      Reset detector mean calculation FIFO
    end if
    if $D(y_t, x_t) = threshold$ **then**
      Push x_t, y_t into the mean calculation FIFO
    end if
  end if
end for
Output the mean of the coordinates in the FIFO as (x_{obj}, y_{obj})

were generated by displaying images on a monitor with limited and predefined motion of the camera, they do not generalize well to real-world situations. To overcome these limitations, we created a new dataset by directly recording objects in lab environment with a freely moving event-based sensor. The in-house dataset, called as Neuromorphic Single Object Dataset (N-SOD), contains three objects with samples of varying length in time (up to 20 s). The three objects to be recognized are a thumper 6-wheel ground robot, an unmanned aerial vehicle, a landing platform along with a background class (**Figure 5A**). The proposed object categorization and detection framework based on PCA-RECT is compared to state-of-the-art event-based works and thus software implementation is used with double numeric precision.

For real-time experiments, we use the commercial event camera, the Dynamic and Active-pixel Vision Sensor (DAVIS) (Brandli et al., 2014). It has 240 × 180 resolution, 130 dB dynamic range and 3 microsecond latency. The DAVIS can concurrently output a stream of events and frame-based intensity read-outs using the same pixel array. An event consists of a pixel location (x, y), a binary polarity value (p) for positive or negative change in log intensity and a timestamp in microseconds (t). In this work, polarity of the events are not considered, and only the event stream of the DAVIS is used.

2.5.1. Parameter Settings

The time thresholds for the nearest neighbor filter and the refractory filter are nominally set to be $\Theta_{noise} = 5$ ms and $\Theta_{ref} = 1$ ms, respectively, as suggested in Padala et al. (2018). We used a FIFO buffer size of 5000 events for dynamically updating the count matrix as and when events are received. Subsequently, the RECT representation with a 2 by 2 averaging filter without zero padding at the boundaries is used to obtain a 9×9 feature vector for all event locations. We also experimented with other feature vector dimensions using a 3×3, 5×5, 7×7 sampling region and found that increasing the context improved the performance slightly. For obtaining the PCA-RECT representation, the number of PCs can be chosen automatically by retaining the PCs that hold 95% eigenenergy of the training data. For testing on the benchmark datasets, a codebook size of 3000 along with spatial pyramid matching (SPM) (Lazebnik et al., 2006) was universally used with a k-d tree with backtracking to find precise feature matches.

3. RESULTS

3.1. N-MNIST and N-Caltech101

The object categorization results on the N-MNIST and N-Caltech101 datasets are given in **Table 1**. As it is common

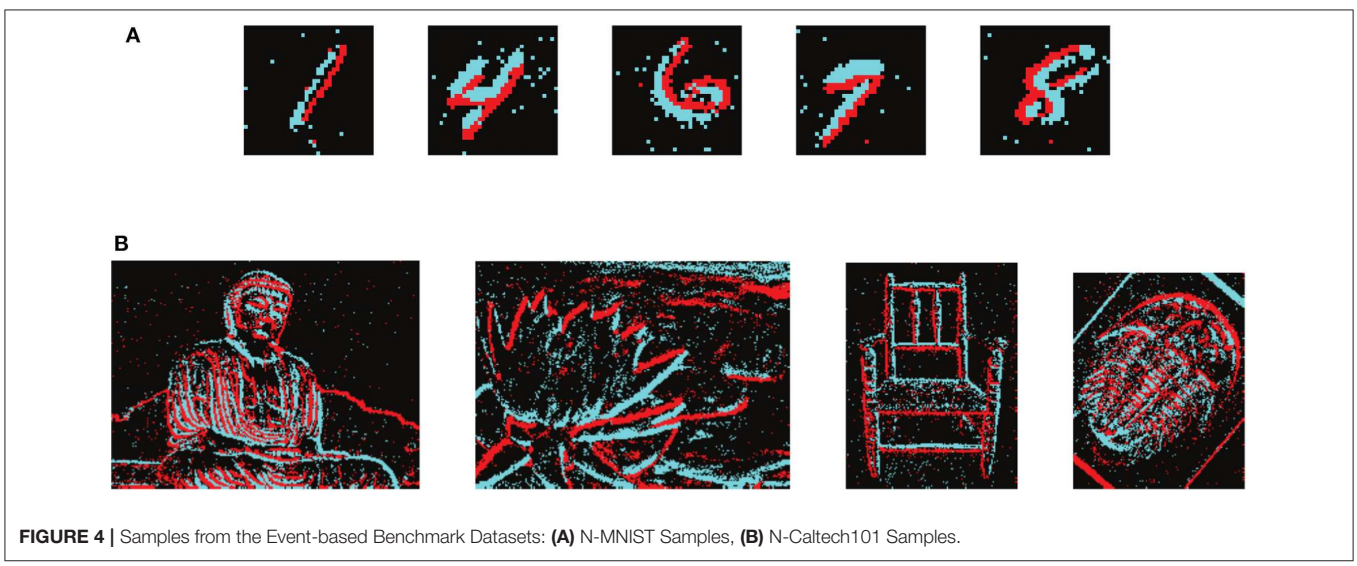


FIGURE 4 | Samples from the Event-based Benchmark Datasets: **(A)** N-MNIST Samples, **(B)** N-Caltech101 Samples.

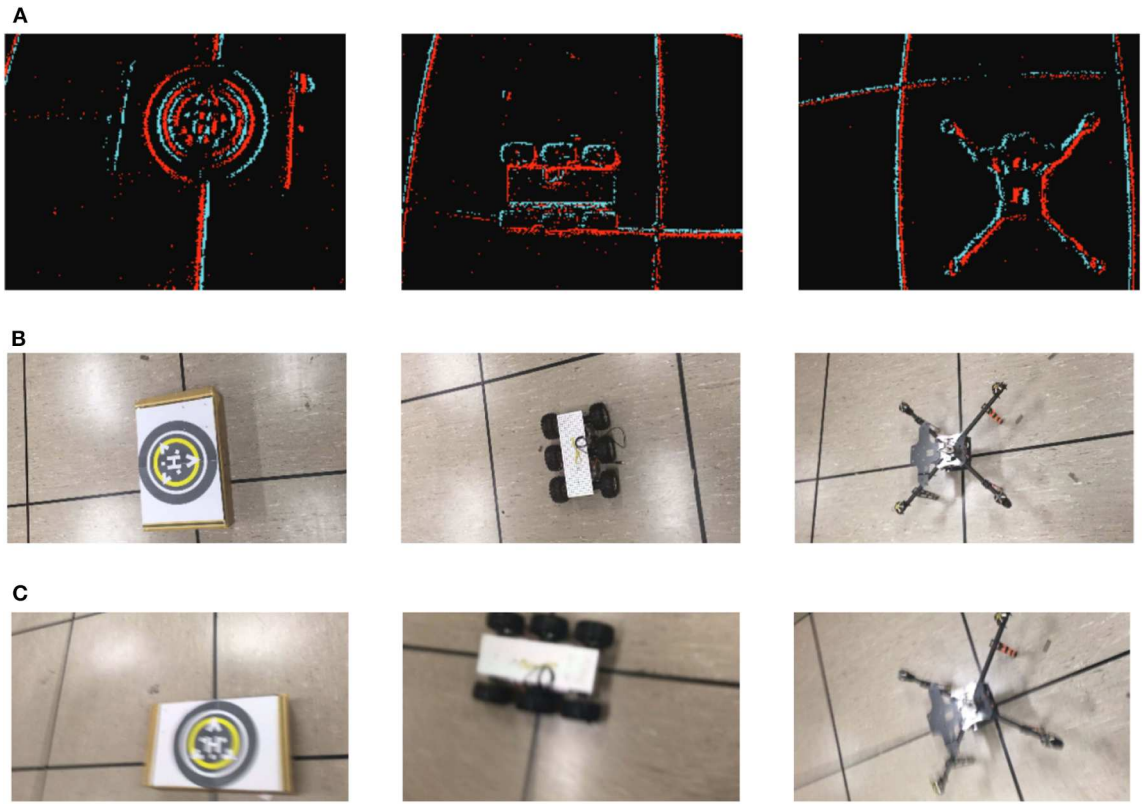


FIGURE 5 | Samples from the in-house datasets containing a Landing platform, UAV and Thumper (Empty Floor as background class): **(A)** N-SOD dataset, **(B)** Frame-based dataset similar to N-SOD, **(C)** Frame-based dataset with blur similar to N-SOD.

TABLE 1 | Comparison of classification accuracy on event-based datasets (%).

	N-MNIST	N-Caltech101
H-First	71.20	5.40
HOTS	80.80	21.0
Gabor-SNN	83.70	19.60
HATS	**99.10**	64.20
vPCA-RECT (this work)	98.72	70.25
PCA-RECT (this work)	98.95	**72.30**
Phased LSTM	97.30	–
Deep SNN	98.70	–

The bold values correspond to the best results for the N-MNIST and N-Caltech101 datasets.

practice, we report the results in terms of classification accuracy. The baselines methods considered were HATS (Sironi et al., 2018), HOTS (Lagorce et al., 2016), HFirst (Orchard et al., 2015b), and Spiking Neural Network (SNN) frameworks reported in Lee et al. (2016) and Neil et al. (2016) and Gabor-SNN as reported in Sironi et al. (2018).

On the widely reported N-MNIST dataset, our method is as good as the best performing HATS method (**Table 1**). Moreover, other SNN methods are also in the same ballpark, which is due to the simple texture-less digit event streams giving distinct features for most methods. Therefore, it is a good benchmark as long as a proposed method performs in the high 90's. A test on the challenging NCaltech-101 dataset will pave way for testing the effectiveness close to a real-world scenario. Our method has the highest classification rate ever reported for an event-based classification method on the challenging N-Caltech101 dataset. The unpublished HATS work is the only comparable method in terms of accuracy, while the other learning mechanisms fail to reach good performance.

3.1.1. Vary Hyper-Parameters

There are two important considerations while using the RECT representation: the feature dimension d obtained from the filtered matrix Equation (7) and the size of the filter itself ($p \times q$). Another way of interpreting the feature dimension is the "square grid length" that determines the number of filtered cells (these are the opaque rectangular grids containing more than one event in **Figure 1**) aggregated from the filtered event count matrix. This is easier to visualize and also vary in steps of 3×3, 5×5, 7×7, etc. In a similar vein, the pooling of the event count matrix (C) using the rectangular filter $A(p, q)$ results in a "sub-sampled" representation $R \in \mathbb{R}^{(m/p)\times(n/p)}$, and consequently, choosing a RECT patch of dimension d centered at $R(y/p, x/p)$ is equivalent to choosing a "larger radius" in the event count matrix $C \in \mathbb{R}^{(m)\times(n)}$ and then performing filtering and aggregation.

Again, it is easy to vary and visualize this RECT radius, say in steps of 5, instead of choosing various combinations of filter sizes (p, q). In the following, we vary the RECT grid and the radius to investigate the effects on classification performance using the N-Caltech101 dataset.

Figure 6A illustrates the classification performance trend observed for increasing radius of the event descriptor while keeping the resolution of the grid fixed. Similar to the trend observed in Ramesh et al. (2019a), a radius of more than 10 pixels results in sub-optimal performance. On the other hand, **Figure 6B** shows the effect of varying grid resolution on the accuracy. It is interesting to see that as the contextual information is captured finely using denser grids, while fixing the RECT radius to 10, there is a general increase in accuracy at the expense of increase in feature dimension. For instance, a 11×11 grid already results in a high feature dimension of 121 and thus increasing the complexity of the subsequent feature matching step using the k-d tree. In our application using N-SOD, presented in the next subsection, a 9×9 grid with a radius of 10 was used. Next, the performance of the feature selection methods are investigated.

Figure 7A shows the performance of the feature selection methods (PCA and vPCA). As expected for PCA, increase in the number of PCs results in better performance until about the 95% eigenenergy cut-off, which is typically about 60 in our case. It is also worth noticing that just retaining five dimensions can give better performance compared to existing works. For vPCA feature selection, the number of selected features depends on the size of the smaller evaluation codebook. The smaller the evaluation codebook, the lesser the entropy, and thus lesser the number of features selected. Similar to PCA, when noisy features are discarded from the RECT representation, the classifier performance increases.

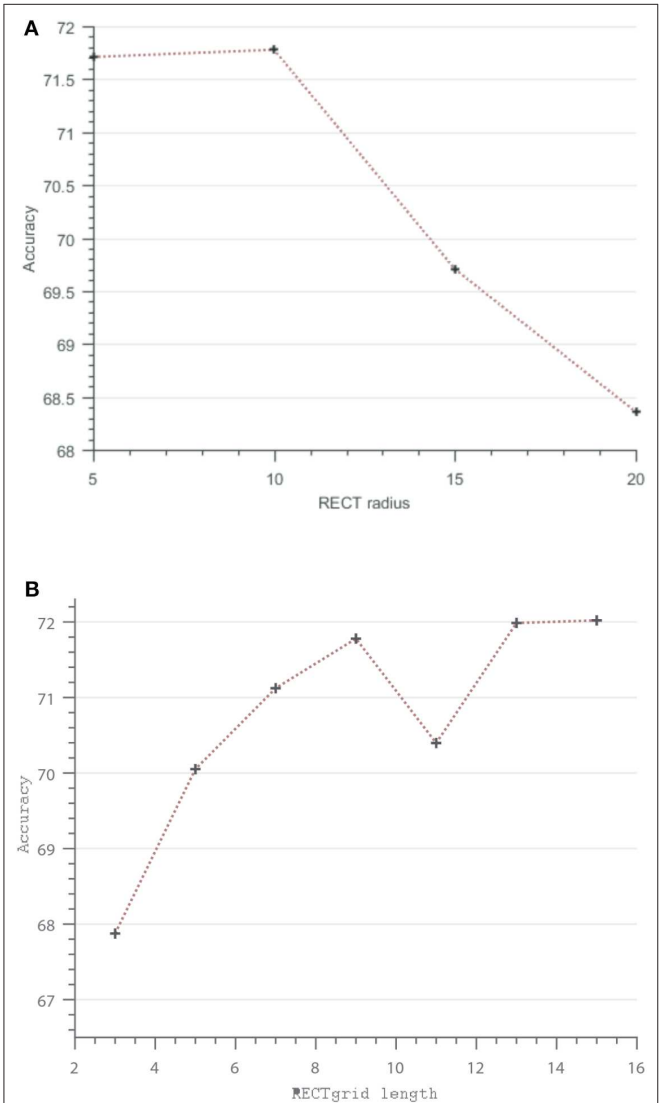


FIGURE 6 | Vary RECT parameters: **(A)** RECT radius vs. accuracy, **(B)** RECT square grid length vs. accuracy.

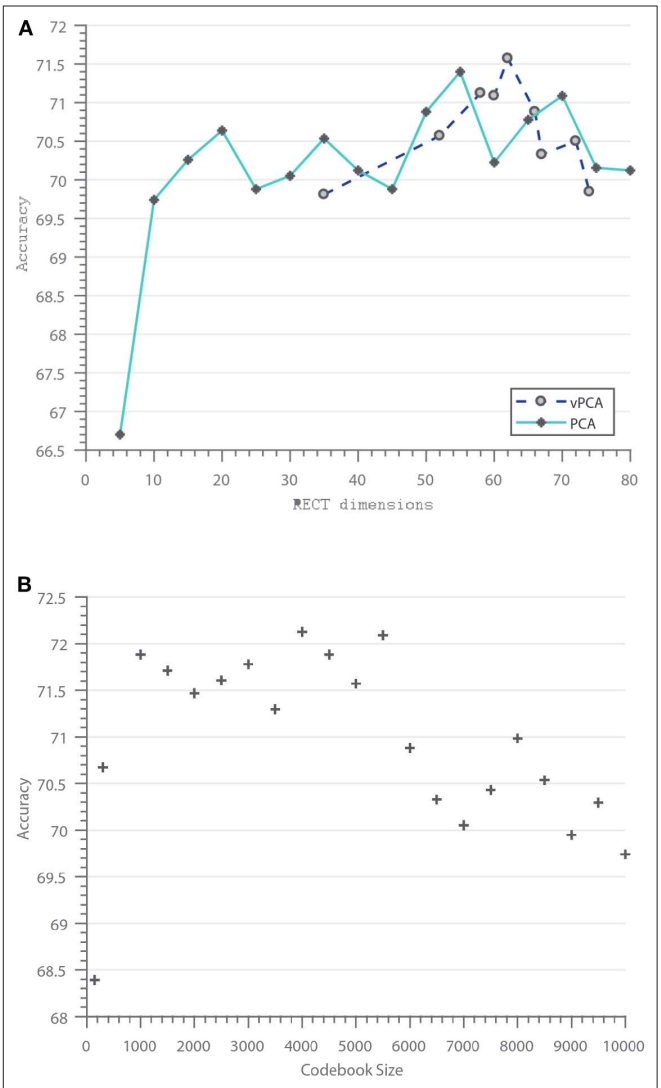


FIGURE 7 | Vary feature dimension and codebook size: **(A)** Feature selection methods (PCA, vPCA) vs. Classification Accuracy, **(B)** Codebook size vs. Classification accuracy.

Besides feature selection, larger dictionaries or codebooks tend to provide higher classification accuracy (Nowak et al., 2006), however, the high-dimensionality of the object representation when combined with spatial pyramid matching (14 times the codebook size for a 1×1, 2×2, and 3×3 SPM representation) can degrade the performance for larger codebooks, as shown in **Figure 7B**, where two distinct clusters can be spotted. Codebook sizes less than 5000 with SPM perform better than larger codebooks. This trend has been observed in previous works as well (Lazebnik et al., 2006).

3.2. N-SOD

For testing on the N-SOD dataset, we divide the dataset into training and testing, with 80% temporal sequence samples per class for training and the remaining for testing. Using the training features, a dictionary is generated. Since the temporal sequences are of different length, for a fixed number of events, say every 10^5 events, an object representation is extracted using the codebook and a linear SVM classifier is trained. Similarly for testing, for every 10^5 events, the object representation is classified using the SVM.

Based on the above setup, an accuracy of 97.14% was obtained (**Table 2**) with a dictionary size of 950, which resulted in a k-d tree with 10 layers. We also experimented with lower dictionary sizes such as 150, 300, 450, etc., and the performance drop was insignificant (>96%). On the other hand, using a k-d tree with backtracking, descriptor normalization, etc., achieved close to 100% accuracy on offline high-performance PCs, which of course does not meet low-power and real-time requirements. In summary, the proposed vPCA-RECT method with a backtracking-free k-d tree implementation mildly compromises on accuracy to handle object detection and categorization using an event camera in real-time.

We report the precision and recall of the detection results by ascertaining if the mean position of the detected result is within the ground truth bounding box. We obtained: (a) *Precision* - $(498/727) = 0.685$: The percentage of the detections belonging to the object that overlap with the groundtruth (b) *Recall* - $(498/729) = 0.683$: The percentage of correct detections that are retrieved by the system. The number of "landmarks" were set to 20 in the above experiments while similar results were obtained for values such as five and ten. It is worth pointing out that the codebook size used for 4-class N-SOD detection and recognition, thereby for the FPGA implementation, need not be in the thousands as with the complex N-Caltech101 dataset for giving high accuracy.

TABLE 2 | Confusion Matrix (%) for the best result on the in-house N-SOD dataset.

	Background	LP	Thumper	UAV
Background	95.4128	0.3058	3.3639	0.9174
LandingPlatform	0	99.2268	0.5155	0.2577
Thumper	0	1.9257	96.9739	1.1004
UAV	0	0	3.1884	96.8116

3.3. FPGA Performance

The hardware implementation and performance of the Xilinx Zynq-7020 FPGA running at 100 MHz was evaluated by direct comparison with the results of the algorithm's software version in MATLAB. The Zynq was interfaced to a down-looking DAVIS camera, on-board an unmanned aerial vehicle flying under unconstrained lighting scenarios. We recommend viewing our submitted video[1] that clearly shows the classification/detection process better than still images. Vivado design suite was used for synthesis and analysis of the design. The in-built logic simulator ISIM was used for testing; first, to verify that the behavior was met, and later for verification of timing performance and latency requirements post-synthesis and post-implementation.

3.3.1. Timing

The time taken for a single event to be classified for the worst possible k-d tree path was 560 nanoseconds, where roughly 80% of the time is employed traversing the tree. The rest is employed for buffering (5%), count matrix updating (5%), and SVM inference (10%). On the other hand, the detection task includes a comparison of codewords and consequent updating of the detection count matrix, which happens for each event and takes 50 nanoseconds. Later, the mean calculation between the respective coordinates consists of a summation and division. The former is proportional to the number of values in the operation and takes one clock cycle per element (in operation this approximates to 5 values), and the latter requires 80 nanoseconds of processing. This amounts to 130 nanoseconds which is negligible since it only happens once for every set of valid classified events (S_d).

Due to the asynchronous nature of the sensor, it is not uncommon to receive a consecutive batch of events in a very short period (say 10 μs). These events cannot be handled in parallel, since each of them modifies the classification count matrix, and the SVM feature representation. Then, the events that arrive while the tree is been traversed are buffered and later processed. This may add a delay in the output of about 2 event cycles (about 1 μs) depending on the amount of events triggered at the same instant, however, the refractory filter avoids this case for multiple events triggered at the same pixel. In any case, the DAVIS camera output has a minimum event throughput at 1μs (mean inter-event interval about 10 μs), and thus is a rare case inhibiting real-time processing.

The classification and detection tasks are performed in parallel and follow different periods of operation. Classification is applied on a FIFO storing the last S_c events, and it is consequently updated for each incoming event. Hence, a classification output is provided for each new input data. Separately, the detection pipeline works on a periodic basis and only for a specific classification result, providing a valid output once every S_d events.

The latency of the system is on par with similar works on neuromorphic vision tracking on FPGA (Moeys et al., 2016; Linares-Barranco et al., 2019), taking into consideration that these works are implemented for low-level object tracking.

[1]Demo: https://youtu.be/yWfCmHnV5f0

Additionally, our system outperforms similar applications using frame-based cameras using FPGA or microprocessors, which by definition normally operate in the scale of milliseconds. **Table 4** presents a summary of these measurements.

3.3.2. Resource Utilization

A summary of utilization of hardware elements can be seen in **Table 3**. The modules corresponding to the *k*-d tree and SVM require memory initialization to store tree nodes properties and SVM coefficients. Hence, Read-Only Memory (ROM) was utilized for this purpose. This accounts to 128 KB for the *k*-d tree module and 180 KB for the SVM module. These resources are synthesized into RAM blocks in the FPGA, but these are only used for reading as would be the case with a regular ROM element. Digital signal processing (DSP) slices were utilized to perform integer division. There are two division operations in the detection pipeline, and each of these dividers require two DSP slices; one multiply block and one multiply adder block.

3.3.3. Power Consumption

Table 4 also lists the power consumption of the vPCA-RECT system in comparison to state-of-the-art methods. For our system, the DAVIS event camera operates at a few milliwatts (10 mW) while the Zynq operates at about 3 W including the base power for running Ubuntu. The algorithmic implementation itself increases the dynamic on-chip power by only 0.37 W. As a comparison, event-based blob tracking implementation on FPGA (Moeys et al., 2016) reported 0.775 W running at 50 MHz. In general, FPGA-based recognition systems for RGB cameras (Schlessman et al., 2006; Hikawa and Kaida, 2015; Mousouliotis et al., 2018), which present solutions running at equal or lower clock frequencies, consume more power than our implementation. Similarly, Zhai et al. (2013) and Ali et al. (2009) present works that take advantage of the mixed computation capabilities of the Xilinx Zynq chip, but get hindered by the high latency characteristic of a frame-based system.

To provide a broader context to the power consumption of frame-based systems, let us consider (De Smedt et al., 2015) (Brix embedded system, Intel-I7 processor with 8G RAM) used for real-time object tracking. It consumes about 22 W, which is 7x more than our full hardware implementation. Note that the Zynq module is a powerful and flexible development tool, but far exceeds the utilities compared to SmartFusion FPGAs that allow sleep modes, non-volatile configuration memory, and have much lower power consumption overall. In other words, there is significant room for very low-power implementation (less than 1 W) of our framework with appropriate hardware choices and development efforts.

TABLE 3 | Hardware utilization report for the FPGA running the proposed modules.

	Utilization	Available	Utilization %
LUT	18238	53200	34.28
LUTRAM	12124	17400	69.68
FF	2065	106400	1.94
BRAM	48	140	34.29
DSP	4	220	1.82
IO	102	200	51.00

TABLE 4 | Comparison of power consumption and latency of existing object detection systems with the proposed method.

	Frequency (MHz)	Power (Watts)	Latency (ns)
vPCA-RECT (ours)	100	**0.37**	560
Moeys et al. (2016)	50	0.78	440
Zhai et al. (2013)	58	0.90	11000 k
Ali et al. (2009)	50	0.66	91000 k
De Smedt et al. (2015)	2600	22.0	∼

The bold value correspond to our system's power consumption.

3.4. Comparison to CNN

In order to compare to state-of-the-art deep neural networks, we recorded a similar dataset to N-SOD using a frame-based camera (**Figure 5B**) and transfer learning via AlexNet classified the object images. The total number of images recorded were in the order of 6000. With an equivalent train/test split compared to N-SOD, perfect performance can be achieved on the clearly captured test images. In fact, as **Figure 8** shows, perfect performance can be achieved on the test images with as little as 5% of the data used for training. It is indeed surprising to see that with 0.5% training data (8 samples per object category), the accuracy can be near perfect.

However, when we tested the Alexnet model (trained on normal images) on a dataset under motion blur conditions (**Figure 5C**), an accuracy of only 79.20% was obtained. It was clear that the black UAV frame when blurred looks like the black-stripped background and creates much confusion as seen from

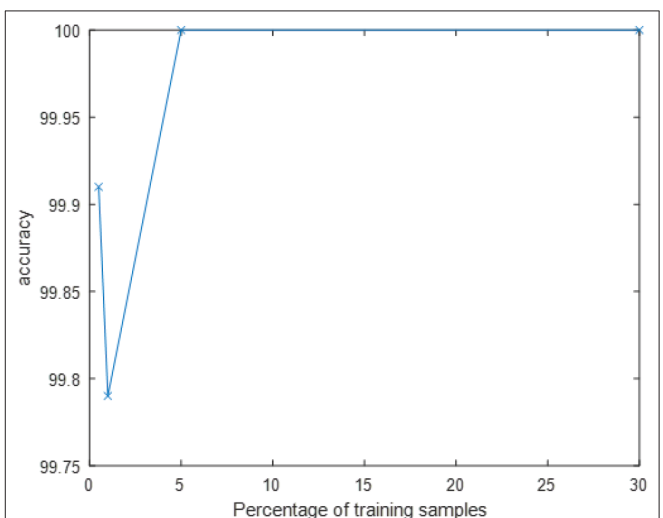


FIGURE 8 | Alexnet test accuracy vs. Percentage of training samples.

TABLE 5 | Confusion Matrix (%) for CNN classifier on the "blur" frame-based dataset (%).

	Background	LP	Thumper	UAV
Background	100.00	0	0	0
Landing platform	0.64	99.36	0	0
Thumper	9.63	0	90.37	0
UAV	72.02	0	9.3	27.05

Table 5. This confirms the disadvantage of using frame-based cameras to handle unconstrained camera motion. Note that fast camera motion leads to only an increase in data-rate for event-based cameras and has no effect on the output. In fact, recordings of N-SOD have significant amount of such fast motions.

Additionally, we recorded images to test the performance of CNN on data captured under low-lighting conditions and slow motion conditions. A near-perfect performance on these set of images was impressive, as the features extracted by CNN were robust enough to be invariant under extreme lighting conditions. Similarly, one could argue that "blurred" images when included in the training will boost the accuracy of the deep learning model. We confirmed that by training on 30% normal (1976 images) plus 3% blur (72 images), and testing on the rest of the data captured under normal, blur, and low-lighting conditions. This mixed testing allowed the CNN to correctly classify the UAV blurred images (99.4% accuracy). Nonetheless, this is a rather unnatural training setting, one that is not expected to be deployed in the real-world. Moreover, other works have also concluded that existing networks are indeed susceptible to many image quality issues, particularly to blur and noise (Dodge and Karam, 2016).

4. DISCUSSION

We have demonstrated object detection and categorization in an energy-efficient manner using event cameras, where the only information that is important for these tasks is how edges move, and the event camera naturally outputs it. The proposed PCA-RECT feature takes advantage of this sparsity to generate a low-dimensional representation. The low-dimensional representation is further exploited for feature matching using a k-d tree approach, capable of obtaining the best performance on the challenging Neuromorphic Caltech-101 dataset compared to state-of-the-art works.

Although k-d trees enable fast and large scale nearest neighbor queries among high dimensional data points, such as those produced by RECT or PCA-RECT[2], their application is restricted to efficiently computing distance measures. Thus, as long as there are descriptors, global or local, k-d trees are a good fit to both event data and RGB frames. Nonetheless, it remains to be seen whether global event-based descriptors, say HATS (Sironi et al., 2018), will benefit from k-d trees. On the other hand, the sparsity of events leads to less data compared to intensity frames recorded at 30 Hz or 10 MB/s (the DAVIS outputs typically at 150KB/s). This tends to lend well to the use of k-d trees, given there will be lesser information to build and decode. Overall, k-d trees could be better utilized for real-time and embedded applications for event camera data compared to RGB frames, and its performance remains to be fully explored.

It is important to note that we demonstrated very competitive performance compared to Deep SNN on the N-MNIST dataset using the proposed dictionary-based framework. However, it is indeed expected that deep features learned using neural networks shall outperform hand-crafted features, such as PCA-RECT, in the future. Even so, it is non-trivial as to how a deep learning approach can be effectively and efficiently suited to a purely spike-based or event-based data. In this work, real-time FPGA implementation was achieved with several careful design considerations, such as a backtracking-free k-d tree for matching to the codewords, a virtual PCA-RECT representation obtained by analyzing the k-d tree partitioning of the feature space, etc. To the best of our knowledge, this is the first work implementing a generic object recognition framework for event cameras on an FPGA device, verified in a lab demo setting under unconstrained motion and lighting setup, thereby demonstrating a high performance over resource ratio. Additionally, it is well-known that dictionary-based methods easily scale to the number of samples, since performance depends only on the codebook size. For instance, searching a large image dataset with 10 million images takes only about 50 ms (Jégou et al., 2010) using compact representations of descriptors. This type of large-scale recognition using event cameras shall be a future research direction for us and the larger neuromorphic vision community.

In terms of comparison to a frame-based setup, we found the average elapsed time for feature extraction of a single image under CPU execution environment (0.6726 s) hindered real-time performance. However, this latency can be drastically reduced using GPUs while power consumption increases significantly. This is an important case where frame-based paradigm is unfavorable compared to the low-power event-based implementation presented in this paper. Additionally, under fast moving conditions of the sensor, frame-based CNN was shown to perform unreliably. Thus, when event-based processing can accomplish higher accuracy and reliability under low-power settings, as demonstrated in this work under closed-loop conditions, there is great potential for silicon retinas as an alternative or complimentary visual sensor for myriad other applications.

DATA AVAILABILITY STATEMENT

All datasets generated for this study are included in the article/supplementary material. The N-SOD data set is available at https://tinyurl.com/s84nlm4 and the source code (MATLAB and VHDL) is available at https://github.com/nusneuromorphic.

[2]PCA-RECT feature can be high-dimensional when the number of chosen dimensions are close to that of the corresponding RECT representation.

AUTHOR CONTRIBUTIONS

BR: thesis director and main contributor. Formalized the theory, implemented the algo in MATLAB and evaluated the results. AU: ported the algo to FPGA hardware and evaluated the results. HY: assist BR in software experiments and verify hardware feasibility using CPP implementation. LD: hardware system integration and filtering implementation. GO: co-supervisor and instigator of the work.

FUNDING

This work was supported in part by the Temasek Research Fellowship awarded to GO.

ACKNOWLEDGMENTS

The authors would like to thank Chen Yu and other lab members who assisted in data collection.

REFERENCES

Ali, U., Malik, M. B., and Munawar, K. (2009). "FPGA/soft-processor based real-time object tracking system," in *Proceedings - 2009 5th Southern Conference on Programmable Logic, SPL 2009*, 33–37.

Arya, S., and Mount, D. M. (1993). "Algorithms for fast vector quantization," in *Data Compression Conference* (Sao Carlos), 381–390.

Beis, J. S., and Lowe, D. G. (1997). "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (San Juan: IEEE Computer Society), 1000–1006.

Brandli, C., Berner, R., Yang, M., Liu, S. C., and Delbruck, T. (2014). A 240 x 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE J. Solid State Circuits* 49, 2333–2341. doi: 10.1109/JSSC.2014.2342715

Conradt, J., Berner, R., Cook, M., and Delbruck, T. (2009). "An embedded AER dynamic vision sensor for low-latency pole balancing," in *IEEE International Conference on Computer Vision Workshop* (Kyoto), 780–785.

De Smedt, F., Hulens, D., and Goedeme, T. (2015). "On-board real-time tracking of pedestrians on a uav," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (Boston, MA), 1–8.

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

Dodge, S., and Karam, L. (2016). "Understanding how image quality affects deep neural networks," in *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)* (Lisbon), 1–6.

Galleguillos, C., Rabinovich, A., and Belongie, S. (2008). "Object categorization using co-occurrence, location and appearance," in *IEEE Conference on Computer Vision and Pattern Recognition* (Anchorage, AK: IEEE Computer Society), 1–8.

Hikawa, H., and Kaida, K. (2015). Novel fpga implementation of hand sign recognition system with som-hebb classifier. *IEEE Trans. Circuits Syst. Video Technol.* 25, 153–166. doi: 10.1109/TCSVT.2014.2335831

Iacono, M., Weber, S., Glover, A., and Bartolozzi, C. (2018). "Towards event-driven object detection with off-the-shelf deep learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Madrid), 1–9.

Jégou, H., Douze, M., Schmid, C., and Pérez, P. (2010). "Aggregating local descriptors into a compact image representation," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (San Francisco, CA), 3304–3311.

Kueng, B., Mueggler, E., Gallego, G., and Scaramuzza, D. (2016). "Low-latency visual odometry using event-based feature tracks," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Daejeon), 16–23.

Lagorce, X., Orchard, G., Gallupi, F., Shi, B. E., and Benosman, R. (2016). Hots: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Lazebnik, S., Schmid, C., and Ponce, J. (2006). "Beyond bags of features: spatial pyramid matching for recognizing natural scene categories," in *Computer Vision and Pattern Recognition*, Vol. 2 (New York, NY), 2169–2178.

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

Lenz, G., Ieng, S., and Benosman, R. (2018). Event-based dynamic face detection and tracking based on activity. *arXiv[Preprint]*.arXiv:1803.10106.

Linares-Barranco, A., Perez-Pena, F., Moeys, D. P., Gomez-Rodriguez, F., Jimenez-Moreno, G., Liu, S.-C., et al. (2019). Low latency event-based filtering and feature extraction for dynamic vision sensors in real-time FPGA applications. *IEEE Access* 7, 134926–134942. doi: 10.1109/ACCESS.2019.2941282

Liu, H., Moeys, D. P., Das, G., Neil, D., Liu, S. C., and Delbruck, T. (2016). "Combined frame- and event-based detection and tracking," in *IEEE International Symposium on Circuits and Systems (ISCAS)* (Montreal, QC), 2511–2514.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision* 60, 91–110. doi: 10.1023/B:VISI.0000029664.99615.94

Manderscheid, J., Sironi, A., Bourdis, N., Migliore, D., and Lepetit, V. (2019). "Speed invariant time surface for learning to detect corner points with event-based cameras," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA), 1–10.

Moeys, D. P., Delbruck, T., Rios-Navarro, A., and Linares-Barranco, A. (2016). "Retinal ganglion cell software and FPGA model implementation for object detection and tracking," in *Proceedings - IEEE International Symposium on Circuits and Systems* (Montreal, QC: Institute of Electrical and Electronics Engineers Inc.), 1434–1437.

Mousouliotis, P. G., Panayiotou, K. L., Tsardoulias, E. G., Petrou, L. P., and Symeonidis, A. L. (2018). "Expanding a robot's life: low power object recognition via fpga-based dcnn deployment," in *7th International Conference on Modern Circuits and Systems Technologies (MOCAST)* (Thessaloniki), 1–4.

Muja, M., and Lowe, D. G. (2009). "Fast approximate nearest neighbors with automatic algorithm configuration," in *VISAPP International Conference on Computer Vision Theory and Applications* (Lisbon), 331–340.

Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). "Phased LSTM: accelerating recurrent network training for long or event-based sequences," in *Neural Information Processing Systems, NIPS'16* (Barcelona: Curran Associates Inc.), 3889–3897.

Ni, Z., Bolopion, A., Agnus, J., Benosman, R., and Regnier, S. (2012). Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics. *IEEE Trans. Robot.* 28, 1081–1089. doi: 10.1109/TRO.2012.2198930

Nowak, E., Jurie, F., and Triggs, B. (2006). "Sampling strategies for bag-of-features image classification," in *European Conference on Computer Vision*, Vol. 3954 (Graz), 490–503.

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. Neurosci.* 7:178. doi: 10.3389/fnins.2013.00178

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

Orchard, G., Meyer, C., Etienne-Cummings, R., Posch, C., Thakor, N., and Benosman, R. (2015b). HFirst: a temporal approach to object recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 2028–2040. doi: 10.1109/TPAMI.2015.2392947

Padala, V., Basu, A., and Orchard, G. (2018). A noise filtering algorithm for event-based asynchronous change detection image sensors on truenorth and its implementation on truenorth. *Front. Neurosci.* 12:118. doi: 10.3389/fnins.2018.00118

Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B., and Delbruck, T. (2014). Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. *Proc. IEEE* 102, 1470–1484. doi: 10.1109/JPROC.2014.2346153

Ramesh, B., Jian, N. L. Z., Chen, L., Xiang, C., and Gao, Z. (2017a). Scalable scene understanding via saliency consensus. *Soft Comput*. 23, 2429–2443. doi: 10.1007/s00500-017-2939-2

Ramesh, B., Thi, L., Orchard, G., and Xiang, C. (2017b). "Spike context: a neuromorphic descriptor for pattern recognition," in *IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Turin), 1–4.

Ramesh, B., Ussa, A., Vedova, L. D., Yang, H., and Orchard, G. (2019a). "PCA-RECT: an energy-efficient object detection approach for event cameras," in *ACCV 2018 Workshops* (Perth, WA: Springer International Publishing), 434–449.

Ramesh, B., Yang, H., Orchard, G. M., Le Thi, N. A., Zhang, S., and Xiang, C. (2019b). Dart: distribution aware retinal transform for event-based cameras. *IEEE Trans. Pattern Anal. Mach. Intell.* doi: 10.1109/TPAMI.2019.2919301. [Epub ahead of print].

Redmon, J., and Farhadi, A. (2018). Yolov3: an incremental improvement. *CoRR*, abs/1804.02767.

Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1137–1149. doi: 10.1109/TPAMI.2016.2577031

Scheerlinck, C., Barnes, N., and Mahony, R. (2018). "Continuous-time intensity estimation using event cameras," in *Asian Conference on Computer Vision* (Perth, WA: Springer International Publishing), 308–324.

Schlessman, J., Cheng-Yao Chen, Wolf, W., Ozer, B., Fujino, K., and Itoh, K. (2006). "Hardware/software co-design of an FPGA-based embedded tracking system," in *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)* (New York, NY), 123–123.

Silpa-Anan, C., and Hartley, R. (2008). "Optimised kd-trees for fast image descriptor matching," in *IEEE Conference on Computer Vision and Pattern Recognition* (Anchorage, AK), 1–8.

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). HATS: histograms of averaged time surfaces for robust event-based object classification. *arXiv[Preprint]*.arXiv:1803.07913.

Vikram, T. N., Tscherepanow, M., and Wrede, B. (2012). A saliency map based on sampling an image into random rectangular regions of interest. *Pattern Recogn.* 45, 3114–3124. doi: 10.1016/j.patcog.2012.02.009

Zhai, X., Bensaali, F., and McDonald-Maier, K. (2013). "Automatic number plate recognition on fpga," in *International Conference on Electronics, Circuits, and Systems (ICECS)* (Abu Dhabi: IEEE), 325–328.

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 Neuroscience

ORIGINAL RESEARCH
published: 28 February 2020
doi: 10.3389/fnins.2020.00150

Synaptic Delays for Insect-Inspired Temporal Feature Detection in Dynamic Neuromorphic Processors

Fredrik Sandin and Mattias Nilsson**

Embedded Intelligent Systems Lab (EISLAB), Luleå University of Technology, Luleå, Sweden

Spiking neural networks are well-suited for spatiotemporal feature detection and learning, and naturally involve dynamic delay mechanisms in the synapses, dendrites, and axons. Dedicated delay neurons and axonal delay circuits have been considered when implementing such pattern recognition networks in dynamic neuromorphic processors. Inspired by an auditory feature detection circuit in crickets, featuring a delayed excitation by post-inhibitory rebound, we investigate disynaptic delay elements formed by inhibitory–excitatory pairs of dynamic synapses. We configured such disynaptic delay elements in the DYNAP-SE neuromorphic processor and characterized the distribution of delayed excitations resulting from device mismatch. Interestingly, we found that the disynaptic delay elements can be configured such that the timing and magnitude of the delayed excitation depend mainly on the efficacy of the inhibitory and excitatory synapses, respectively, and that a neuron with multiple delay elements can be tuned to respond selectively to a specific pattern. Furthermore, we present a network with one disynaptic delay element that mimics the auditory feature detection circuit of crickets, and we demonstrate how varying synaptic weights, input noise and processor temperature affect the circuit. Dynamic delay elements of this kind open up for synapse level temporal feature tuning with configurable delays of up to 100 ms.

Keywords: pattern recognition, spiking neural network (SNN), neuromorphic, delay line, embedded intelligence, DYNAP, insect-inspired computing

OPEN ACCESS

Edited by:
Elisabetta Chicca,
Bielefeld University, Germany

Reviewed by:
Federico Corradi,
Imec, Netherlands
Tara Julia Hamilton,
University of Technology Sydney, Australia

****Correspondence:***
Fredrik Sandin
fredrik.sandin@ltu.se
Mattias Nilsson
mattias.1.nilsson@ltu.se

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *09 October 2019*
Accepted: *07 February 2020*
Published: *28 February 2020*

Citation:
Sandin F and Nilsson M (2020) Synaptic Delays for Insect-Inspired Temporal Feature Detection in Dynamic Neuromorphic Processors. Front. Neurosci. 14:150. doi: 10.3389/fnins.2020.00150

1. INTRODUCTION

Processing of temporal patterns in signals is a central task in perception, learning, and control of behavior in both biological and technological systems (Indiveri and Sandamirskaya, 2019). Unlike digital circuits, which are designed to perform precise logic and arithmetic operations, neurons are unreliable, stochastic and slow information processing entities which form networks that function reliably through distributed information processing and adaptation. Neural circuits are therefore interesting models for development of mixed signal analog–digital processing and perception systems implemented in resource efficient nano-electronic substrates that are subject to device mismatch and failure (Strukov et al., 2019). In particular, energy-efficient neuromorphic processors and sensor systems have been developed by matching the device dynamics to neural dynamics, for example in the form of CMOS analog circuits operating in the subthreshold regime where semiconductor electron diffusion mimics ion diffusion in biological ion channels (Mead, 1990; Indiveri et al., 2011; Schuman et al., 2017). The dynamic nature and spatial structure of biological neurons (synapses, dendrites, axons, etc.) implies that neurons are inherently capable of temporal pattern recognition (Mauk and Buonomano, 2004) and pattern generation, also without

recurrent connections. Furthermore, the event-driven neurons in Spiking Neural Networks (SNNs) are typically sparsely activated and offer an efficient way of doing inference (Rueckauer et al., 2017). SNNs with biologically plausible dynamics thus offer an interesting alternative model for temporal and spatial (spatiotemporal) pattern recognition (Pfeiffer and Pfeil, 2018), which can be further developed with guidance from biology. However, it is an open problem how such neuromorphic pattern recognition solutions can be engineered in practical applications such that the dynamic nature of the hardware is efficiently exploited.

Delays are essential for neuromorphic processing of temporal patterns in spike trains (Sheik et al., 2013) and have been studied since the early 90s, see for example the work by Van der Spiegel et al. (1994). Temporal delays have for example been implemented in neuromorphic processors in the form of multicompartment models (Hussain et al., 2015; Schemmel et al., 2017) and dedicated, specifically tuned delay neurons in the network architecture (Sheik et al., 2012a,b; Coath et al., 2014). In the latter case the resulting SNN is similar to a model of the auditory thalamocortical system described by Coath et al. (2011). Nielsen et al. (2017) present a low-power pulse delay and extension circuit for neuromorphic processors, which implements programmable axonal delays ranging from fractions of microseconds up to tens of milliseconds. In polychronous (Izhikevich, 2006) architectures, asynchronously firing neurons project to a common target along delay lines so that spikes arrive at the target neuron simultaneously, thus causing it to fire. A polychronous SNN with delay adaptation for spatiotemporal pattern recognition has been implemented in a Field-Programmable Gate Array (FPGA) and in a custom mixed-signal neuromorphic processor (Wang et al., 2013, 2014).

The typical signal propagation delays in axons (Swadlow, 1985) and dendrites (Agmon-Snir and Segev, 1993) of cortical neurons range up to tens of milliseconds. Furthermore, the dynamics of synapses also play an important role for the processing of temporal and spatiotemporal patterns (Mauk and Buonomano, 2004) and offer efficient dynamic mechanisms for sequence detection and learning (Buonomano, 2000). Synaptic dynamics enable pattern recognition architectures with high fan-in, which is beneficial in neuromorphic systems where multicompartment modeling, axon/neuron reservation and spike transmission is costly. Rost et al. (2013) present an SNN architecture with spike frequency adaptation and synaptic short-term plasticity that models auditory pattern recognition in cricket phonotaxis. There, synaptic short-term depression and potentiation is implemented to make neurons act as high-pass and low-pass filters, respectively. The resulting signals are combined in a neuron that acts as a band-pass filter and thereby responds to a frequency band that is matched to the particular sound pulse period of the crickets. Insects offer interesting opportunities to develop neuromorphic systems by modeling and finding guidance from their neural circuits, where the relatively low complexity allows neuromorphic engineers to transfer the principles of neural computation to applications (Dalgaty et al., 2018).

Our present investigation is inspired by a more recent description of the cricket auditory system (Schöneich et al., 2015) and preliminary work (Nilsson, 2018) indicating that dynamic synapses in a neuromorphic processor can be used to imitate the post-inhibitory rebound of the non-spiking delay neuron in the auditory circuit of the cricket. We configured disynaptic delay elements composed of inhibitory and excitatory dynamic synapses in the low-power Dynamic Neuromorphic Asynchronous Processor (DYNAP) model SE from aiCTX (Moradi et al., 2018). DYNAP-SE features reconfigurable mixed-mode analog/digital neuron and synapse circuits with biologically faithful dynamics. We investigated the properties and parameter dependence of the disynaptic delay elements in a population of neuromorphic neurons and found that delayed excitations of up to 100 ms can be achieved, and that the parameters can be selected so that the delay and delayed excitation amplitude depends mainly on the synaptic efficacies. Furthermore, we imitated the post-inhibitory rebound of the non-spiking neuron in the auditory circuit of the cricket (Schöneich et al., 2015) with one disynaptic element and investigated a circuit with three spiking neurons that reliably detects the species-specific sound-pulse interval of 20 ms. Since delays of tens of milliseconds are useful for implementing different kinds of neural circuits, cortical circuits in particular, the easily configurable properties of the disynaptic delay elements described and characterized in the following open up for further implementations and studies of SNN architectures in neuromorphic processors.

2. MATERIALS AND METHODS

2.1. The DYNAP-SE Neuromorphic Processor

The DYNAP-SE neuromorphic processor uses a combination of low-power, inhomogeneous sub-threshold analog circuits and fast, programmable digital circuits for the emulation of SNN architectures with bio-physically realistic neuronal and synaptic behaviors (Moradi et al., 2018), making it a platform for spike-based neural processing with co-localized memory and computation (Indiveri and Liu, 2015). Specifically, the DYNAP-SE comprises four-core neuromorphic chips, each with 1k analog silicon neuron circuits. Each neuron has a Content-Addressable Memory (CAM) block containing 64 addresses representing the presynaptic neurons that the neuron is connected to. Information about spike-activity is transmitted between neurons in an Address-Event Representation (AER) digital routing scheme. Four different types of synaptic behavior are available for each connection: Fast excitatory, slow excitatory, subtractive inhibitory, and shunting inhibitory. The dynamic behaviors of the neuronal and synaptic circuits of the DYNAP-SE are governed by analog circuit parameters which are set by programmable on-chip temperature compensated bias-generators (Delbruck et al., 2010).

The inhomogeneity of the analog low-power circuits that constitute the neurons and synapses of the DYNAP-SE neuromorphic processor is due to device mismatch, and gives rise to variations in the dynamic behaviors of the silicon neurons and synapses that the analog circuits constitute. These variations are analogous to differences in values of the parameters governing

the differential equations that model the neuronal and synaptic dynamics implemented in the chips. Consequently, one set value of a neuronal or synaptic bias parameter, in one core of the DYNAP-SE, results in a distribution of the corresponding parameter values in the population of neurons and synapses of that core.

2.1.1. Spiking Neuron Model

In the DYNAP-SE, neurons are implemented according to the Adaptive Exponential Integrate-and-Fire (AdEx) spiking neuron model (Brette and Gerstner, 2005). The model describes the neuron membrane potential, V, and the adaptation variable, w, with two coupled non-linear differential equations,

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T e^{(V-V_T)/\Delta_T} - w + I, \quad (1a)$$

$$\tau_w\frac{dw}{dt} = a\,(V - E_L) - w, \quad (1b)$$

where C is the membrane capacitance, g_L the leak conductance, E_L the leak reversal potential, V_T the spike threshold, Δ_T the slope factor, I the (postsynaptic) input current, τ_w the adaptation time constant, and a the subthreshold adaptation. The membrane potential increases rapidly for $V > V_T$ due to the non-linear exponential term, which leads to rapid depolarization and spike generation at time $t = t_{spike}$, where the membrane potential and adaptation variable are updated according to

$$V \rightarrow V_r, \quad (2a)$$

$$w \rightarrow w + b, \quad (2b)$$

where V_r is the reset potential and b is the spike-triggered adaptation.

2.1.2. Dynamic Synapse Model

In the DYNAP-SE, synapses are implemented with sub-threshold Differential Pair Integrator (DPI) log-domain filters proposed by Bartolozzi and Indiveri (2007) and further described by Chicca et al. (2014). The response of the DPI for an input current I_{in} can be approximated with a first-order linear differential equation,

$$\tau\frac{d}{dt}I_{out} + I_{out} = \frac{I_{th}}{I_\tau}I_{in}, \quad (3)$$

where I_{out} is the (postsynaptic) output current, τ and I_τ are time constant parameters, and I_{th} is an additional control parameter that can be used to change the gain of the filter. This approximation is valid in the domain where $I_{in} \gg I_\tau$ and $I_{out} \gg I_{I_{th}}$. The AdEx neuron model and the synapse equation are used in the following to describe the disynaptic delay elements that we configure in the DYNAP-SE in order to approximate the cricket auditory feature detection circuit.

2.2. Cricket Auditory Feature Detection Circuit

We consider the auditory feature detection circuit for sound pattern recognition in the brain of female field crickets, described by Schöneich et al. (2015), which is used for the recognition of the sound pulse pattern of the male calling song. The circuit, consisting of five neurons, responds selectively to a species-specific sound-pulse interval of roughly 20 ms, by using a detection mechanism that relies on the coincidence of a direct neural response and a delayed response to the received sound pulses. In this circuit, a coincidence detecting neuron, LN3, receives excitatory projections along two separate pathways; one directly from the ascending neuron AN1, and the other via the inhibitory neuron LN2 followed by a non-spiking delay neuron LN5, which we approximate here with a delay element formed by an inhibitory–excitatory synapse pair, see **Figure 1** (adapted from Nilsson 2018).

The non-spiking inhibitory neuron, LN5, in the cricket projects to LN3 and provides a delayed excitation of LN3 due to Postinhibitory Rebound (PIR). The duration of the delay matches that of the species-specific sound Interpulse Interval (IPI) of roughly 20 ms that the circuit is specialized for detecting, so that the delayed excitation arrives at the coincidence detecting neuron, LN3, simultaneously with the excitation caused by the subsequent sound pulse. The coincident excitations of LN3 enables it to fire and excite the feature detecting neuron, LN4, which, in turn, signals the feature detection by firing.

2.3. Disynaptic Delay Elements

The PIR of the non-spiking neuron LN5 in the cricket auditory feature detection circuit provides the delayed excitation of LN3 required for feature detection. For a general discussion of such delays, see Buonomano (2000) and Mauk and Buonomano (2004). Spike-based dynamic neuromorphic processors, such as the DYNAP-SE, cannot directly implement non-spiking neurons, such as the LN5 neuron in the cricket circuit, and flexible routing of such analog signals is problematic. Therefore, we approximate LN5 and PIR with an inhibitory–excitatory pair of dynamic synapses with different time constants, so that the sum of the two postsynaptic currents initially is inhibitory and subsequently becomes excitatory some time after presynaptic stimulation. For the inhibitory effect, a synapse of the subtractive type is used in the DYNAP-SE. As its name implies, the subtractive inhibitory synapse type allows for combining excitation and inhibition dynamics by summing inhibitory and excitatory postsynaptic currents, as opposed to the shunting synapse type which inhibits the neuron using a different mechanism. This summation of postsynaptic currents is the central mechanism of the proposed disynaptic delay element. For the excitatory part, the slow synapse type is used, leaving the fast synapse type available for bias configuration and use for stimulation of the postsynaptic neuron; in this case, for the projection from AN1 to LN3.

The proposed disynaptic delay element can be modeled with Equation (3), and the membrane potential resulting from presynaptic stimulation can be illustrated by solving Equation (1). **Figure 2** shows a numerical simulation of the disynaptic delay element model for a 20 ms constant input current that represents the presynaptic stimulation, as in **Figure 1**.

Since the simulated neuron is in the subthreshold regime ($V \ll V_T$), Equation (1) is simplified by setting the exponential term to zero and omitting the adaption variable. The neuron and synapse parameters are selected so that the membrane potential

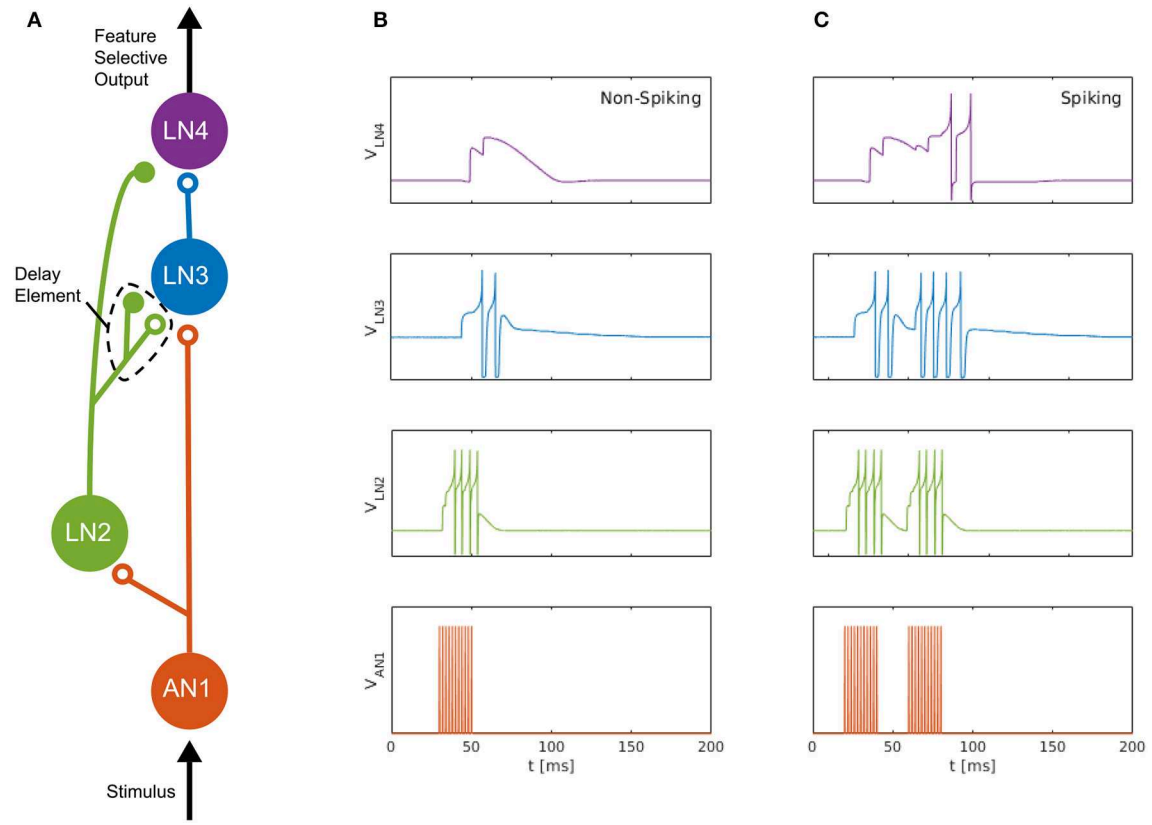


FIGURE 1 | Neuromorphic feature detection circuit inspired by an auditory feature detection circuit in field crickets. **(A)** SNN architecture comprising four spiking neurons, on which open circles and solid disks denote, respectively, excitatory and inhibitory synapses. The disynaptic delay element imitates the dynamics of the non-spiking delay neuron, LN5, in the feature detection circuit of the cricket (Schöneich et al., 2015). **(B)** Measured neuron membrane potentials in the DYNAP-SE, following a 20-ms pulse stimulus. **(C)** Similarly, membrane potentials resulting from a pair of 20-ms stimulus pulses with a 20-ms interval, which causes the feature detecting neuron, LN4, to fire. By overcoming its inhibition and spiking, LN4 signals the feature detection.

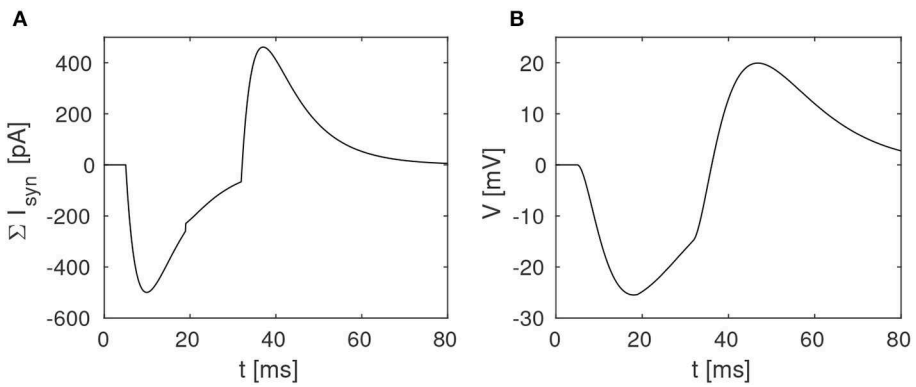


FIGURE 2 | Simulation of the disynaptic delay element model. **(A)** Sum of inhibitory and excitatory postsynaptic currents from the delay element. **(B)** Resulting postsynaptic neuron membrane potential.

is comparable to the potential measured in the hardware, and should thus not be directly compared with biological potentials and threshold values.

Dynamic disynaptic elements of this type are expected to provide a delayed excitation that qualitatively matches the effect of PIR in the output of non-spiking delay neurons like the LN5. Furthermore, we expect that the time delay and relative amplitude of inhibition and excitation can be configured, for example by modifying the synapse time constants and efficacies. The experimental results presented below demonstrate that this

is indeed feasible, and that for some bias settings it is possible to control the time delay and delayed excitation amplitude with the synaptic efficacies only.

2.3.1. Neuromorphic Implementation

The disynaptic delay elements were configured in the DYNAP-SE in two different ways. First, we aimed to mimic the post-inhibitory rebound in the cricket auditory circuit with a delay of about 20 ms. The delay elements were stimulated with four spikes equally spaced over the $\sim$ 20-ms stimulus-response of LN2 for a 20-ms sound pulse, which represents the projection from LN2 to LN5 in the cricket circuit. The time constant of the inhibitory synapse of the delay element was set so that the resulting inhibition of LN3 corresponded to the inhibition caused by LN5 in the cricket; that is, a couple of ms longer than the 20-ms sound-pulse duration. The excitatory synapse was tuned so that the LN3 excitation lasts somewhat longer than that of the initial inhibition, approximately to the end of the corresponding PIR excitation of LN5 in the cricket. The weight of the inhibitory synapse was set higher than that of the excitatory synapse, such that the sum of inhibition and excitation turned out negative, thus inhibiting the neuron for the duration of the delay. For the excitatory synapse, the weight was set to yield a substantial excitation of the postsynaptic neuron following the inhibition, while not generating spikes without additional synaptic stimulation. In this manner, the effect of the non-spiking LN5 on LN3 is imitated with the summation of an inhibitory postsynaptic current and an excitatory postsynaptic current produced by two synapses on LN3. The resulting DYNAP-SE bias values are found in **Table 1**.

TABLE 1 | Bias parameter values used for the characterization of individual disynaptic delay elements in the DYNAP-SE.

Parameter type	Parameter name	Coarse value	Fine value	Current level
Neuronal	IF_AHTAU_N	7	35	L
	IF_AHTHR_N	7	1	H
	IF_AHW_P	7	1	H
	IF_BUF_P	3	80	H
	IF_CASC_N	7	1	H
	IF_DC_P	0	40	H
	IF_NMDA_N	1	213	H
	IF_RFR_N	4	40	H
	IF_TAU1_N	5	39	L
	IF_TAU2_N	0	15	H
	IF_THR_N	6	4	H
Synaptic	NPDPIE_TAU_S_P	6	120	H
	NPDPIE_THR_S_P	1	30	H
	NPDPII_TAU_F_P	5	100	H
	NPDPII_THR_F_P	3	60	H
	PS_WEIGHT_EXC_S_N	1	110	H
	PS_WEIGHT_INH_F_N	1	130	H
	PULSE_PWLK_P	5	40	H
	R2R_P	4	85	H

Given the large parameter space of a dynamic neuromorphic processor like the DYNAP-SE, we then explored different ways to simplify the configuration of the disynaptic delay elements for delays up to about 100 ms. One identified possibility is to lower the constant injection current of the neurons receiving the delayed signal, to such an extent that the inhibition by the delay elements make the neuron reach its minimum membrane potential. This results in delay elements for which the duration of inhibition, τ_{inh}, can be controlled with the inhibitory weight of the delay element, w_{inh}. Furthermore, the amplitude of the post-inhibitory excitation, V_{max}, is then controlled by the excitatory weight of the delay element, w_{exc}, as well as by varying the number of presynaptic spikes stimulating the delay element. The DYNAP-SE bias values for this configuration of the delay elements are found in **Table 2**.

2.3.2. Characterization

For the purpose of characterization, the proposed disynaptic delay elements were implemented, in parallel, in one core of a DYNAP-SE neuromorphic processor; one delay element on each of the 256 neurons in the core. All of these neurons were then stimulated as described in section 2.3.1, and their membrane potentials were measured with an oscilloscope. To avoid oscilloscope and DYNAP-SE time synchronization issues, we analyzed the membrane potential measurements without reference to the precise timing of the presynaptic stimulation. The full duration at half minimum of the inhibition and the full duration at half maximum of the subsequent excitation, see **Figure 2**, can be determined from membrane potential measurements without reference to the timing of presynaptic spikes. Thus, we define the timescales of inhibition and delayed excitation in terms of the Full Duration at Half Maximum/Minimum (FDHM). We characterized the disynaptic delay elements with the distributions of the following five quantities: the minimum membrane potential, V_{min}, the maximum membrane potential, V_{max}, the FDHM of inhibition, τ_{inh}, the FDHM of excitation, τ_{exc}, and the time duration from the FDHM onset of the inhibition to the FDHM onset of the excitation, τ_{delay}. These quantities are illustrated in **Figure 3**, and allowed us to investigate the effect of different bias parameter settings on the disynaptic delay elements in a population of neurons in the DYNAP-SE. This way the bias parameter values of the delay elements could for example be tuned to imitate the behavior of the delay neuron LN5 in the cricket. Further details on the experimental settings are described in section 2.5.

2.4. Neuromorphic Feature Detection Circuits

2.4.1. Cricket Circuit

For the implementation of the cricket auditory feature detection circuit, as described in section 2.2, in the DYNAP-SE neuromorphic processor, stimuli representing the projections from AN1 upon auditory stimulation were generated in the form of 11 spikes evenly distributed over the pulse duration of 20 ms (in the noise-free case), yielding 10 Interspike Intervals (ISIs) of 2 ms each. Each of the remaining three neurons of the circuit, see **Figure 1**, were modeled on separate cores in one chip of the DYNAP-SE. The DYNAP-SE bias parameter values

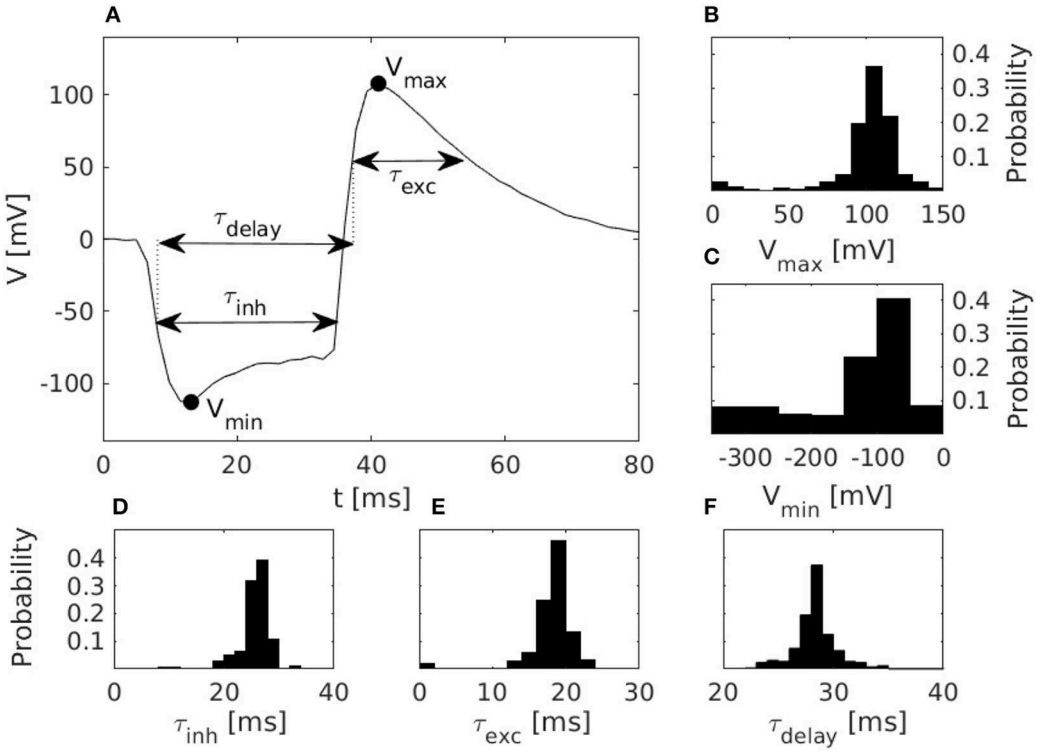


FIGURE 3 | Characteristics of disynaptic delay elements configured in the DYNAP-SE neuromorphic processor. **(A)** Postsynaptic membrane potential vs. time, illustrating the delayed excitation resulting from a presynaptic pulse. **(B)** Distribution of the maximum measured membrane potential, V_{max}, resulting from a presynaptic pulse. **(C)** Similarly, the distribution of the minimum measured membrane potential, V_{min}. **(D)** Distribution of the inhibitory timescale, τ_{inh}, defined as the full width at half minimum. **(E)** Distribution of the excitatory timescale, τ_{exc}, defined as the full width at half maximum. **(F)** Distribution of the delay time, τ_{delay}, defined as the duration from the onset of τ_{inh} to the offset of τ_{exc}. The distributions in panels **(B–F)** were obtained via characterization of one DYNAP-SE core, comprising, in parallel, one disynaptic delay element on each of the 256 neurons, with biases configured according to **Table 1**.

TABLE 2 | Bias parameter values used for configuration of the disynaptic delay elements in the DYNAP-SE.

Parameter type	Parameter name	Coarse value	Fine value	Current level
Neuronal	IF_DC_P	1	30	H
Synaptic	NPDPIE_TAU_S_P	7	210	H
	NPDPIE_THR_S_P	1	30	H
	NPDPII_TAU_F_P	6	80	H
	NPDPII_THR_F_P	3	60	H
	PS_WEIGHT_EXC_S_N	0	8–80	H
	PS_WEIGHT_INH_F_N	0	1–150	H
	PULSE_PWLK_P	5	40	H
	R2R_P	4	85	H

*Neuronal parameters not defined in this table were set according to **Table 1**.*

TABLE 3 | Bias parameter values used for the inhibitory neuron, LN2, in the DYNAP-SE implementation of the cricket feature detection network.

Parameter type	Parameter name	Coarse value	Fine value	Current level
Neuronal	IF_AHTAU_N	7	35	L
	IF_AHTHR_N	7	1	H
	IF_AHW_P	7	1	H
	IF_BUF_P	3	80	H
	IF_CASC_N	7	1	H
	IF_DC_P	7	2	H
	IF_NMDA_N	7	1	H
	IF_RFR_N	4	208	H
	IF_TAU1_N	6	21	L
	IF_TAU2_N	5	15	H
	IF_THR_N	3	20	H
Synaptic	NPDPIE_TAU_F_P	5	165	H
	NPDPIE_THR_F_P	1	100	H
	PS_WEIGHT_EXC_F_N	0	190	H
	PULSE_PWLK_P	0	43	H
	R2R_P	4	85	H

for the neurons LN2, LN3, and LN4 are found in **Tables 3**–**5**, respectively, and the neuromorphic implementations of these neurons are described in the following.

For the implementation of the inhibitory neuron, LN2, a single neuron on a reserved core was used. This neuron was

TABLE 4 | Bias parameter values used for the coincidence detecting neuron, LN3, in the DYNAP-SE implementation of the cricket feature detection network.

Parameter type	Parameter name	Coarse value	Fine value	Current level
Synaptic	NPDPIE_TAU_F_P	5	200	H
	NPDPIE_TAU_S_P	6	120	H
	NPDPIE_THR_F_P	1	30	H
	NPDPIE_THR_S_P	1	30	H
	NPDPII_TAU_F_P	5	100	H
	NPDPII_THR_F_P	3	60	H
	PS_WEIGHT_EXC_F_N	1	144–161	H
	PS_WEIGHT_EXC_S_N	1	110	H
	PS_WEIGHT_INH_F_N	1	130	H
	PULSE_PWLK_P	5	40	H
	R2R_P	4	85	H

Neuronal parameters set according to ***Table 1****.*

TABLE 5 | Bias parameter values used for the feature detecting neuron, LN4, in the DYNAP-SE implementation of the cricket feature detection network.

Parameter type	Parameter name	Coarse value	Fine value	Current level
Synaptic	NPDPIE_TAU_F_P	5	80	H
	NPDPIE_THR_F_P	1	140	H
	NPDPII_TAU_F_P	6	180	H
	NPDPII_THR_F_P	3	140	H
	PS_WEIGHT_EXC_F_N	0	71–82	H
	PS_WEIGHT_INH_F_N	0	60	H
	PULSE_PWLK_P	0	43	H
	R2R_P	4	85	H

Neuronal parameters set according to ***Table 3****.*

set to receive the generated stimulation representing AN1 by assigning a synaptic connection of the fast excitatory type. The bias parameter values from section 5.7.3 in the DYNAP-SE user guide [1] were used as reference. The parameter values of the fast excitatory synapse were then adjusted in order to model the behavior of LN2 as observed in the cricket. The synaptic time constant, NPDPIE_TAU_F_P, was adjusted to match that of the cricket, and the synaptic weight, PS_WEIGHT_EXC_F_N, and threshold parameter, NPDPIE_THR_F_P, were adjusted for LN2 to respond with the right amount of four to five spikes for each input pulse.

For the coincidence detecting neuron, LN3, the proposed delay elements were implemented according to the earlier description. An excitatory connection of the fast type was added for LN3 to receive the projection from AN1.

For the excitatory connection from LN3 to the feature detecting neuron LN4, a synapse of the fast type was used, and, for the inhibitory connection from LN2 to LN4, a synapse of the subtractive type was used. Bias parameter values from section 5.7.3 in the DYNAP-SE user guide were used for neuronal parameters, and as reference values for the fast excitatory synapses. For the fast inhibitory synapse, bias values from section 5.7.5 in the user guide were used as reference. The bias parameters, time constant, threshold and weight, for both synapse types, were then hand-tuned in order to approximate the behavior of LN4 in one DYNAP-SE neuron, so as to make LN4 spike, thus signaling feature detection, for stimuli with IPIs of 20 ms, but not for IPIs of 0, 10, 30, 40, and 50 ms.

2.4.2. Single-Neuron Feature Detector

We further investigated the possibility that a single neuron in the DYNAP-SE with multiple disynaptic delay elements can respond selectively to spatiotemporal spike patterns, which match the difference in the delay times resulting from device mismatch. Specifically, we configured a neuron with two inputs via two different disynaptic delay elements. The input patterns consist of spike pairs, one spike for each delay element, with a variable spike-time interval. Patterns with spike-time intervals that match the delay-time difference between the two delay elements should generate postsynaptic currents with coincident maxima, thus resulting in maximum excitation of the neuron.

The neuron and delay elements were configured as described in section 2.3.1 with bias parameter values according to **Table 2**, with a few modifications: The threshold, $\texttt{IF\_THR\_N} = (6, 135)$, and excitatory synaptic efficacy was modified so that the neuron generates output spikes for two input spikes, and the inhibitory weight of the delay elements was modified accordingly. Furthermore, the time-constant of the excitatory synapse was lowered to compensate for the strong excitation required, $\texttt{NPDPIE\_TAU\_S\_P} = (5, 70)$ and $\texttt{NPDPIE\_THR\_S\_P} = (0, 210)$. The numbers in parentheses denote coarse and fine parameter values of the DYNAP-SE, respectively.

The synapses were selected with an off-line Hebbian-like learning rule such that, for the spike patterns considered, the neuron responded selectively to spike patterns with intermediately long intervals, but not to spike patterns with shorter or longer intervals. Spike patterns were generated as described in the next section, and the neuron was stimulated one hundred times with each pattern. Based on these experiments the average probability of the neuron to spike for each type of pattern was determined.

2.5. Experiments

In all of the experiments conducted in this work, the DYNAP-SE neuromorphic processor was controlled using the cAER event-based processing framework for neuromorphic devices. More specifically, a custom module making use of the tools for configuration and monitoring provided by cAER was created and added to the framework. All stimuli were synthetically generated using the built-in spike generator in the FPGA of the DYNAP-SE, which generates spike-events according to assigned ISIs and virtual source-neuron addresses.

The DYNAP-SE features analog ports for monitoring of neuron membrane potentials. For measurements of these potentials, the 8-bit USB oscilloscope SmartScope by LabNation was used. Since these measurements only capture the neuron

[1] https://aictx.ai/technology/

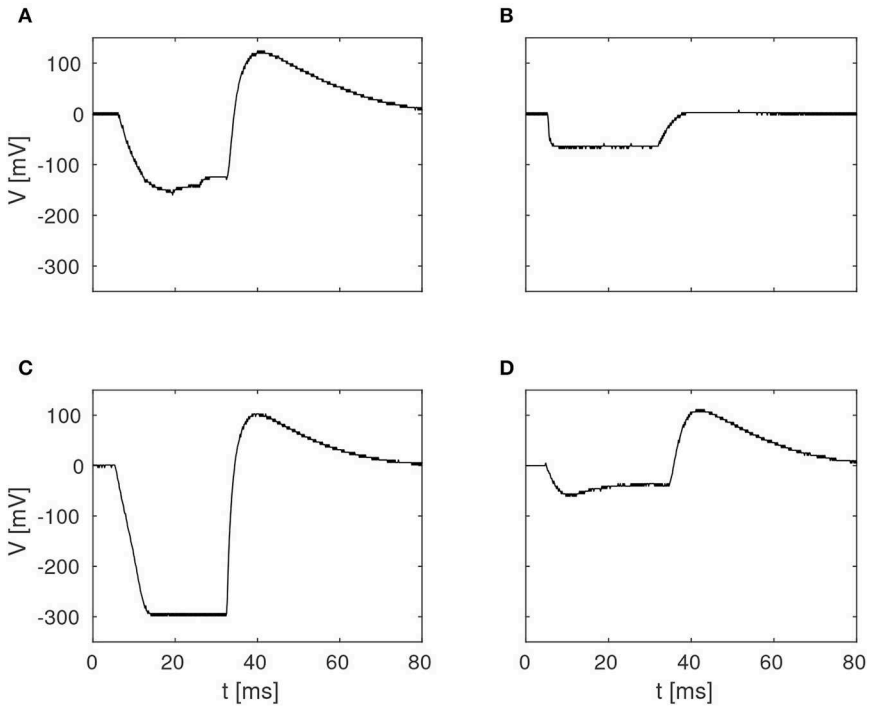


FIGURE 4 | Examples of four different membrane potentials measured in the characterization of the delay elements summarized in **Figure 3**. These variations were observed in one core with 256 neurons, with biases configured according to **Table 1**.

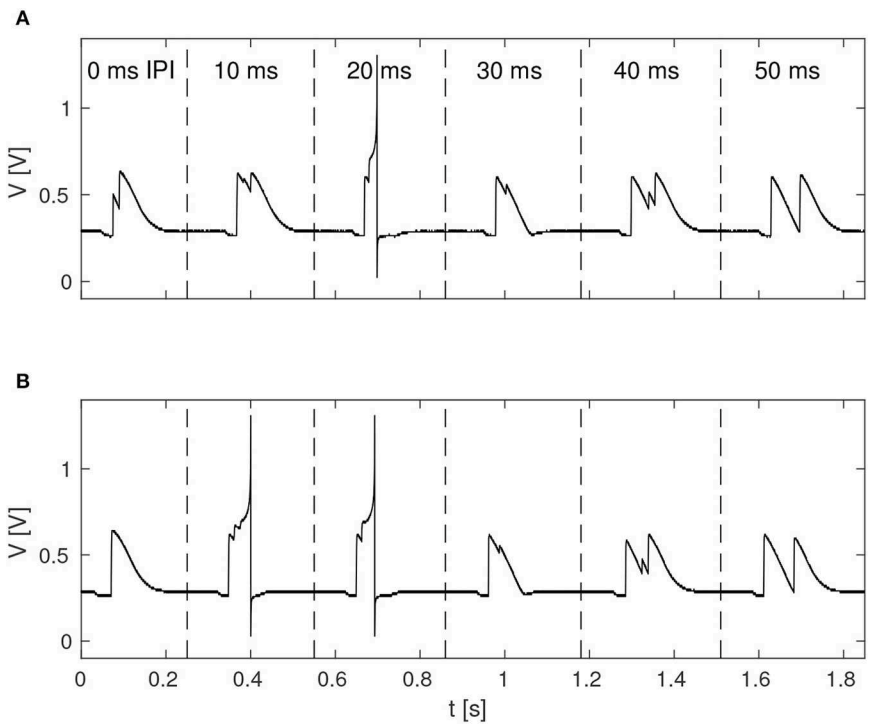


FIGURE 5 | Response of LN4 for double-pulse stimuli with IPIs of 0, 10, 20, 30, 40, and 50 ms, respectively. **(A)** Noiseless case. **(B)** Example for 20% noise, with a false positive for the 10-ms IPI.

membrane potential, there is no information about the precise relative timing of spike-events in the resulting data. Because of this, the durations of inhibition and excitation of the delay elements were defined in terms of the FDHM as described above.

For the extraction of the delay parameters defined in section 2.3.1, the stimulus was repeatedly broadcast to all neurons in the core, and for each stimulation cycle one neuron was monitored with the oscilloscope using the programmable analog outputs of the DYNAP-SE. The stimulation cycle was given a duration of 0.5 s, in order for the neurons to relax to a resting state before and after stimulation. At the initial state of rest, the resting potential was automatically estimated for each neuron. The resting potential was subsequently subtracted from the measurement data, such that the resulting resting potentials are zero. This was done to make the parameter values of the different neurons comparable with each other.

3. RESULTS

3.1. Characteristics of Delay Elements

Results from the characterization of the disynaptic delay elements, implemented in parallel on each of the 256 neurons in one core of the DYNAP-SE neuromorphic processor, are presented in **Figure 3**.

The figure shows the pulse-response of one typical delay element from the resulting population, along with histograms of the distributions of parameters that characterize each delay element. The resulting values of V_{max} range from 3 to 143 mV and center around 105 mV. V_{min} has a thicker tail of the distribution and range from −310 to −20 mV, with most values between −100 and −50 mV. The time constant distributions have relatively thin tails. τ_{inh} has values between 6 and 47 ms with probability peaking between 26 and 28 ms. τ_{exc} ranges from 0 to 38 ms with probability peaking between 18 and 20 ms, and τ_{delay} spans between 22 and 51 ms with probability peaking between 28 and 29 ms.

The pulse-responses of four different delay elements are presented in **Figure 4**, which illustrates the variety of delay dynamics obtained thanks to device mismatch. Here, the variance of the minimum voltage, V_{min}, is especially evident, but variation in other parameters can also be observed, such as V_{max}, in the case of the virtually non-existing excitation in **Figure 4B**.

3.2. Cricket Feature Detection

The function of the neuromorphic implementation of the feature detection SNN was investigated by stimulating it with double

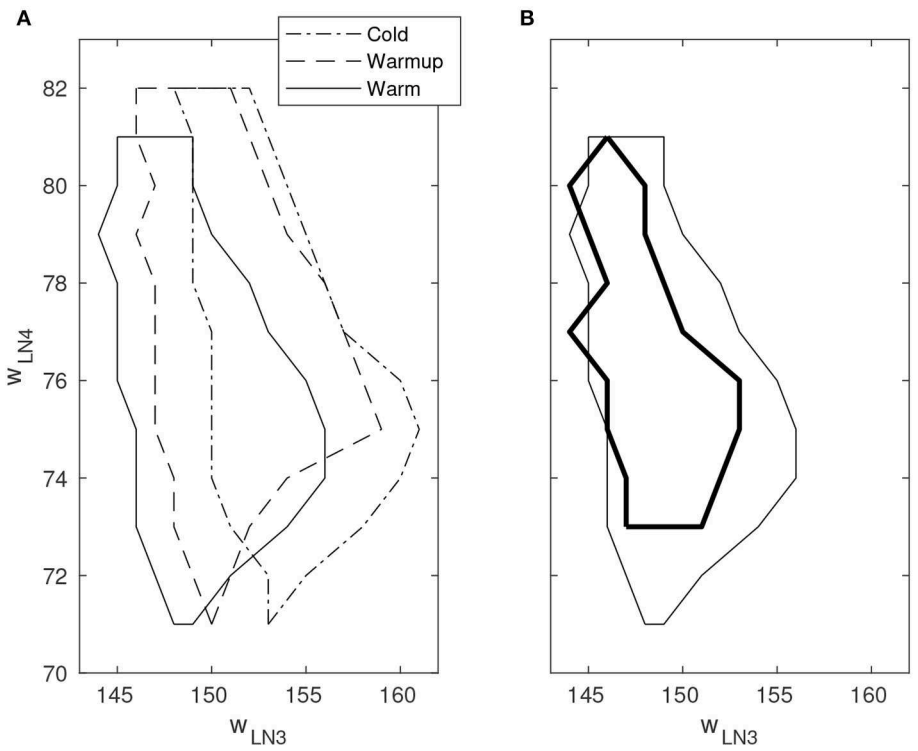


FIGURE 6 | Boundary of correct stimulus classification in synaptic parameter space. Outside the enclosed region, false positives and/or false negatives occur with varying probability. The horizontal and vertical axes indicate the fine integer bias-values of the excitatory synaptic weight for the neurons LN3 and LN4, respectively. Multiple line types indicate experiments performed under different environmental conditions. **(A)** Movement of the classification boundary observed after several hours of continuous operation from cold startup. The temperature change is likely caused by the FPGA that is enclosed in the system. **(B)** Shrinkage of the classification boundary in presence of 10% spike-timing noise in the stimulus (bold line). Boundary points are temperature dependent.

pulses of 20 ms duration each, while increasing the IPI from 0, 10, 20, 30, 40, to 50 ms. Furthermore, in order to investigate the effect of noise in the stimuli, as is likely to be present in real-world environments, different levels of spike-timing noise was introduced in the generated stimuli by randomly perturbing the value of the ISIs with values drawn from a continuous uniform distribution. **Figure 5** shows the membrane potential of LN4 during correct classification of noiseless double pulses of all of the IPIs mentioned above, as well as the result in the presence of 20% spike-timing noise, where some false positives are observed for the 10 ms IPI.

By varying the weights of the excitatory projection from AN1 to LN3 and the excitatory synaptic weight of LN4, respectively, a boundary of correct classification of stimuli could be identified in the space spanned by these two parameters. Outside the boundary, false positives and/or false negatives occur with varying probability. The boundary was observed to move substantially in the parameter space as time progressed after cold startup of the DYNAP-SE and this is likely due to heating by the FPGA that is enclosed in the DYNAP-SE system. This change was observed over multiple runs of the experiment and appears to be qualitatively consistent. Furthermore, the shift of the boundary in the presence of spike-timing noise in the stimuli was investigated. **Figure 6** shows the boundary of correct classification, as measured at three separate points in time after device initialization, spanning from minutes to several hours of run-time. The figure also shows the shrinkage of the classification boundary in the presence of 10% spike-timing noise in the stimuli, in relation to the steady-state of the boundary after several hours of system run-time.

A quantitative investigation of the IPI dependence of the feature detection circuit was made by repeatedly stimulating the network with double pulses of different IPIs as described earlier, while observing the response in LN3 and LN4 by recording and counting the spikes of both neurons. For each IPI, the network was presented with the corresponding double-pulse stimulus 50 times. **Figure 7** shows, in the case of noiseless stimuli, the average number of spikes from LN3 and LN4, respectively, centrally within the synaptic boundary of correct classification, as well as at the boundary. Centrally within the boundary of correct classification, LN4 responded exclusively to the 20 ms pulse interval, with no false positives or negatives. On the boundary of the parameter space, LN4 began to exhibit false positives for the 10 ms IPI, with 0.32 ± 0.47 spikes per double-pulse stimulus.

Similarly, **Figure 8** shows the results for the best synaptic configuration used in the previous experiment, centrally located within the boundary of correct classification, but for different levels of spike-timing noise. As expected the network performed correct classification in the noiseless case. The introduction of noise caused LN4 to exhibit false positives, in particular for the

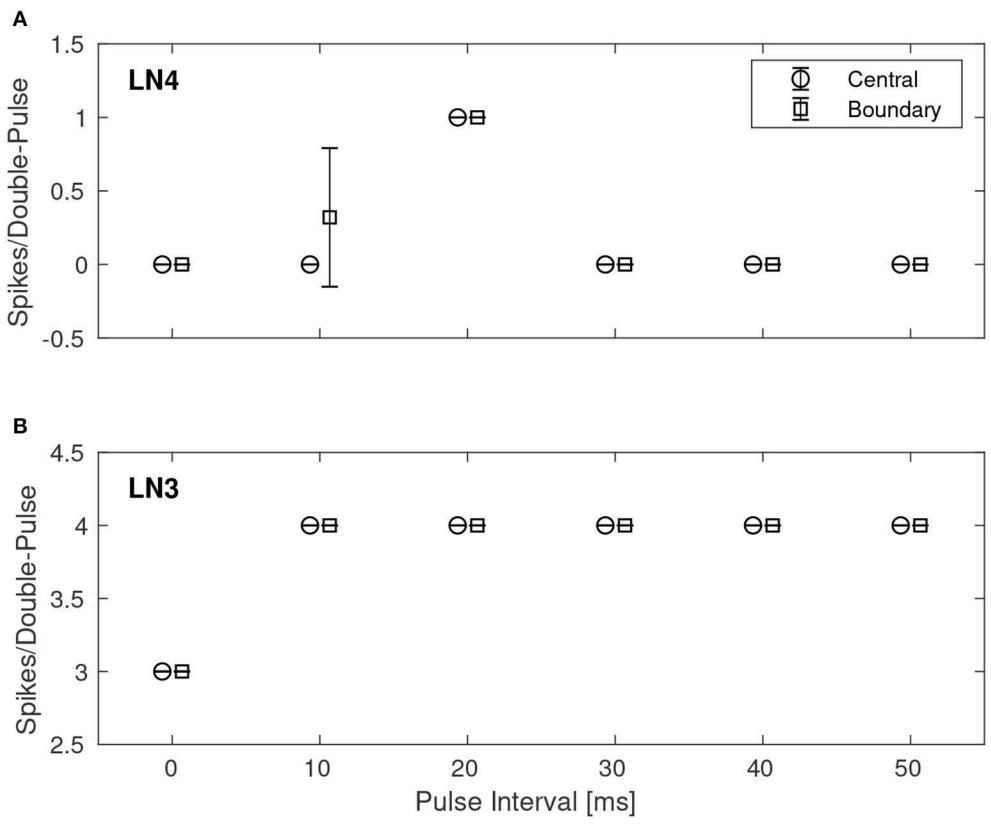


FIGURE 7 | Average number of spikes from LN3 and LN4 per double-pulse stimulus for varying IPIs and two different bias configurations: one central to, and one on the boundary of, the region illustrated in **Figure 6**. For each IPI, the data-points are graphically separated by 4/3 ms to improve clarity of the visualization. Error bars denote ±1 standard deviation. **(A)** Feature detecting neuron, LN4. **(B)** Coincidence detecting neuron, LN3.

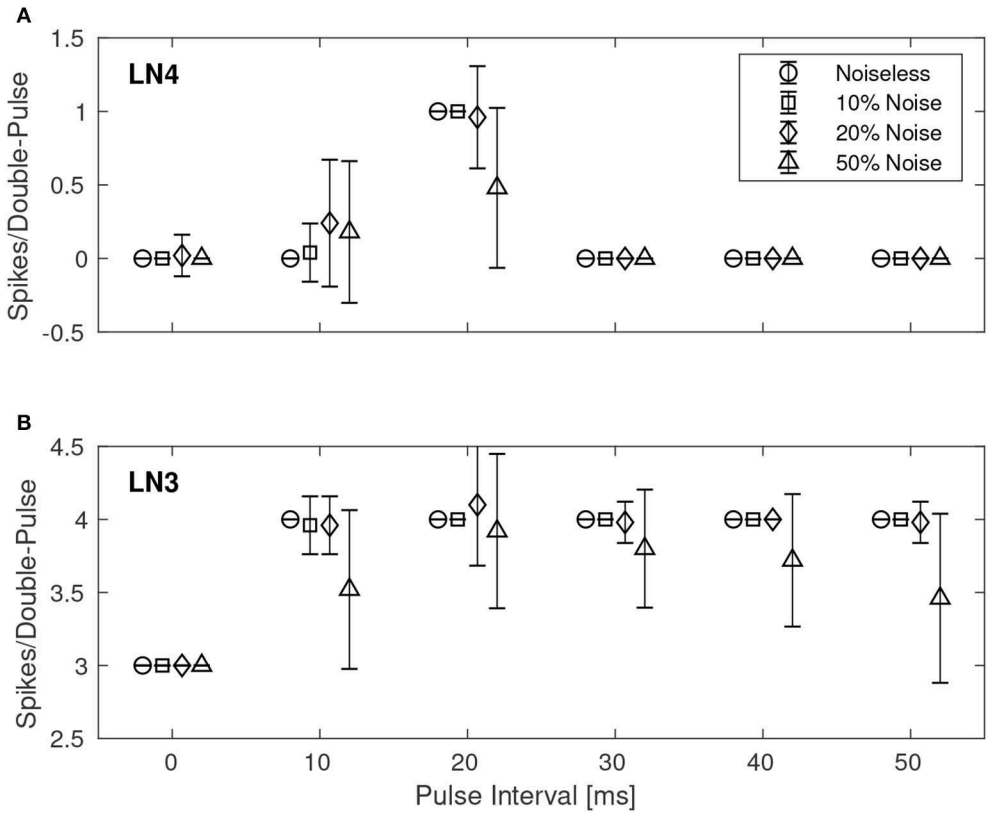


FIGURE 8 | Average number of spikes from LN3 and LN4 per double-pulse stimulus for varying IPIs and different levels of spike-timing noise in the stimuli. For each IPI, the data-points are graphically separated by 4/3 ms to improve the clarity of the visualization. Error bars denote ±1 standard deviation. **(A)** Feature detecting neuron, LN4. **(B)** Coincidence detecting neuron, LN3.

10 ms IPI. At higher levels of noise also false negatives were observed. In the case of 50% noise the response of LN4 was 0.18 $\pm$ 0.48 spikes per double-pulse for the 10 ms IPI, and 0.48 $\pm$ 0.54 spikes for the 20 ms IPI.

3.3. Reconfigurability of Delay Elements

Given the large parameter space of a dynamic neuromorphic processor, such as the DYNAP-SE, we explored different ways to simplify the configuration of the disynaptic delay elements for delays up to about 100 ms. **Figure 9A** shows four configurations of one delay element, with the maximum membrane potential of the post-inhibitory excitation ranging between 20 and 110 mV, and the durations of inhibition ranging between 50 and 90 ms, according to the FDHM definition.

A table with delay element weight values and resulting values of τ_{inh} and V_{max}, from a total of 12 such variations, is presented in **Figure 9B**; the data-points corresponding to the membrane potentials in **Figure 9A** are marked with filled disks.

3.4. Feature Detection With Multiple Delay Elements

Disynaptic delay elements produce variable delayed excitations when stimulated with presynaptic spikes, as demonstrated in **Figure 9**. Furthermore, the delayed excitations are subject to device mismatch variability, as demonstrated in **Figure 3**. Thus, as described in section 2.4.2 we investigated the possibility that a single neuron with multiple disynaptic delay elements can respond selectively to spatiotemporal patterns that match the different delay times. We find that this is possible, and one example is illustrated in **Figure 10**, which shows the results for one neuron in DYNAP-SE with two delay elements (DE1 and DE2) stimulated with eleven different spatiotemporal patterns.

The experiment with each pattern is repeated one hundred times. The neuron fires selectively when the time interval between presynaptic spikes, $t_{DE2} - t_{DE1}$, is 3 to 4 ms, while the probability of firing is low for shorter and longer presynaptic spike intervals. The neuron does not fire when $t_{DE2} - t_{DE1} < 0$.

4. DISCUSSION

SNN architectures for temporal pattern recognition require delays, and the dynamics of synapses, dendrites and axons of cortical neurons correspond to a spectrum of signal propagation delays ranging up to about 100 ms. In this work, we investigate delays produced by inhibitory–excitatory pairs of conventional conductance-based dynamic synapses implemented in the DYNAP-SE neuromorphic processor. Our main results presented in **Figures 3**, **9**, **10** demonstrates that configurable delayed

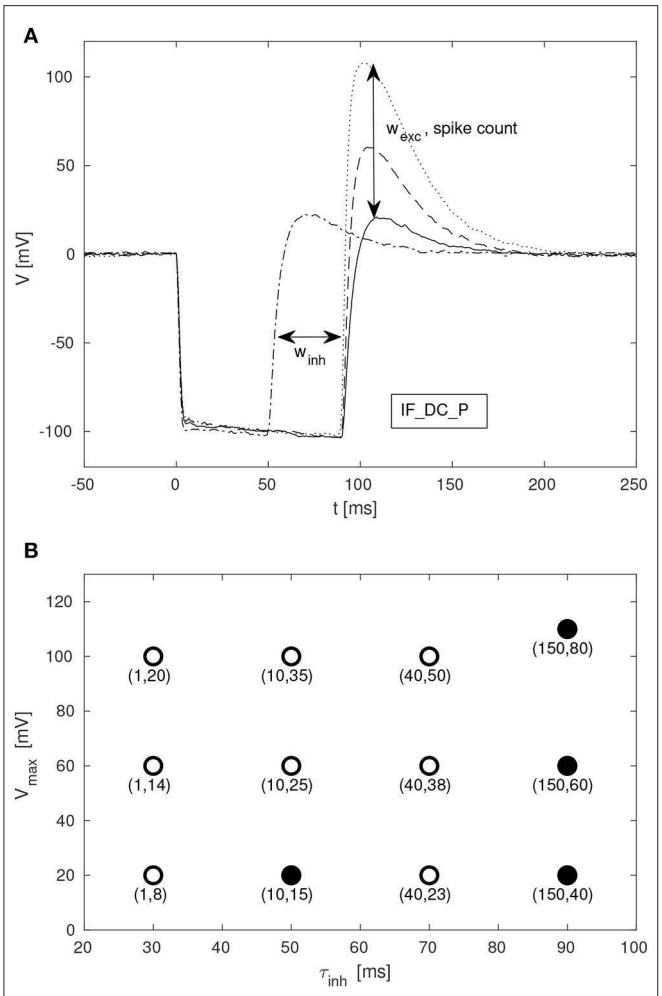


FIGURE 9 | Configuration of disynaptic delay elements. **(A)** Postsynaptic membrane potential vs. time, resulting from a presynaptic pulse. The delay is controlled mainly by the inhibitory synaptic efficacy, w_{inh}. The amplitude of the delayed excitation is controlled mainly by the excitatory synaptic efficacy, w_{exc}, and by the number of presynaptic spikes. Note that the membrane potential reaches its minimum possible value during inhibition, and that the difference between this value and the resting potential is controlled with the constant injection current of the neuron, controlled by the bias parameter `IF_DC_P`. **(B)** Maximum membrane potential, V_{max}, vs. duration of inhibition, τ_{inh}, for different values of (w_{inh}, w_{exc}). Each point is denoted with the corresponding fine integer bias values of the inhibitory and excitatory synaptic weights, respectively.

excitations of up to about 100 ms can be implemented in this way, and that a single neuron with multiple disynaptic delay elements can respond selectively to spatiotemporal input patterns. **Figure 3** illustrates that for one particular configuration of the disynaptic elements, which is selected to mimic the PIR of a particular non-spiking delay neuron in crickets, a distribution of delays are realized in one neuromorphic core thanks to device mismatch. Furthermore, **Figure 9B** illustrates a subset of the possible disynaptic configurations resulting in different delays ($\tau_{inh} = 30, 50, 70, 90$ ms) and delayed excitation amplitudes. Thus, by configuring the two synaptic parameters of the disynaptic elements, variable excitation strengths and delays of up to about $\tau_{delay} \simeq 100$ ms are achieved, which is similar to the range of dendritic and axonal signal propagation delays in cortical circuits (Dayan and Abbott, 2005).

At the quantitative level, we observe some differences between the feature detection results presented in section 3.2 and the behavior of the cricket circuit described by Schöneich et al. (2015). In the crickets, the response of the coincidence detector neuron LN3 for different IPIs varies so that the distribution of the number of spikes of LN3 increases as the interval gets closer to the species-specific IPI of 20 ms. This is not the case in the results presented here, and further optimization of the neuron and synapse parameters are required if this behavior is to be imitated. As illustrated in **Figure 7B**, our LN3 reliably produces the same number (but different timings) of spikes for all of the different IPIs, with the exception of the 0 ms IPI. A more plausible trend is observed in the case of 50% input noise, but in that case the classification results are weaker. Hence, the classification mechanism relies on the timing of spikes and the balance of inhibition and excitation.

Temporal feature detection and pattern recognition are central tasks in advanced sensor and perception systems. Thus, low-power SNN processors enabling learning and recognition of complex spatiotemporal patterns (Indiveri and Sandamirskaya, 2019; Strukov et al., 2019) have many potential applications, for example for always-on machine monitoring (Martin del Campo et al., 2013; Martin del Campo and Sandin, 2017), where the system needs to operate autonomously and wirelessly with limited resources over the expected lifetime of the monitored machine component (Martin del Campo, 2017; Häggström, 2018). Although we sidestep Dale's principle, the dynamic disynaptic delay elements investigated here have the desirable property that each neuron can be configured with multiple disynaptic elements, as illustrated in **Figure 10**. By combining multiple disynaptic delay elements, for example in line with the idea of polychronous networks (Izhikevich, 2006), more complex spatiotemporal patterns can be detected in principle. Since the disynaptic delay elements are realized with ordinary dynamic synapses, the approach is not limited to this particular neuromorphic processor, although the distribution of delays obtained is processor and device-mismatch dependent.

Further work is required to investigate how the repertoire of synaptic delays can be exploited and configured/learned to solve practical pattern recognition tasks, and to further develop the understanding of how device mismatch, noise and temperature variations affect different network architectures. With dynamic synapses featuring short- and long-term plasticity, additional mechanisms for sequence detection and learning can also be realized (Buonomano, 2000) and investigated. Furthermore, SNNs can faithfully reproduce dynamics of brain networks, which appear to self-organize near a critical point where no privileged spatial or temporal scale exist, which has interesting consequences for information processes (Cocchi et al., 2017). Thus, Neuromorphic Engineering (Indiveri and Horiuchi, 2011; Strukov et al., 2019) and dynamic neuromorphic processors opens the way to new interesting architectures for pattern recognition and generation in machine perception and control.

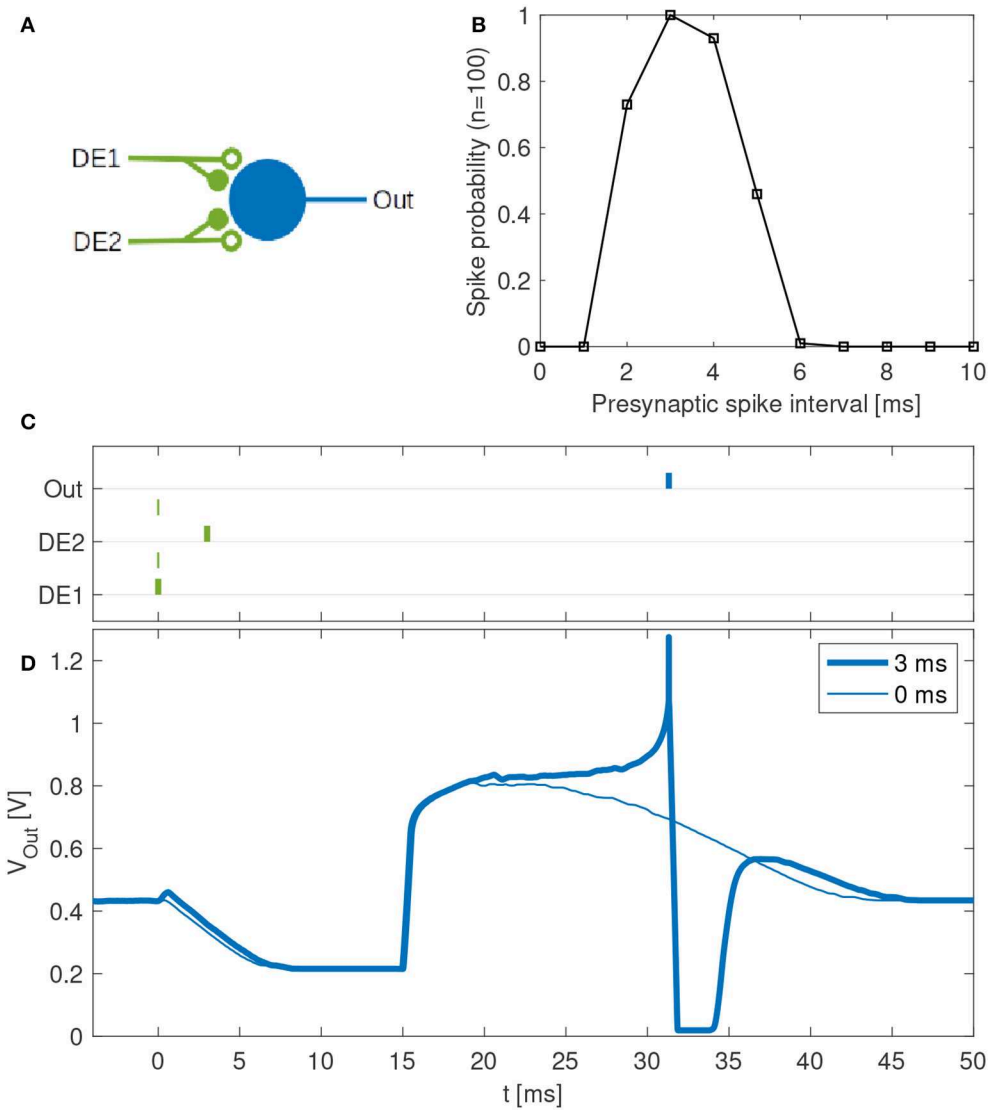


FIGURE 10 | Feature detection by a single neuron in the DYNAP-SE. **(A)** Neuron with one output (Out) and two inputs with disynaptic delay elements (DE1 and DE2). **(B)** Probability that the neuron spikes vs. the presynaptic spike interval, which denotes the time between two presynaptic spikes at DE1 and DE2, respectively. This neuron spikes with maximum probability when a spike arrives to DE2 about 3 ms later than to DE1. The neuron does not spike for presynaptic spike intervals below about 2 ms and above about 6 ms. **(C)** Examples of spike times for presynaptic spike intervals of 3 ms (bold lines) and 0 ms (thin lines). In the latter case no postsynaptic spike is generated. **(D)** Examples of membrane potentials measured for 3 ms (bold line) and 0 ms (thin line) presynaptic spike intervals. No spike is generated when the two presynaptic spikes arrive simultaneously. With a presynaptic spike interval of 3 ms the neuron spikes reliably.

DATA AVAILABILITY STATEMENT

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

AUTHOR CONTRIBUTIONS

FS conceived the possibility to imitate non-spiking PIR delay in the DYNAP-SE with synaptic dynamics, supervised the experiments to be carried out, and wrote part of the manuscript. MN implemented the code that controls the DYNAP-SE, performed the experiments, and wrote part of the manuscript.

FUNDING

This work was supported by The Kempe Foundations under contract JCK-1809 and SMK-1429, and was enabled by a collaboration with the Institute of Neuroinformatics in Zurich supported by STINT under contract IG2011-2025.

ACKNOWLEDGMENTS

We thank the reviewers for constructive criticism that helped us improve the manuscript. Ideas leading to the work presented here have been discussed at the CapoCaccia Neuromorphic Engineering Workshop, in particular with Giacomo Indiveri, and the bias parameters of the delay elements presented in **Table 1** are based on bias parameter values kindly shared by Nicoletta Risi. We thank Federico Corradi and Carsten Nielsen for technical support with the DYNAP-SE neuromorphic system. We thank Jerker Delsing for supporting the work in this area at EISLAB and Jonas Ekman for support at the departmental level.

REFERENCES

Agmon-Snir, H., and Segev, I. (1993). Signal delay and input synchronization in passive dendritic structures. *J. Neurophysiol.* 70, 2066–2085. doi: 10.1152/jn.1993.70.5.2066

Bartolozzi, C., and Indiveri, G. (2007). Synaptic dynamics in analog vlsi. *Neural Comput.* 19, 2581–2603. doi: 10.1162/neco.2007.19.10.2581

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

Buonomano, D. V. (2000). Decoding temporal information: a model based on short-term synaptic plasticity. *J. Neurosci.* 20, 1129–1141. doi: 10.1523/JNEUROSCI.20-03-01129.2000

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

Coath, M., Mill, R., Denham, S. L., and Wennekers, T. (2011). "Emergent feature sensitivity in a model of the auditory thalamocortical system," in *From Brains to Systems*, eds C. Hernández, R. Sanz, J. Gómez-Ramirez, L. S. Smith, A. Hussain, A. Chella, and I. Aleksander (New York, NY: Springer), 7–17.

Coath, M., Sheik, S., Chicca, E., Indiveri, G., Denham, S., and Wennekers, T. (2014). A robust sound perception model suitable for neuromorphic implementation. *Front. Neurosci.* 7:278. doi: 10.3389/fnins.2013.00278

Cocchi, L., Gollo, L. L., Zalesky, A., and Breakspear, M. (2017). Criticality in the brain: a synthesis of neurobiology, models and cognition. *Prog. Neurobiol.* 158, 132–152. doi: 10.1016/j.pneurobio.2017.07.002

Dalgaty, T., Vianello, E., De Salvo, B., and Casas, J. (2018). Insect-inspired neuromorphic computing. *Curr. Opin. Insect Sci.* 30, 59–66. doi: 10.1016/j.cois.2018.09.006

Dayan, P., and Abbott, L. F. (2005). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA: The MIT Press.

Delbruck, T., Berner, R., Lichtsteiner, P., and Dualibe, C. (2010). "32-bit configurable bias current generator with sub-off-current capability," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* (Paris), 1647–1650.

Häggström, F. (2018). *Robust energy management for IoT machine elements* (Ph.D. Thesis). Luleå University of Technology, Embedded Intelligent Systems Lab, Luleå, Sweden.

Hussain, S., Liu, S.-C., and Basu, A. (2015). Hardware-amenable structural learning for spike-based pattern classification using a simple model of active dendrites. *Neural Comput.* 27, 845–897. doi: 10.1162/NECO_a_00713

Indiveri, G., and Horiuchi, T. (2011). Frontiers in neuromorphic engineering. *Front. Neurosci.* 5:118. doi: 10.3389/fnins.2011.00118

Indiveri, G., Linares-Barranco, B., Hamilton, T., van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Indiveri, G., and Liu, S. (2015). Memory and information processing in neuromorphic systems. *Proc. IEEE* 103, 1379–1397. doi: 10.1109/JPROC.2015.2444094

Indiveri, G., and Sandamirskaya, Y. (2019). The importance of space and time for signal processing in neuromorphic agents: the challenge of developing low-power, autonomous agents that interact with the environment. *IEEE Signal Process. Magaz.* 36, 16–28. doi: 10.1109/MSP.2019.2928376

Izhikevich, E. M. (2006). Polychronization: computation with spikes. *Neural Comput.* 18, 245–282. doi: 10.1162/089976606775093882

Martin del Campo, S. (2017). *Unsupervised feature learning applied to condition monitoring* (Ph.D. Thesis). Luleå University of Technology, Embedded Intelligent Systems Lab, Luleå, Sweden.

Martin del Campo, S., Albertsson, K., Nilsson, J., Eliasson, J., and Sandin, F. (2013). "FPGA prototype of machine learning analog-to-feature converter for event-based succinct representation of signals," in *Machine Learning for Signal Processing (MLSP), 2013 IEEE International Workshop on* (Southampton), 1–6.

Martin del Campo, S., and Sandin, F. (2017). Online feature learning for condition monitoring of rotating machinery. *Eng. Appl. Artif. Intell.* 64, 187–196. doi: 10.1016/j.engappai.2017.06.012

Mauk, M. D., and Buonomano, D. V. (2004). The neural basis of temporal processing. *Annu. Rev. Neurosci.* 27, 307–340. doi: 10.1146/annurev.neuro.27.070203.144247

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

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

Nielsen, C., Qiao, N., and Indiveri, G. (2017). "A compact ultra low-power pulse delay and extension circuit for neuromorphic processors," in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Turin), 1–4.

Nilsson, M. (2018). *Monte carlo optimization of neuromorphic cricket auditory feature detection circuits in the dynap-se processor* (Master's Thesis). Luleå University of Technology, Luleå, Sweden.

Pfeiffer, M., and Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Front. Neurosci.* 12:774. doi: 10.3389/fnins.2018.00774

Rost, T., Ramachandran, H., Nawrot, M. P., and Chicca, E. (2013). "A neuromorphic approach to auditory pattern recognition in cricket phonotaxis," in *Circuit Theory and Design (ECCTD), 2013 European Conference on* (Dresden: IEEE), 1–4.

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

Schemmel, J., Kriener, L., Müller, P., and Meier, K. (2017). "An accelerated analog neuromorphic hardware system emulating nmda- and calcium-based non-linear dendrites," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK), 2217–2226.

Schöneich, S., Kostarakos, K., and Hedwig, B. (2015). An auditory feature detection circuit for sound pattern recognition. *Sci. Adv.* 1:e1500325. doi: 10.1126/sciadv.1500325

Schuman, C. D., Potok, T. E., Patton, R. M., Birdwell, J. D., Dean, M. E., Rose, G. S., et al. (2017). A survey of neuromorphic computing and neural networks in hardware. *CoRR*, abs/1705.06963.

Sheik, S., Chicca, E., and Indiveri, G. (2012a). "Exploiting device mismatch in neuromorphic vlsi systems to implement axonal delays," in *Neural Networks (IJCNN), The 2012 International Joint Conference on* (Brisbane, QLD: IEEE), 1–6.

Sheik, S., Coath, M., Indiveri, G., Denham, S. L., Wennekers, T., and Chicca, E. (2012b). Emergent auditory feature tuning in a real-time neuromorphic vlsi system. *Front. Neurosci.* 6:17. doi: 10.3389/fnins.2012.00017

Sheik, S., Pfeiffer, M., Stefanini, F., and Indiveri, G. (2013). "Spatio-temporal spike pattern classification in neuromorphic systems," in *Conference on Biomimetic and Biohybrid Systems* (London: Springer), 262–273.

Strukov, D., Indiveri, G., Grollier, J., and Fusi, S. (2019). Building brain-inspired computing. *Nat. Commun.* 10:4838. doi: 10.1038/s41467-019-12521-x

Swadlow, H. A. (1985). Physiological properties of individual cerebral axons studied *in vivo* for as long as one year. *J. Neurophysiol.* 54, 1346–1362. doi: 10.1152/jn.1985.54.5.1346

Van der Spiegel, J., Donham, C., Etienne-Cummings, R., Fernando, S., Mueller, P., and Blackman, D. (1994). "Large scale analog neural computer with programmable architecture and programmable time constants for temporal pattern analysis," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)* Vol. 3 (Orlando, FL), 1830–1835.

Wang, R. M., Cohen, G., Stiefel, K. M., Hamilton, T. J., Tapson, J. C., and van Schaik, A. (2013). An FPGA implementation of a polychronous spiking neural network with delay adaptation. *Front. Neurosci.* 7:14. doi: 10.3389/fnins.2013.00014

Wang, R. M., Hamilton, T. J., Tapson, J., and van Schaik, A. (2014). A mixed-signal implementation of a polychronous spiking neural network with delay adaptation. *Front. Neurosci.* 8:51. doi: 10.3389/fnins.2014.00051

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 reviewer FC declared providing technical help to the authors with the material they used in their research, with no collaboration, before the review.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 09 April 2020
doi: 10.3389/fnins.2020.00275

Event-Based Gesture Recognition With Dynamic Background Suppression Using Smartphone Computational Capabilities

Jean-Matthieu Maro[1]*, *Sio-Hoi Ieng*[1,2] and *Ryad Benosman*[1,2,3,4]*

[1] Sorbonne Université, INSERM, CNRS, Institut de la Vision, Paris, France, [2] CHNO des Quinze-Vingts, INSERM-DGOS CIC 1423, Paris, France, [3] Departments of Ophthalmology/ECE/BioE, University of Pittsburgh, Pittsburgh, PA, United States, [4] Department of Computer Science, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, United States

In this paper, we introduce a framework for dynamic gesture recognition with background suppression operating on the output of a moving event-based camera. The system is developed to operate in real-time using only the computational capabilities of a mobile phone. It introduces a new development around the concept of time-surfaces. It also presents a novel event-based methodology to dynamically remove backgrounds that uses the high temporal resolution properties of event-based cameras. To our knowledge, this is the first Android event-based framework for vision-based recognition of *dynamic* gestures running on a smartphone without off-board processing. We assess the performances by considering several scenarios in both indoors and outdoors, for static and dynamic conditions, in uncontrolled lighting conditions. We also introduce a new event-based dataset for gesture recognition with static and dynamic backgrounds (made publicly available). The set of gestures has been selected following a clinical trial to allow human-machine interaction for the visually impaired and older adults. We finally report comparisons with prior work that addressed event-based gesture recognition reporting comparable results, without the use of advanced classification techniques nor power greedy hardware.

Keywords: gesture recognition, event-based, neuromorphic, background suppression, smartphone, dynamic vision sensor (DVS), dynamic gesture recognition, mobile device

OPEN ACCESS

Edited by:
Elisabetta Chicca,
Bielefeld University, Germany

Reviewed by:
Francisco Barranco,
University of Granada, Spain
Cristina Conde,
Rey Juan Carlos University, Spain

***Correspondence:**
Jean-Matthieu Maro
corr@jmatthi.eu
Ryad Benosman
ryad.benosman@upmc.fr

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *30 October 2019*
Accepted: *10 March 2020*
Published: *09 April 2020*

Citation:
Maro J-M, Ieng S-H and Benosman R (2020) Event-Based Gesture Recognition With Dynamic Background Suppression Using Smartphone Computational Capabilities. Front. Neurosci. 14:275. doi: 10.3389/fnins.2020.00275

1. INTRODUCTION

This article focuses on the problem of gesture recognition and dynamic background suppression using the output of a neuromorphic asynchronous event-based camera (**Figure 1**) connected to a mobile phone (Maro et al., 2019). The system does not rely on off-board resources. Event-based cameras (Lichtsteiner et al., 2008; Delbruck et al., 2010; Posch et al., 2011) offer a novel path to computer vision by allowing to operate at high temporal precision at equivalent frame rates at the order of several kilohertz. Contrary to standard frame-based cameras, which have a pre-defined acquisition rate, individual pixels of neuromorphic cameras are independent and react to relative changes of illuminance in their own field-of-view. Event-based cameras are scene dependent and therefore burn very little power depending on the amount of recorded data (1–10 mW). They hold the promise of low computational costs while operating at high temporal scales. However, there has been no development of a proof of concept using these properties in the context of

FIGURE 1 | A neuromorphic camera (an ATIS) **(B)** is plugged into a smart-phone **(A)** using an USB link **(C)**, allowing mid-air gesture navigation on the smart-phone.

edge computation. In this paper, we introduce a working prototype of a mobile phone event-based application. We chose the popular task of vision-based gesture recognition and dynamic background suppression. These are good targets to make use of the dynamic properties of event-based sensors. We chose to use a scalable machine learning architecture relying on the concept of time-surfaces introduced in Lagorce et al. (2016) and extended it to operate on the limited available computational resources. The system has been designed to operate on each incoming event rather than creating frames from the output of the sensor to then send them to a GPU.

Compared to previous event-based approaches that tackled the problem of gesture recognition, we emphasize the importance of using the information carried out by the timing of past events to obtain a robust low-level feature representation to avoid binning events into frames. We also address the difficult problem of dynamic background suppression by introducing a novel low power event-based technique operating in the temporal domain. This technique goes beyond existing background suppression methodologies. It uses the properties of data-driven acquisition and its high temporal resolution to segment a scene by setting a relation between depth and relative activity, thus allowing the foreground and background to be differentiated.

We also introduce a new dataset of gestures (*NavGesture*) recorded using an event-based camera and available for public download. The neuromorphic field still lacks datasets that take full advantage of the precise timing of event-based cameras. Available datasets such as N-MNIST and N-Caltech101 (Orchard et al., 2015a) are recording scenes where dynamics are artificially introduced. Even true neuromorphic datasets such as Poker-DVS (Serrano-Gotarredona and Linares-Barranco, 2015) or N-Cars (Sironi et al., 2018) contain limited *intrinsic* dynamic properties that could be used for classification. We intend to observe objects that can be classified using only their dynamic properties (or motion) and not from their spatial distribution. As an example, if one considers the N-Cars (Sironi et al., 2018) database, most objects appear as "flashes" that provide a snapshot of the object to be recognized. The DvsGesture dataset (Amir et al., 2017) fulfills the requirement of having dynamic properties, however the camera is set static with the same centring for all samples with no activity in the background. The American Sign Language dataset, ASL-DVS (Bi et al., 2019)

offers various centring and scales but aims to recognizing hand postures and also lacks dynamic properties. The proposed dataset (*NavGesture*) is a new step toward bridging the gap between laboratory-recorded datasets and everyday real situations. It features a challenging set of dynamic gestures to classify, with heterogeneous centring and scaling using a moving camera both in indoor and outdoor environments.

1.1. Gesture Recognition on Mobile Devices

Gesture recognition on mobile devices is a quickly expanding field of research that uses a variety of sensors and methods (Pisharady and Saerbeck, 2015; Asadi-Aghbolaghi et al., 2017; Aditya et al., 2018). While resource-constrained devices such as smartphones disallow the use of certain technologies requiring high energy consumption such as vision-based depth (RGB-D) sensors, current mobile phones have a wide variety of built-in sensors. Several techniques use: phone speakers (Wang Z. et al., 2019), inertial sensors (Deselaers et al., 2015; Gupta et al., 2016; Li et al., 2018) or proximity sensors (Kim and Kang, 2010; Cheng et al., 2011). It is worth noticing that (Won et al., 2015) propose to use a neuromorphic camera as a proximity sensor instead of the conventional infra-red sensitive photo-diode. Other techniques use external components such as: e-gloves (Kau et al., 2015), radio-frequency chips (Kellogg et al., 2014) and even in some cases an external IMU for teeth gesture recognition (Gálvez et al., 2019).

Smartphones also use standard RGB cameras, allowing vision-based recognition. As pointed in Chakraborty et al. (2018), dynamic gestures must be captured at high frame rates in order to avoid motion blur and in some cases even missing a gesture. However, processing high frame rates video data in real time on a smartphone is computationally challenging if not impossible. This might explain why most if not all of the vision-based gesture recognition methods running on smartphones without off-board processing are only applied to static gestures (hand poses) (Ghanem et al., 2017; Lahiani et al., 2017). The only vision-based dynamic gesture recognition method for smartphone we found is proposed by Rao and Kishore (2016). However, no proof of concept operating on a mobile phone has been developed as the system has only been simulated on a resource-capped standard computer. Furthermore vision-based methods require to segment the hand from the background. This is often solved either by background pre-sampling (Dadiz et al., 2017) or by using skin color calibration (Jin et al., 2016; Lahiani et al., 2016). We will shortly show that this can be performed differently if one considers the high temporal resolution of event-based cameras.

1.2. Gesture Recognition Using Event-Based Cameras

Neuromorphic cameras coupled with event-based processing open new perspectives for resource management as both computation and memory can be allocated only to active parts of a visual scene. In the past few years a large number of work tackled computer vision problems using event-based cameras while keeping in mind the necessity of avoiding at all costs the temptation to generate frames from the sensor's output, to cite a few: optical flow estimation (Benosman et al., 2014), high-speed tracking (Serrano-Gotarredona et al., 2009; Ni et al., 2012; Valeiras et al., 2015), object classification (Sheik et al., 2013; Lagorce et al., 2015; Orchard et al., 2015b), 3D reconstruction (Ieng et al., 2018), or pose estimation (Reverter Valeiras et al., 2016).

Generating images from the output of event-based cameras to take advantage of decades of standard computer vision research is becoming a popular stream of research (Kogler et al., 2009; Mueggler et al., 2015; Pradhan et al., 2019; Rebecq et al., 2019). This has lead to the development of pipelines that convert conventional frame-based datasets into events either using hardware (Orchard et al., 2015a; Hu et al., 2016; Wang Y. et al., 2019) or software (Chadha et al., 2019). These data are then often converted back into frames in order to use frame-based techniques such as CNN. There is currently a need to carry out research on event-by-event processing to take full advantage of all the properties of neuromorphic vision sensors (Cadena et al., 2016; Chen et al., 2019). These sensors cannot only be used to generate high frame rates or high dynamic range images as one loses all advantages of the sparseness and low computation power associated to event-based acquisition.

To our knowledge, the first gesture recognition system using a Dynamic Vision Sensors (DVS) is the Rock-Scissor-Paper game from Ahn et al. (2011), which detected the final static hand pose using event activity. Samsung has developed several gesture recognition systems. In early experiments, they proposed to use Leaky Integrate-and-Fire (LIF) neurons to correlate space-time events in order to extract the trajectory of gestures, using a stereo-pair of DVS in Lee J. et al. (2012); Lee et al. (2014). This method is also adapted to track a finger tip using a single DVS (Lee J. H. et al., 2012), and event activity rate is also used to discriminate finger tip movements from hand swipes. Samsung also proposed to use the Adaptive Resonance Theory (ART) for continuous gesture recognition, first with HMM (Park et al., 2012), then with CNN (Park et al., 2015). In parallel to the trajectory extraction approaches, global motion-based features were proposed. Kohn et al. (2012) proposed a motion-based analysis of body movements using the relative event activity accumulated into 40 ms frames, while Lee K. et al. (2012) used pseudo optical-flow. To cope with varying speeds, Clady et al. (2016) proposed a motion-based feature that decays depending on the speed of the optical flow. Two end-to-end neuromorphic systems for gesture recognition have been proposed in recent years. The first one used the SpiNNaker neuromorphic board (Liu and Furber, 2015) and the second was implemented by IBM Research on the TrueNorth neuromorphic chip (Amir et al., 2017). However, both systems bin events into frames at some point in order to use a CNN for classification. Along with their implementation IBM has also released the DvsGesture dataset, which has become widely used in the neuromorphic community. It has been used in multiple papers: spatio-temporal filters that feed a CNN (Ghosh et al., 2019), SNN (Kaiser et al., 2018; Shrestha and Orchard, 2018), and a PointNet adaptation (Wang Q. et al., 2019).

Sign Language recognition has also been investigated but with a focus on static hand postures using events-to-frame techniques (Rivera-Acosta et al., 2017) or a graph-based CNN (Bi et al., 2019). Chen et al. (2019) proposed a new representation called *Fixed Length Gist Representation* (FLGR), mapping events to a higher dimensional feature. All presented methods used data from a static neuromorphic camera, with no background clutter. Furthermore, centring and scaling is in general the same except for (Bi et al., 2019). The only work to our knowledge with a focus on cluttered background and featuring one to several subjects *per se* quence, is the hand detection method proposed by Li et al. (2017). Unfortunately, they did not release their dataset. Also, it is worth mentioning that almost all presented works use at some point an events-to-frame conversion such as temporal or index binning, pixel spike rate or global memory surfaces. The only methods that process events in an event-based manner are scarce: (Lee J. H. et al., 2012; Lee K. et al., 2012), Clady et al. (Clady et al., 2016), SLAYER (Shrestha and Orchard, 2018) and FLGR (Chen et al., 2019).

In this work, we will consider more general scenarios offered by a moving camera that induces numerous new issues to solve such as: a higher number of emitted events, heterogeneous centering and scaling, unwanted shaking and important background clutter. Eliminating the background is an important step for event by event processing. Kyung et al. (2014) proposed a background suppression method for neuromorphic cameras, but converted events to frames. Our approach is purely event-based and drastically contrasts from any existing background removal algorithm as it uses only the timing of events and it does not rely on conventional approaches such as: code-books (Elgammal et al., 2000), probabilistic approaches (Stauffer and Grimson, 1999), sample-based methods (Barnich and Droogenbroeck, 2011), subspace-based techniques (Oliver et al., 2000), or even deep learning (Babaee et al., 2018).

2. EVENT-BASED CAMERAS AND THE EVENT-BASED PARADIGM

The Address Event Representation (AER) neuromorphic camera used in this work is the Asynchronous Time-based Image Sensor (ATIS) (see **Figure 1B**) (Posch et al., 2011). Each pixel is fully autonomous, independent, and asynchronous, it is triggered by a change in contrast within its field of view. A pixel emits a visual event when the luminance change exceeds a certain threshold, typically around 15% in contrast. The nature of this change is encoded in the *polarity p* of the visual event, which can be either ON ($p = 1$) or OFF ($p = 0$), depending on the sign of the luminance change (see **Figure 2**). We must emphasize that p does not carry meaningful information *per se*: indeed, a given object can induce both polarities depending on if the background is lighter or darker than the observed object. Hence, the polarity is context-dependant and can not be taken into account except in the case of a controlled environment and stimulus. The ATIS has a high temporal precision, in the order of hundreds of microseconds, which allows the capture of highly dynamical scenes while avoiding motion blur (Mueggler et al., 2014). The k-th visual event e_k of the output stream of the camera can be mathematically written as the following triplet:

$$e_k = (\boldsymbol{x_k}, t_k, p_k) \tag{1}$$

where $\boldsymbol{x_k}$ is the spatial location of the visual event on the focal plane, t_k its time-stamp, and p_k its polarity.

3. METHODS

3.1. Dynamic Background Suppression

The Dynamic Background Suppression (DBS) uses the simple idea that the closer an object is to the camera, the more events it will generate as its apparent motion will be more important than a farther object. From this property it is possible to link the relative local activity within the focal plane to depth. A low event relative activity can be associated to the background and hence dismissed, whereas relative high activity regions could correspond to the foreground. Although the technique could be applied to each pixel, we will estimate the relative activity considering portions of the focal plane that will be divided into a grid of cells, as shown in **Figure 3**.

Let each cell c be composed of a set of pixels where activity is expressed by A_c. For each incoming event $e_k = (\boldsymbol{x_k}, t_k, p_k)$ emitted by a pixel belonging to a cell c, we can apply the following update of its activity A_c as:

$$A_c \leftarrow A_c \cdot \exp(-\frac{t_k - t_c}{\tau_b}) + 1 \tag{2}$$

where t_k is the time-stamp of the current event e_k, t_c the last time c has been updated, and τ_b is a decaying time-constant.

We can then compute the average activity $\overline{A}$ of a all cells. An incoming event $e_k = (\boldsymbol{x_k}, t_k, p_k)$ belonging to c is sent to the machine learning module only if:

$$A_c \geq \max(\alpha\overline{A}, A_T) \tag{3}$$

where α is a scalar to set the aggressiveness of the filter, and A_T is a threshold for minimum foreground activity. The activity of a cell and the threshold $\overline{A}$ are computed for each incoming event, which enables or disables a given cell at the temporal resolution of incoming events. Cells with a low activity are considered as background and are prevented from emitting events. In principle each time a cell is updated the general mean activity has to be updated. Events are timed at the μs and are orders of magnitude faster than any conventional urban real scene dynamics. The mean activity can then be updated at much lower temporal scales set experimentally according to the computation power available and perhaps the situation (one can infer acceleration from the built-in IMU). The proof of principle of the technique is shown in **Figure 4** and an example of a denoised clip is provided in **Video S1**.

3.2. Time-Surfaces as Spatio-Temporal Descriptors

A time-surface (Lagorce et al., 2016) is a descriptor of the spatio-temporal neighborhood around an incoming event e_k. We

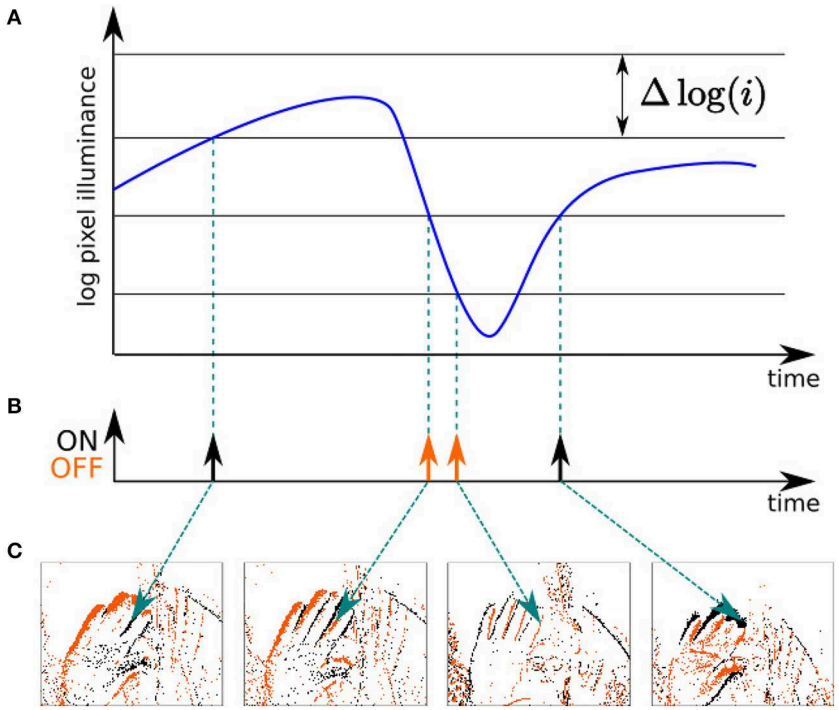


FIGURE 2 | Principle of operation of the neuromorphic camera used in this work. **(A)** When the change in illuminance of a given pixel's field of view exceeds a certain threshold, **(B)** it emits a visual event, which is either "ON" or "OFF" depending on the sign of the change. **(C)** A given pixel responds asynchronously to the visual stimuli in its own field of view.

define the time-context $T_k(\boldsymbol{u}, p)$ of the event e_k as a map of time differences between the time-stamp of the current event and the time-stamps of the most recent events in its spatial neighborhood. This $(2R+1) \times (2R+1)$ map is centered on e_k, of spatial coordinates $\boldsymbol{x_k}$. The time-context can be expressed as:

$$T_k(\boldsymbol{u}, p) = \{t_k - t \mid t = \max_{j \leq k} \{t_j \mid \boldsymbol{x_j} = (\boldsymbol{x_k} + \boldsymbol{u}),\ p_j = p\}\} \quad (4)$$

where $\boldsymbol{u} = [u_x, u_y]^T$ is such that $u_x \in [\![-R, R]\!]$ and $u_y \in [\![-R, R]\!]$.

Finally, we obtain the time-surface $S_k(\boldsymbol{u}, p)$ associated with the event e_k, by applying a linear decay kernel of time-constant τ to the time-context T_k:

$$S_k(\boldsymbol{u}, p) = \begin{cases} 1 - \frac{T_k(\boldsymbol{u},p)}{\tau}, & \text{if } T_k(\boldsymbol{u}, p) < \tau \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

S_k is a low-level representation of the local spatio-temporal neighborhood of the event e_k. **Figure 5** illustrates how time-surfaces are computed from the stream of events.

Discarding time-surfaces. A time-surface can be computed for each new incoming event, but would generate overlapping time-surfaces and introduce redundancy. As the event-based camera performs native contour extraction, we must ensure that a sufficient number of events to form a full contour are taken into account. Therefore, time-surfaces must be discarded if they contain too little information, using the following heuristic:

$$card(\{(\boldsymbol{u}, p),\ T_k(\boldsymbol{u}, p) < \tau\}) \geq 2R \quad (6)$$

3.3. Event-Based Hierarchical Pattern Matching

Following the principle of using deep multiple temporal and spatial scales introduced in HOTS (Lagorce et al., 2016), incoming visual events are fed to a network composed of several layers. As events flow into the network, only their polarities are updated on successive "feature planes." Polarities in the network correspond to learned patterns or elementary features at that temporal and spatial scale. However, as time-surfaces can be discarded, the network output stream contains less events than the input stream, which is an important property that builds on the native low output of the event-based camera to lower the computational cost.

3.3.1. Creating a Layer and Learning Prototypes

An iterative online clustering method is used to learn the base patterns (hereinafter called prototypes), as it allows to process events as they are received, in an event-based manner. A layer is composed of a set of N prototypes, which all share the same radius R (which corresponds to the neuron's receptive field), and the same time-constant τ. The triplet (N, R, τ) defines a layer. First, a set of N time-surface prototypes C_i, with $i \in [\![0, N-1]\!]$, is created. The C_i are initialized by using random time-surfaces obtained from the stream of events. For each incoming event e_k we compute its associated time-surface S_k of radius R and time-constant τ. Using the L2 Euclidean distance, we compute the closest matching prototype C_i in the layer, which we update with

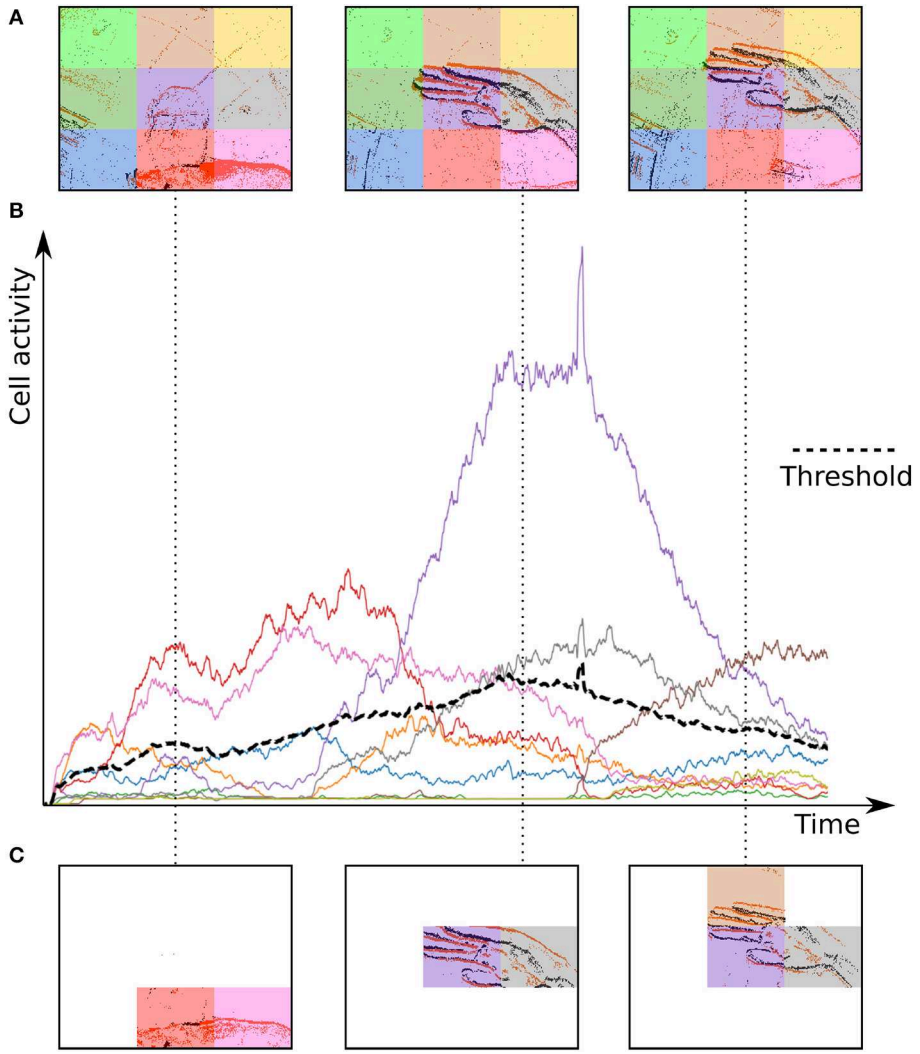


FIGURE 3 | Operating principle of the Dynamic Background Suppression (DBS). **(A)** A gesture is performed in front of the camera, which pixel array is divided into cells. **(B)** Each cell has its own activity counter that decays over time. **(C)** Only cells with their activity greater than the mean activity (black dashes) of all cells can spike.

S_k using the following rule, improved from Lagorce et al. (2016):

$$C_i \leftarrow C_i + \alpha_i \frac{S_k \cdot C_i}{\|S_k\| \ \|C_i\|}(S_k - C_i) \tag{7}$$

with α_i the current learning rate of C_i defined as:

$$\alpha_i = \frac{1}{1 + A_i}$$

where A_i is the number of time-surfaces which have already been assigned to C_i. If a prototype C_i is poorly triggered, it is re-initialized and forced to learn a new pattern. This prevents badly initialized prototypes to stay unused, and helps them converge to meaningful representations.

3.3.2. Building the Hierarchy

One can then stack layers in a hierarchical manner, in order to form a network (see **Figure 6**). First, the visual stimulus is presented to the event-based camera (**Figure 6A**), which outputs a stream of visual events. A given event e_m of the stream must go through all the layers before the next event e_{m+1} is processed. At each layer (N, R, τ), if the time-context T_m of the event e_m satisfies Equation (6), the corresponding time-surface S_m is computed (see **Figure 6B**). Then, the best matching prototype C_c is updated using Equation (7) (see **Figure 6B**). At this point, the polarity p_m of e_m is modified so that $p_m = c$, c being the ID of the best matching prototype. Event e_m is then sent to the next layer to be processed in a similar manner. We must emphasize that the first layer, which receives *visual events* from the camera does not take the polarity (that corresponds to the increase or decrease in contrast) into account for the reason exposed in section 2. All *visual events* have their polarity p set to zero. In the subsequent layers, however, the polarity now encodes a pattern, and we refer to them as *pattern events* instead of *visual events* for which the polarity corresponds to

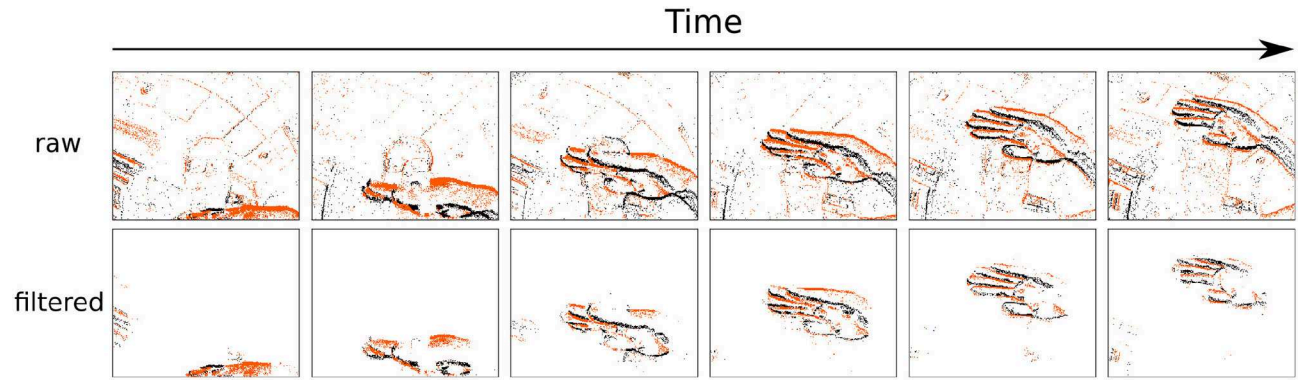


FIGURE 4 | Denoising example of a gesture clip from the NavGesture-walk data-set. The presented gesture is a "swipe down". Top row is the raw stream of visual events, and the bottom row is the denoised stream, at the output of the 3rd stage of the cascade presented in this paper. Each snapshot from the top row is made of 10,000 events, and bottom row contains only the kept events of those 10,000. "ON" events are orange, "OFF" events are black. The filtering lead to the removal of 83.8% of all events. Even after removing this many events each gesture is still easily recognizable by the human eye.

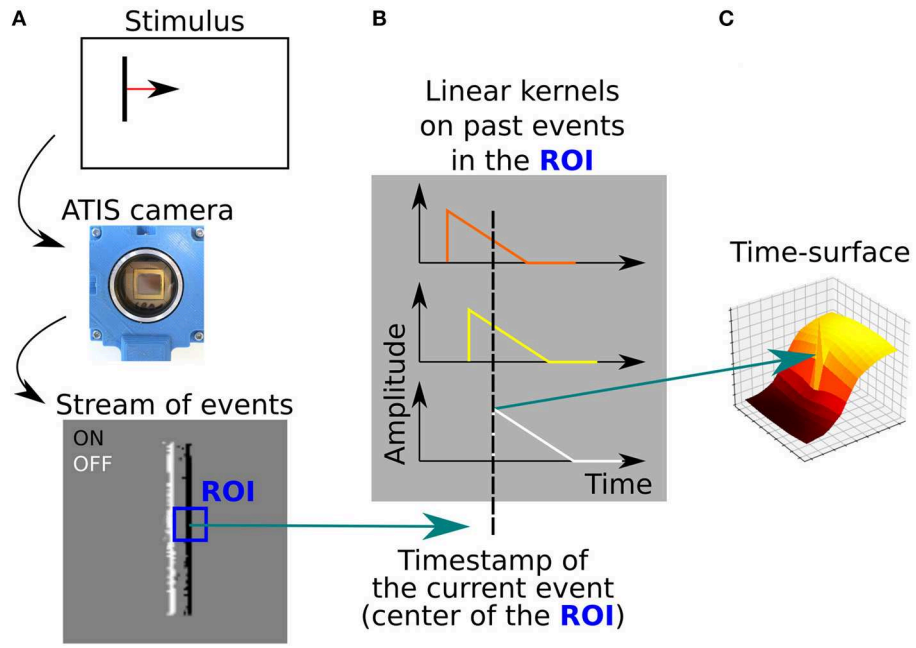


FIGURE 5 | (A) A moving vertical bar is presented to the event-based camera, which output a stream of visual events. The edges of the bar are ON (white) and OFF (black) events. A ROI is defined around the current event (blue square). **(B)** The time-stamps of visual events contained the ROI are decayed using a linear kernel. **(C)** The resulting extracted time-surface, that encodes both the contour orientation and the dynamic of the motion.

a luminance change. Pattern events are then fed to the next layer, and processed in a similar manner. As we go higher in the hierarchy of layers, subsequent layers combine patterns from previous layers, thus their prototypes (and so the corresponding polarities) encode more and more sophisticated patterns. As an illustration, the first layer can only encode the shape and the direction of the motion. The second layer however, because it is working with the first layer output can encode changes of direction in the motion. Once the full hierarchy has been trained, meaning that its time-surface prototypes have converged, the learning is disabled: prototypes are no longer updated using Equation (7).

The network can now serve as a feature extractor: the polarities of events output by the network will be used as features for classification. Because this algorithm is truly event-based and data-driven the computation time directly depends on the number of events transmitted by the camera.

4. A NEW NEUROMORPHIC DATASET: NAVGESTURE

As mentioned in the previous section, existing gesture and action recognition datasets are recorded using a non-moving camera

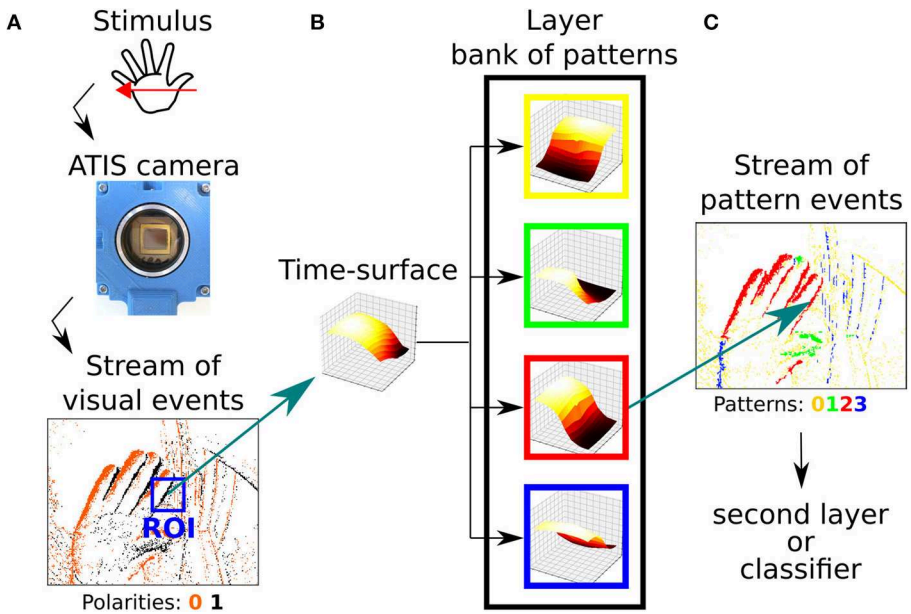


FIGURE 6 | (A) A stimulus is presented in front of a neuromorphic camera, which encodes it as a stream of event. **(B)** A time-surface can be extracted from this stream. **(C)** This time-surface is matched against known pattern, which are also time-surfaces, and that can be used as features for classification.

set in front of a static background (Amir et al., 2017; Bi et al., 2019; Chen et al., 2019; Ghosh et al., 2019; Wang Y. et al., 2019). In some other popular neuromorphic datasets such as N-MNIST and N-Caltech101 (Orchard et al., 2015a), the event-based camera is set up on a pan-tilt in front of a computer screen, hence the dynamics of recorded objects correspond to the pan-tilt movement. The same issue arises in N-Cars (Sironi et al., 2018) because of the very short duration of each clip. Furthermore cars are cropped, removing most of the background.

The proposed dataset offers a challenging gesture recognition task because of its dynamic and changing backgrounds. All gestures were recorded in selfie mode, with the users holding the camera with one hand and performing the gesture with their free hand. The fact that users where holding the phone leads to a wide variety of centring and gesture distance to the camera. The dataset features both right-handed and left-handed users. The users were either sitting or walking, indoors and outdoors, in uncontrolled lighting conditions. The neuromorphic camera used is an ATIS (Posch et al., 2011) with a lens VM-6.5-IR-CCD from Universe Optics. This choice was made in order to facilitate the "auto"-centring by the end-users, by allowing a larger field of view.

The NavGesture dataset has originally been designed to facilitate the use of a smartphone by the elderly and the visually impaired. The gesture dictionary has 6 gestures in order to be easily memorized. They have been selected to be the most compact set able to operate a mobile phone. Four of them are "sweeping" gestures: *Right, Left, Up, Down*. These are designed to navigate through the items in a menu. The *Home* gesture, a "hello"-waving hand, can be used to go back to the main menu, or to obtain help. Lastly, the *select* gesture, executed only using fingers, closing them as a claw in front of the device, and then reopening them, is used to select an item.

The NavGesture dataset is split into two subsets, depending on whether users were sitting or walking: NavGesture-sit and NavGesture-walk. The NavGesture-sit dataset features 28 subjects, 12 being visually impaired subjects, with a condition ranging from 1 to 4/5 on the WHO blindness scale and 16 being people from the laboratory. The gestures were recorded in real use condition, with the subject sitting and holding the phone in one hand while performing the gesture with their other hand. Some of the subjects were shown video-clips of the gestures to perform, while others had only an audio description of the gesture. This inferred some very noticeable differences in the way each subject performed the proposed gestures, in terms of hand shape, trajectory, motion and angle but also in terms of the camera pose. Each subject performed 10 repetitions of the 6 gestures. In a second stage, all the acquired clips were manually labeled and segmented. We removed problematic clips, such as wrongly executed gestures or gestures executed too close to the camera. The manually curated dataset contains a total of 1,342 clips.

In the NavGesture-walk the users walked through an urban environment, both indoors in the laboratory, and outdoors in the nearby crowded streets in the center of Paris. Users recorded the gestures while walking, holding the phone with one hand and performing the gestures with the other. This uncontrolled setting leads to much more variation in pose, unwanted camera movements, dynamic backgrounds and lighting conditions. This dataset features 10 people from the laboratory that performed 5 times each of the 6 gestures. The dataset contains a total of 339 clips. An overview is presented in **Table 1**. An example of

TABLE 1 | Characteristics of the three Gesture Datasets used in this work.

Dataset	#users	#classes	#clips	Camera	Background	Framing
DvsGesture	29	10 + 1	1,342 + 122	Static	No	Upper body
NavGesture-sit	28	6	1,342	Handheld	Yes, moderate	Selfie, user sitting
NavGesture-walk	10	6	339	Handheld	Yes, important	Selfie, user walking

the "Swipe Up" gesture is shown in **Figure 4**. The NavGesture dataset is publicly available at https://www.neuromorphic-vision.com/public/downloads/navgesture/.

5. EXPERIMENTS AND RESULTS

The first experiment on the Faces dataset focuses on extracting static properties. We show that a single layer is sufficient to provide good results. The following experiments required more layers. As the neuromorphic camera detects change in contrast, these can either be ON or OFF events depending on the contrast between the foreground and the background. Indeed, the same moving object could generate ON events in front of a dark background, and OFF events in front of a light background, as explained earlier. This is the reason why in all the following experiments we did not take the polarity of visual events into account, as the polarity is context-dependent. An example of this phenomena is a moving hand in front of a black and white stripped background. This is why we considered that only the illuminance *change* carries information for these classification tasks, and not the fact that the illuminance *increased* (ON event) or *decreased* (OFF event).

For all classification tasks, the output of end-layers (larger time scale) is integrated over time to generate a histogram of activity per feature as in Lagorce et al. (2016). This histogram is then used as a dynamic signature of the observed stimulus. This signature is fed to a classifier, in this case a nearest neighbor. More sophisticated classifiers could be used, but this demonstrates that extracted features are sufficient for classification.

5.1. Static Properties: Experiments on the Faces Dataset

This dataset contains clips of the faces of 7 subjects. Each subject was recorded 24 times, resulting in 168 clips. The subjects had to move their head in a square-shaped trajectory, by following a dot on a computer screen. The dynamic is therefore the same for all subjects, and does not carry any meaningful information for the classification task. Experiments were performed on a standard desktop computer. We performed 10-fold cross-validation with 5 examples in the train subset, and 19 in the test subset. We used a single-layer with $N = 32$ prototypes, receptive fields of radius $R = 6$ and $\tau = 5$ ms, we obtained 96.6% recognition score on this dataset. By increasing the number of prototypes to $N = 64$, we achieved 98.5% in average recognition rate. We noticed that increasing τ higher than 5 ms was not beneficial and even decreased our classification accuracy. This is because time-surfaces encode both static properties such as shape and dynamic properties such as optical flow. A small τ will mainly encode static properties whereas a larger τ will also encode dynamic properties such as pseudo optical-flow. When we added a second layer, the recognition rate dropped. A single layer is therefore sufficient to encode static properties such as shape. The classification was made using a 1-nearest neighbor, and does not rely on advanced classification techniques.

In comparison, the HOTS model in Lagorce et al. (2016) performed at 79% using a three-layer architecture, with its end-layer having $N = 32$ of prototypes. It must be noted that this improvement in recognition rate also comes with a faster computation because of the reduction in the size of used time-surfaces, from size 4,624 in HOTS to size 169 in our work.

Classification scores depend on the number of prototypes: the more prototypes, the higher the recognition rate.

5.2. Dynamic Properties: Experiments on the NavGesture Datasets

In both NavGesture-sit and NavGesture-walk datasets, subjects hold the phone in their hand, which results in camera movements and unwanted jitters that generate background activity. In the case of the NavGesture-walk the visual background is even more present as subjects are walking while performing the gestures. The experiments were performed on a standard desktop computer, and we used k-fold cross-validation, with k the number of subjects.

In order to remove events generated by the background we used the Dynamic Background Suppression method introduced in section 3.1. The DBS uses the following parameters, set experimentally:

- $\tau_b = 300\mu$s
- $\alpha = 2$
- $A_T = 5$
- grid size : 3×3

Figure 4 illustrates the effect of the DBS. **Table 2** reports the mean percentage of remaining events for each gesture after removing the background. The DBS allows to remove around 40% of events before the feature extraction. This has a direct impact on processing time as we compute event by event.

In our experiments we used networks composed of 1 to 3 layers. We observed that two-layers networks perform better. Some gestures such as "Select" or "Home" have changes in direction, which can be encoded by networks with two or more layers. However, we suspect that three-layers networks encode features that are too complex for the stimulus, resulting in less discriminative features and a lower recognition rate.

Because events are decayed over time, the value of τ must correspond to the dynamic of the stimulus (Clady et al., 2016). If

τ is too small, the extracted time-surface will encode only spatial information. If τ is too large, the trail of older events will blur the shape, encoding only direction of movement. In more extreme cases with τ going to larger and larger values, the resulting time-surface will carry less and less information, as all past events will have the same weight. Of course this has also a close relation with the radius of the time-surface as larger radii can encode longer trails of events.

This observation leads to the fact that τ should be set in regard to the radius R of the time-surface and the velocity v of the apparent motion in pixel per second:

$$\tau \approx \frac{R}{v} \tag{8}$$

We observed that a first layer with a τ value in the order of 10 ms allowed to encode both shape and direction of motion (only direction, not changes in direction). The second and end-layer has a τ value of 100 ms, in order to encode changes in the direction of motion.

A direct difficulty comes from the almost fish-eye field of view of the camera: if the phone is not held vertically or if the gesture is a bit off-axis, it becomes very difficult at the edges of the field of view to determine if the motion is vertical or horizontal.

Ablation study. In order to assess the benefits of the DBS in obtaining better recognition rates, we compared the performance achieved with and without the DBS. Results show that DBS does improve recognition rates, increasing the score from 81.3 to 92.6% when using the NavGesture-walk dataset, as shown in **Table 3**.

TABLE 2 | Mean percentage of events left after each the Dynamic Background Suppression for each gesture class.

Gesture	Mean number of event	Mean percentage left after the DBS
Down	988,901	41%
Home	2,398,850	48%
Left	969,014	42%
Right	962,501	43%
Select	1,212,222	30%
Up	1,110,652	44%

5.3. Experiments on the DvsGesture Dataset

Amir et al. (2017) released a 10-class (plus a rejection class with random gestures) dataset of hand and arm gestures, performed by 29 subjects under 3 different lighting conditions. The camera is mounted on a stand while the subjects stood still in front of it. This dataset has no background so the DBS was not used. Authors split the dataset into a training set of 23 subjects and a testing set of 6 subjects, preventing cross-validation for comparison purposes. We used the same 2-layer network architecture as the one used for NavGesture. The only difference is that we increased the number of prototypes in the last layer because the gestures are more complex. In order to take into account the spatial component of gestures, we split the pixel array into sub-regions, using a 3×3 grid. This is possible because the centring is very similar for all clips in the dataset. Hence, the final feature is a histogram of size $3 \times 3 \times 64 = 576$. We achieved a classification accuracy of 96.59% for the 10-class subset and 90.62% for the 10 classes plus the rejection class. One can observe in the confusion matrix (**Figure 7**) that "Hand clap," "Arm roll," "Air guitar," and "Air drum" are the only gestures that are mistaken. These gestures all share very similar hand movements at the same spatial location, located in front of the torso. "Arm roll" and "Air drum" are also very similar. Their difference lie in the fact that hands in "Arm roll" move along the same vertical line, and we suspect that the receptive field is too small to capture this information.

When adding the rejection class, the same gestures get confused. Indeed, only one clip of "Left hand wave" gets mistaken for "Air guitar", which is understandable as the left hand in these two classes performs the same movement at the same location. The global accuracy decreases mostly because of the "Hand clap" that gets misclassified more often and because of the "Other gestures" that also are harder to classify.

One can observe in **Table 4** that for the 10-class classification task our system performs in the same range of accuracy using a k-NN as other very elaborate systems using state-of-the-art neural networks.

It must be noted that the same time constants gave best results for both NavGesture and DvsGesture, which shows that decay must be chosen in accordance with the stimulus, in both case gestures. Indeed, previous work such as HOTS (Lagorce et al., 2016) and (Sironi et al., 2018) used decay times that were three

TABLE 3 | Summary of obtained results on the NavGesture dataset.

ID	Dataset	Layer 1			Layer 2			DBS	Classifier	Results
		N	R	τ	N	R	τ			
E1	NavGesture-sit	8	2	10 ms	8	2	100 ms	✓	k-NN	95.9%
E2	NavGesture-walk	8	2	10 ms	8	2	100 ms	✓	k-NN	92.6%
E3	NavGesture-walk	8	2	10 ms	8	2	100 ms		k-NN	81.3%
E4	NavGesture-walk	8	2	10 ms				✓	k-NN	88.7%

The use of the Dynamic Background Suppression in E2 allows to drastically improve the recognition rate by over 10% compared to E3. Also, the addition of a second layer is beneficial, as shown by the improvement in E2 compared to E4.

orders of magnitude higher than the duration of the stimulus. This resulted in time-surfaces that acted as binary frames instead of encoding the dynamics of the scene. Furthermore, such high decay values resulted in the incapacity of forgetting past events.

6. IMPLEMENTATION ON A SMARTPHONE

The proposed gesture recognition pipeline has been implemented on a mobile phone (Maro et al., 2019), a Samsung Galaxy S6 (model GM-920F), with a custom Android application allowing easy navigation through basic phone functions, such as making a call or sending a pre-defined text message (see **Figure 8**). The event-based camera was directly plugged into the micro-USB port of the mobile phone (see **Figure 1**). The gesture recognition module is implemented in native C++ using JNI to communicate with the Android application. The gesture recognition module consists of basic noise filtering (a refractory period followed by a spatio-temporal denoiser, known as the *background activity filter*, that removes pixel electrical noise), the Dynamic Background Suppression, a 1-layer Feature Extractor ($N = 8$, $R = 2$, $\tau = 10$ ms,) and a k-NN classifier.

We used two strategies to segment gestures, the first one is an "auto-start" based on the global visual scene activity. This option works when users are seated but is inadequate for walking cases. The second strategy relied on pressing a button before a gesture to start the recording. The duration of the recording was tuned experimentally to 2 s which seems to be the experimental upper bound of the duration of a gesture. This 2-s batch of events at once to the gesture recognition module, that returns the gesture class to the Android application to be converted to an Android command. An overview of the system is presented in **Figure 9**.

To assess processing time, we ran five trials for each gesture in two different settings. The input event stream having a duration of 2 s, a real-time processing is reached when the processing time is below 2 s. In the first scenario, the phone was set on a table. In the second scenario the phone was handheld in selfie mode, with the user walking around. All results are compiled in **Table 5**. When looking at the first scenario, we can see that all gestures are under the 2 s barrier, except for the "Home" gesture (a "Hello-waving" gesture). This is because this gesture produces 3 times more events than all other gestures (see **Table 2**). The algorithm being truly event-based, the processing time directly depends on the number of events to process. Also during trials 3 and 4, the user waved his hand 5, 6 times, while in trials 1, 2, and 5 waved only 3, 4 times. The second scenario is the handheld selfie mode scenario, where the background generates a high number of events, hence necessitating longer processing time. However, all gestures except for the gesture "Home" that could be computed in real-time. This gesture should be replaced by another more event-based friendly gesture that would generate less events, or should be more constrained by forcing users to only wave 1 or 2 times.

This prototype was tested by untrained visually impaired end-users, in real use conditions. The subjects were asked to perform certain tasks to operate the phone. These preliminary tests lead to a global accuracy of 78%, which is below the 88.7% accuracy we obtained using the same single layer on the NavGesture-walk dataset. We suspect this is partly due to framing and off-axis handling of the phone.

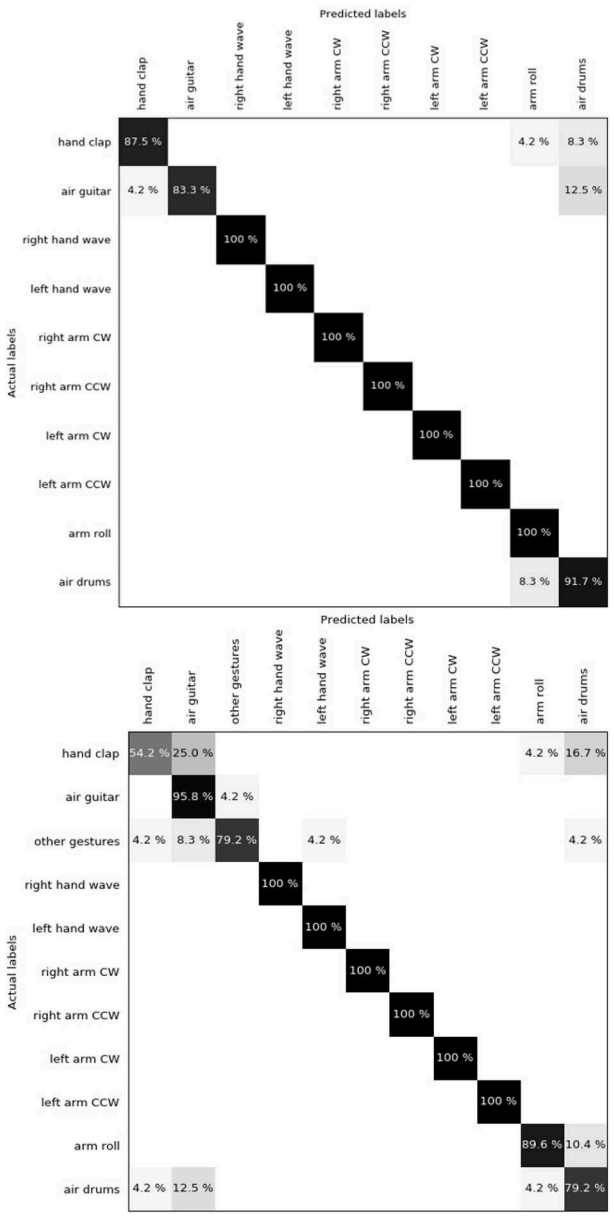


FIGURE 7 | (Top) Confusion matrix for DvsGesture using 10 classes. Global accuracy is 96.59%. "Hand clap", "Arm roll", "Air guitar" and "Air drum" are the only gestures that get confused. The reason might be that they generate similar motion in the same spatial location. **(Bottom)** Introducing the rejection class "Other gestures" amplifies the mismatch between the four precedent gestures, leading to a global accuracy of 90.62%. However, it has almost no impact on other gestures (4.2% in the "Other gestures" row corresponds to only one clip).

TABLE 4 | Comparison in accuracy of state-of-the-art methods for the DvsGesture dataset.

	Method	DvsGesture (10 classes)	DvsGesture (10 classes + 1)
Amir et al. (2017)	CNN (avg 192 ms)	91.77% (96.49%)	91.77% (94.59%)
Shrestha and Orchard (2018)	SLAYER		93.64%
Kaiser et al. (2018)	DECOLLE		94.18%
Ghosh et al. (2019)	ST filter + CNN (avg 200 ms)		94.85% (95.94%)
Kaiser et al. (2019)	SNN eRBP		92.7%
Wang Q. et al. (2019)	PointNet++ (avg 118 ms)	96.34% (97.08%)	94.10% (95.32%)
This work	Time-surfaces + k-NN	96.59%	90.62%

When noted (avg) an averaging scheme was proposed to improve the system accuracy. Our method, although using a simple k-NN classifier performs in the same range for the 10-class classification. However, the k-NN lacks the discriminative power to handle the rejection class on the contrary of more sophisticated classifiers.

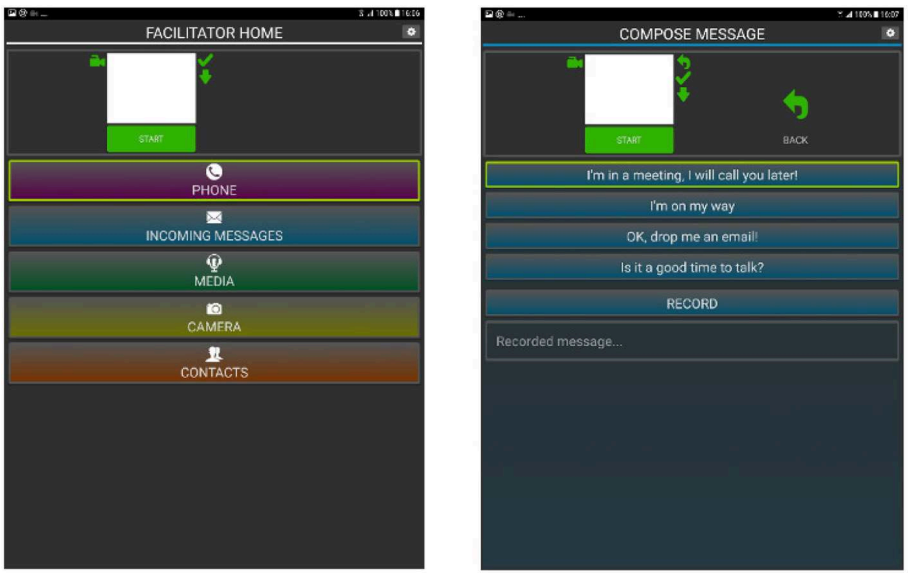


FIGURE 8 | Interface of the Android application that was developed in order to operate the phone using the proposed gestures. Right is the main menu, left illustrates the pre-defined messages the user could send.

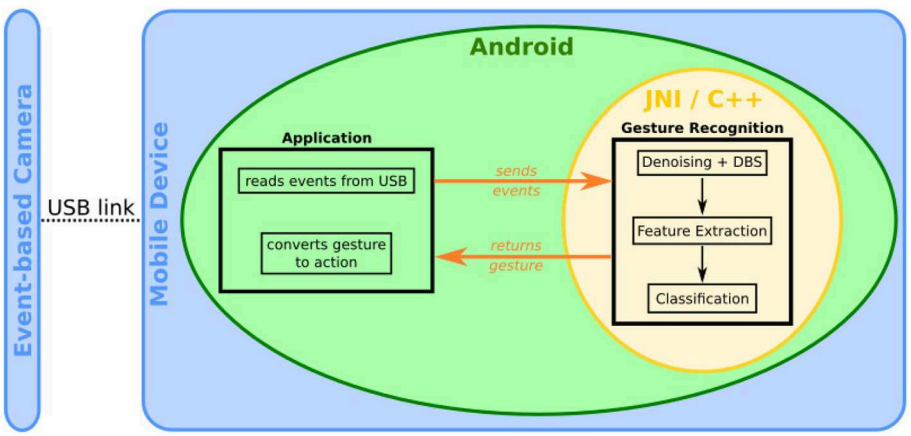


FIGURE 9 | Overview of the Android smartphone system.

TABLE 5 | Processing time in milliseconds for five trials of each gesture on the mobile phone, depending on two conditions.

Trial	Up	Home	Right	Left	Select	Down
Processing time in ms for 2,000 ms of input						
Setting: fixed position (no background)						
1	132	2,343	54	127	40	54
2	57	2,798	60	56	57	45
3	74	3,047	44	275	61	42
4	254	3,833	32	42	29	54
5	48	2107	28	45	47	51
Processing time in ms for 2,000 ms of input						
Setting: outdoor - moving						
1	320	4,119	154	641	138	115
2	614	3,669	704	282	265	451
3	468	4,305	854	421	551	342
4	569	3,681	575	548	956	371
5	899	3,890	722	354	892	620

*"Fixed position" corresponds to a mobile phone set on a table, which means no background. "Outdoor, moving" corresponds to handheld selfie mode, while walking around. Each gesture corresponds to 2,000 ms of events, meaning that except for the "Home" gesture, all proposed gestures can be processed on real-time. The event-based camera is data-driven so a gesture like "Home' which corresponds to several "swipe" gestures will generate more events (see **Table 2**). Our algorithm being truly event-based it is also dependent on the number of events, and takes more processing time the more events it receives.*

7. DISCUSSION AND CONCLUSION

This paper introduced a proof of concept for an event-based Android application for gesture recognition using the computing power of a mobile phone. The main idea was to show that it is possible to make full use of the high temporal resolution of event-based cameras on a power-constrained device. The system used a camera designed to operate with Android using the USB link to stream events. This is by far a very inefficient way to input data to the mobile platform as USB is often too slow and implies time stamping events that adds more bits of information to the acquired events. It is expected that if this type of camera is one day introduced in a mobile device it will use better connectivity such as MIPI buses which are designed for low-power applications and eventually an associated processor. This will remove the need for time stamping and allow both direct routing to the processor and direct computation on the time of arrival of events with no delays. In this paper due to the limitations of the developed software we used 2-s packets of events to optimize communication within the phone. However, we showed that processing required in most cases less than 2 s per batch, which implies that real time performance can be reached if transmission delays are solved. We are confident that a way can be found within Android to transmit events from the camera to the processing stage with no latency. We have also shown that it is possible to handle the stream of events in an asynchronous manner. This allows the temporal machine learning algorithm to be efficient while using only a single core of the phone. The hierarchical temporal network has been optimized for the set of defined gestures showing that robust recognition levels can be reached without requiring the use of GPU or using the non-event-based concept of generating frames from an event-based sensor. Experimental results show that as expected the computation is scene dependent and therefore tightly linked to the amount of events generated by the observed object.

We have also shown that the temporal precision of event-based cameras can tackle different tasks, where it would have been too computationally expensive or even impossible to compute with frames in an elegant and low-power manner. As an example, the background suppression algorithm that for the first time considers outdoor, hand-held scenarios relies on the simple idea that the foreground being closer to the camera will on average generate more events than the background. The idea of using the relative mean activity for background suppression shows that high temporal precision is a valuable feature as it implies that velocity is linked to the amount of data produced, and can be estimated precisely. Moreover, the use of well designed temporal filters can reduce even more the already sparse steam of events, leading to faster event-by-event computation.

There is still so much to develop around the concept of using time as a computational feature for mobile applications. As an example the use of scene dynamics allows to derive techniques such as the one in Lenz et al. (2018) that uses the temporal signature of eye blinks to detect the presence of a face in a scene. This approach introduces an alternative to the current greedy stream of thought that believes everything has to be learned using large databases.

All data collected and used in the paper has been made available to the community. The introduction of this new database will set the groundwork for further work on dynamic background suppression.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

ETHICS STATEMENT

Ethical review and approval was not required for the study on human participants in accordance with the local legislation and institutional requirements. The patients/participants provided their written informed consent to participate in this study. Written informed consent for participation was not required for this study in accordance with the national legislation and the institutional requirements. Written informed consent was obtained from the individual(s) for the publication of any potentially identifiable images or data included in this article.

AUTHOR CONTRIBUTIONS

J-MM compiled the new gesture databases, designed the theory for background suppression, designed the experiments, performed analysis. J-MM and RB interpreted data for gesture recognition. J-MM wrote the article. J-MM, S-HI, and RB helped to edit the manuscript.

FUNDING

This work received funding from the European Union Horizon 2020 research and innovation program under grant agreement No. 644096, and is part of the ECOMODE project.

ACKNOWLEDGMENTS

The authors would like to thank Christopher Reeves for his help in the creation of the NavGesture-sit dataset, and all the people who took part in it. We also would like to thank Antonio Fernández, Andrew Watkinson, and Gregor Lenz for their contribution in the Android application.

This manuscript has been released as a Pre-Print at arXiv (Maro and Benosman, 2018).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00275/full#supplementary-material

Video S1 | Dynamic Background Suppression at work.

REFERENCES

Aditya, K., Chacko, P., Kumari, D., Kumari, D., and Bilgaiyan, S. (2018). "Recent trends in HCI: a survey on data glove, LEAP motion and microsoft kinect," in *2018 IEEE International Conference on System, Computation, Automation and Networking, ICSCA 2018* (Pondicherry), 1–5. doi: 10.1109/ICSCAN.2018.8541163

Ahn, E. Y., Lee, J. H., Mullen, T., and Yen, J. (2011). "Dynamic vision sensor camera based bare hand gesture recognition," in *2011 IEEE Symposium On Computational Intelligence For Multimedia, Signal And Vision Processing* (Paris: IEEE), 52–59. doi: 10.1109/CIMSIVP.2011.5949251

Amir, A., Taba, B., Berg, D. J., Melano, T., McKinstry, J. L., Di Nolfo, C., et al. (2017). "A low power, fully event-based gesture recognition system," in *CVPR* (Honolulu), 7388–7397. doi: 10.1109/CVPR.2017.781

Asadi-Aghbolaghi, M., Clapes, A., Bellantonio, M., Escalante, H. J., Ponce-López, V., Baró, X., et al. (2017). "A survey on deep learning based approaches for action and gesture recognition in image sequences," in *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)* (Washington, DC: IEEE), 476–483. doi: 10.1109/FG.2017.150

Babaee, M., Dinh, D. T., and Rigoll, G. (2018). A deep convolutional neural network for video sequence background subtraction. *Pattern Recogn.* 76, 635–649. doi: 10.1016/j.patcog.2017.09.040

Barnich, O., and Droogenbroeck, M. V. (2011). Vibe: a universal background subtraction algorithm for video sequences. *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn.* 20, 1709–1724. doi: 10.1109/TIP.2010.2101613

Benosman, R., Clercq, C., Lagorce, X., Ieng, S., and Bartolozzi, C. (2014). Event-based visual flow. *IEEE Trans. Neural Netw. Learn. Syst.* 25, 407–417. doi: 10.1109/TNNLS.2013.2273537

Bi, Y., Chadha, A., Abbas, A., Bourtsoulatze, E., and Andreopoulos, Y. (2019). "Graph-based object classification for neuromorphic vision sensing," in *Proceedings of the IEEE International Conference on Computer Vision* (Seoul), 491–501. doi: 10.1109/ICCV.2019.00058

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., et al. (2016). Past, present, and future of simultaneous localization and mapping: towards the robust-perception age. *IEEE Trans. Robot.* 32, 1309–1332. doi: 10.1109/TRO.2016.2624754

Chadha, A., Bi, Y., Abbas, A., and Andreopoulos, Y. (2019). "Neuromorphic vision sensing for CNN-based action recognition," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* (Brighton), 7968–7972. doi: 10.1109/ICASSP.2019.8683606

Chakraborty, B. K., Sarma, D., Bhuyan, M. K., and MacDorman, K. F. (2018). "Review of constraints on vision-based gesture recognition for human-computer interaction," in *IET Computer Vision*, Vol. 12 (Institution of Engineering and Technology), 3–15. doi: 10.1049/iet-cvi.2017.0052

Chen, G., Chen, J., Lienen, M., Conradt, J., Röhrbein, F., and Knoll, A. C. (2019). FLGR: Fixed length GISTS representation learning for RNN-HMM hybrid-based neuromorphic continuous gesture recognition. *Front. Neurosci.* 13:73. doi: 10.3389/fnins.2019.00073

Cheng, H.-T., Chen, A. M., Razdan, A., and Buller, E. (2011). "Contactless gesture recognition system using proximity sensors," in *Consumer Electronics (ICCE), 2011 IEEE International Conference on* (Las Vegas, NV: IEEE), 149–150. doi: 10.1109/ICCE.2011.5722510

Clady, X., Maro, J.-M., Barré, S., and Benosman, R. B. (2016). A motion-based feature for event-based pattern recognition. *Front. Neurosci.* 10:594. doi: 10.3389/fnins.2016.00594

Dadiz, B. G., Abrasia, J. M. B., and Jimenez, J. L. (2017). "Go-Mo (Go-Motion): An android mobile application detecting motion gestures for generating basic mobile phone commands utilizing KLT algorithm," in *2017 IEEE 2nd International Conference on Signal and Image Processing, ICSIP 2017* (Singapore: Institute of Electrical and Electronics Engineers Inc.), 30–34. doi: 10.1109/SIPROCESS.2017.8124500

Delbruck, T., Linares-Barranco, B., Culurciello, E., and Posch, C. (2010). "Activity-driven, event-based vision sensors," in *Proceedings. IEEE International Symposium on Circuits and Systems* (Paris). doi: 10.1109/ISCAS.2010.5537149

Deselaers, T., Keysers, D., Hosang, J., and Rowley, H. A. (2015). GyroPen: Gyroscopes for pen-input with mobile phones. *IEEE Trans. Hum. Mach. Syst.* 45, 263–271. doi: 10.1109/THMS.2014.2365723

Elgammal, A., Harwood, D., and Davis, L. (2000). "Non-parametric model for background subtraction," in *European Conference on Computer Vision* (Dublin), 751–767. doi: 10.1007/3-540-45053-X_48

Gálvez, T. V., Dancu, A., Sapkota, S., and Maes, P. (2019). "Byte.it: discreet teeth gestures for mobile device interaction," in *Conference on Human Factors in Computing Systems - Proceedings* (Glasgow), 1–6.

Ghanem, S., Conly, C., and Athitsos, V. (2017). "A survey on sign language recognition using smartphones," in *ACM International Conference Proceeding Series* (Rhodes). doi: 10.1145/3056540.3056549

Ghosh, R., Gupta, A., Nakagawa, A., Soares, A., and Thakor, N. (2019). Spatiotemporal filtering for event-based action recognition. *arXiv preprint* arXiv:1903.07067.

Gupta, H. P., Chudgar, H. S., Mukherjee, S., Dutta, T., and Sharma, K. (2016). A continuous hand gestures recognition technique for human-machine interaction using accelerometer and gyroscope sensors. *IEEE Sensors J.* 16:1. doi: 10.1109/JSEN.2016.2581023

Hu, Y., Liu, H., Pfeiffer, M., and Delbruck, T. (2016). Dvs benchmark datasets for object tracking, action recognition, and object recognition. *Front. Neurosci.* 10:405. doi: 10.3389/fnins.2016.00405

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

Jin, C. M., Omar, Z., and Jaward, M. H. (2016). "A mobile application of American sign language translation via image processing algorithms," in *Proceedings - 2016 IEEE Region 10 Symposium, TENSYMP 2016* (Bali). doi: 10.1109/TENCONSpring.2016.7519386

Kaiser, J., Friedrich, A., Vasquez Tieck, J. C., Reichard, D., Roennau, A., Neftci, E., and Dillmann, R. (2019). Embodied Neuromorphic Vision with Event-Driven Random Backpropagation. *arXiv [Preprint]*. arXiv:1904.04805.

Kaiser, J., Mostafa, H., and Neftci, E. (2018). Synaptic plasticity dynamics for deep continuous local learning (DECOLLE). *arXiv [Preprint]*. arXiv:1811.10766.

Kau, L. J., Su, W. L., Yu, P. J., and Wei, S. J. (2015). "A real-time portable sign language translation system," in *Midwest Symposium on Circuits and Systems* (Fort Collins, CO). doi: 10.1109/MWSCAS.2015.7282137

Kellogg, B., Talla, V., and Gollakota, S. (2014). "Bringing gesture recognition to all devices," in *NSDI*, Vol. 14 (Seattle, WA), 303–316.

Kim, E. J., and Kang, T. H. (2010). *Mobile device having proximity sensor and gesture based user interface method thereof*. US Patent App. 12/814,809.

Kogler, J., Sulzbachner, C., and Kubinger, W. (2009). "Bio-inspired stereo vision system with silicon retina imagers," in *International Conference on Computer Vision Systems* (Liège: Springer), 174–183. doi: 10.1007/978-3-642-04667-4_18

Kohn, B., Belbachir, A. N., Hahn, T., and Kaufmann, H. (2012). "Event-driven body motion analysis for real-time gesture recognition," in *ISCAS 2012 - 2012 IEEE International Symposium on Circuits and Systems*, 703–706. doi: 10.1109/ISCAS.2012.6272132

Kyung, K. M., Bae, K., Cho, S. H., Jeong, S., and Kim, T. C. (2014). "Background elimination method in the event based vision sensor for dynamic environment," in *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*, 119–120. doi: 10.1109/ICCE.2014.6775934

Lagorce, X., Ieng, S.-H., Clady, X., Pfeiffer, M., and Benosman, R. B. (2015). Spatiotemporal features for asynchronous event-based data. *Front. Neurosci.* 9:46. doi: 10.3389/fnins.2015.00046

Lagorce, X., Orchard, G., Gallupi, F., Shi, B. E., and Benosman, R. (2016). *Hots: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition*. IEEE PAMI.

Lahiani, H., Elleuch, M., and Kherallah, M. (2016). "Real time hand gesture recognition system for android devices," in *International Conference on Intelligent Systems Design and Applications, ISDA* (Marrakech). doi: 10.1109/ISDA.2015.7489184

Lahiani, H., Kherallah, M., and Neji, M. (2017). Vision based hand gesture recognition for mobile devices: a review. *Adv. Intell. Syst. Comput.* 552, 308–318. doi: 10.1007/978-3-319-52941-7_31

Lee, J., Delbruck, T., Park, P. K., Pfeiffer, M., Shin, C. W., Ryu, H., et al. (2012). "Live demonstration: gesture-based remote control using stereo pair of dynamic vision sensors," in *ISCAS 2012 - 2012 IEEE International Symposium on Circuits and Systems* (Seoul). doi: 10.1109/ISCAS.2012.6272144

Lee, J. H., Delbruck, T., Pfeiffer, M., Park, P. K., Shin, C.-W., Ryu, H., et al. (2014). Real-time gesture interface based on event-driven processing from stereo silicon retinas. *IEEE Trans. Neural Netw. Learn. Syst.* 25, 2250–2263. doi: 10.1109/TNNLS.2014.2308551

Lee, J. H., Park, P. K., Shin, C. W., Ryu, H., Kang, B. C., and Delbruck, T. (2012). "Touchless hand gesture UI with instantaneous responses," in *Proceedings - International Conference on Image Processing, ICIP* (Orlando, FL), 1957–1960. doi: 10.1109/ICIP.2012.6467270

Lee, K., Ryu, H., Park, S., Lee, J. H., Park, P. K., Shin, C. W., et al. (2012). "Four DoF gesture recognition with an event-based image sensor," in *1st IEEE Global Conference on Consumer Electronics 2012, GCCE 2012* (Tokyo), 293–294. doi: 10.1109/GCCE.2012.6379606

Lenz, G., Ieng, S., and Benosman, R. (2018). Event-based dynamic face detection and tracking based on activity. *CoRR, abs*/1803.10106.

Li, C., Xie, C., Zhang, B., Chen, C., and Han, J. (2018). Deep Fisher discriminant learning for mobile hand gesture recognition. *Pattern Recogn.* 77, 276–288. doi: 10.1016/j.patcog.2017.12.023

Li, J., Shi, F., Liu, W., Zou, D., Wang, Q., Lee, H., et al. (2017). "Adaptive temporal pooling for object detection using dynamic vision sensor," in *British Machine Vision Conference 2017* (London, UK). doi: 10.5244/C.31.40

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128x128 120db 15us latency asynchronous temporal contrast vision sensor. *IEEE J.f Solid State Circuits*. 43, 566–576. doi: 10.1109/JSSC.2007.914337

Liu, Q., and Furber, S. (2015). Real-time recognition of dynamic hand postures on a neuromorphic system. *Int. J. Electr. Comput. Eng.* 9, 507–514. doi: 10.5281/zenodo.1107243

Maro, J.-M., and Benosman, R. (2018). Event-based gesture recognition with dynamic background suppression using smartphone computational capabilities. *arXiv-[Preprint]* arXiv:1811.07802.

Maro, J.-M., Lenz, G., Reeves, C., and Benosman, R. (2019). "Event-based visual gesture recognition with background suppression running on a smart-phone," in *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)* (Lille: IEEE), 1. doi: 10.1109/FG.2019.8756601

Mueggler, E., Forster, C., Baumli, N., Gallego, G., and Scaramuzza, D. (2015). "Lifetime estimation of events from dynamic vision sensors," in *2015 IEEE international conference on Robotics and Automation (ICRA)* (Seattle, WA: IEEE), 4874–4881. doi: 10.1109/ICRA.2015.7139876

Mueggler, E., Huber, B., and Scaramuzza, D. (2014). "Event-based, 6-dof pose tracking for high-speed maneuvers," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Chicago, IL: IEEE), 2761–2768. doi: 10.1109/IROS.2014.6942940

Ni, Z., Bolopion, A., Agnus, J., Benosman, R., and Regnier, S. (2012). Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics. *IEEE Trans Robot.* 28, 1081–1089. doi: 10.1109/TRO.2012.2198930

Oliver, N. M., Rosario, B., and Pentland., A. P. (2000). A bayesian computer vision system for modeling human interactions. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 831–843. doi: 10.1109/34.868684

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

Orchard, G., Meyer, C., Etienne-Cummings, R., Posch, C., Thakor, N., and Benosman, R. (2015b). *Hfirst: A Temporal Approach to Object Recognition*. TPAMI. doi: 10.1109/TPAMI.2015.2392947

Park, P. K., Lee, J. H., Shin, C. W., Ryu, H. S., Kang, B. C., Carpenter, G. A., et al. (2012). "Gesture recognition system based on Adaptive Resonance Theory," in *Proceedings - International Conference on Pattern Recognition* (Tsukuba).

Park, P. K., Lee, K., Lee, J. H., Kang, B., Shin, C. W., Woo, J., et al. (2015). "Computationally efficient, real-time motion recognition based on bio-inspired visual and cognitive processing," in *Proceedings - International Conference on Image Processing, ICIP* (Quebec City, QC), 932–935. doi: 10.1109/ICIP.2015.7350936

Pisharady, P. K., and Saerbeck, M. (2015). Recent methods and databases in vision-based hand gesture recognition: a review. *Comput. Vis. Image Understand.* 141, 152–165. doi: 10.1016/j.cviu.2015.08.004

Posch, C., Matolin, D., and Wohlgenannt, R. (2011). A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds. *IEEE J. Solid State Circuits* 46, 259–275. doi: 10.1109/JSSC.2010.2085952

Pradhan, B. R., Bethi, Y., Narayanan, S., Chakraborty, A., and Thakur, C. S. (2019). "N-HAR: A neuromorphic event-based human activity recognition system using memory surfaces," in *Proceedings - IEEE International Symposium on Circuits and Systems* (Sapporo). doi: 10.1109/ISCAS.2019.8702581

Rao, G. A., and Kishore, P. V. (2016). Sign language recognition system simulated for video captured with smart phone front camera. *Int. J. Electr. Comput. Eng.* 6, 2176–2187. doi: 10.11591/ijece.v6i5.11384

Rebecq, H., Ranftl, R., Koltun, V., and Scaramuzza, D. (2019). "Events-to-video: Bringing modern computer vision to event cameras," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA), 3857–3866. doi: 10.1109/CVPR.2019.00398

Reverter Valeiras, D., Orchard, G., Ieng, S.-H., and Benosman, R. B. (2016). Neuromorphic event-based 3d pose estimation. *Front. Neurosci.* 9:522. doi: 10.3389/fnins.2015.00522

Rivera-Acosta, M., Ortega-Cisneros, S., Rivera, J., and Sandoval-Ibarra, F. (2017). American sign language alphabet recognition using a neuromorphic sensor and an artificial neural network. *Sensors* 17:2176. doi: 10.3390/s17102176

Serrano-Gotarredona, R., Oster, M., Lichtsteiner, P., Linares-Barranco, A., Paz-Vicente, R., Gómez-Rodríguez, F., et al. (2009). Caviar: a 45k neuron, 5m synapse, 12g connects/s aer hardware sensory-processing-learning-actuating system for high-speed visual object recognition and tracking. *IEEE Trans. Neural Netw.* 20, 1417–1438. doi: 10.1109/TNN.2009.2023653

Serrano-Gotarredona, T., and Linares-Barranco, B. (2015). Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Front. Neurosci.* 9:481. doi: 10.3389/fnins.2015.00481

Sheik, S., Pfeiffer, M., Stefanini, F., and Indiveri, G. (2013). "Spatio-temporal spike pattern classification in neuromorphic systems," in *Biomimetic and Biohybrid Systems. Living Machines 2013*, Vol. 8064 eds N. F. Lepora, A. Mura, H. G. Krapp, P. F. M. J. Verschure, and T. J. Prescott. Lecture Notes in Computer Science (London; Berlin; Heidelberg: Springer).

Shrestha, S. B., and Orchard, G. (2018). Slayer: Spike layer error reassignment in time. *Adv. Neural Inform. Process. Syst.* 2018, 1412–1421.

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). "Hats: Histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 1731–1740. doi: 10.1109/CVPR.2018.00186

Stauffer, C., and Grimson, W. E. L. (1999). "Adaptive background mixture models for real-time tracking," in *Proceedings 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, Vol. 2, (Fort Collins, CO: IEEE), 246–252.

Valeiras, D. R., Lagorce, X., Clady, X., Bartolozzi, C., Ieng, S., and Benosman, R. (2015). An asynchronous neuromorphic event-driven visual part-based shape tracking. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 3045–3059. doi: 10.1109/TNNLS.2015.2401834

Wang, Q., Zhang, Y., Yuan, J., and Lu, Y. (2019). "Space-time event clouds for gesture recognition: from rgb cameras to event cameras," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)* (Waikoloa Village, HI: IEEE), 1826–1835. doi: 10.1109/WACV.2019.00199

Wang, Y., Du, B., Shen, Y., Wu, K., Zhao, G., Sun, J., et al. (2019). "EV-gait: event-based robust gait recognition using dynamic vision sensors," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Long Beach, CA), 6358–6367. doi: 10.1109/CVPR.2019.00652

Wang, Z., Hou, Y., Jiang, K., Dou, W., Zhang, C., Huang, Z., et al. (2019). Hand gesture recognition based on active ultrasonic sensing of smartphone: a survey. *IEEE Access* 7, 111897–111922. doi: 10.1109/ACCESS.2019.2933987

Won, J. Y., Ryu, H., Delbruck, T., Lee, J. H., and Hu, J. (2015). Proximity sensing based on a dynamic vision sensor for mobile devices. *IEEE Trans. Indus. Electron.* 62, 536–544. doi: 10.1109/TIE.2014.2334667

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 Neuroscience

ORIGINAL RESEARCH
published: 08 May 2020
doi: 10.3389/fnins.2020.00451

Event-Based Eccentric Motion Detection Exploiting Time Difference Encoding

Giulia D'Angelo[1]*, *Ella Janotte*[2], *Thorben Schoepe*[2], *James O'Keeffe*[3], *Moritz B. Milde*[4], *Elisabetta Chicca*[2] and *Chiara Bartolozzi*[1]

[1] Event Driven Perception for Robotics, Italian Institute of Technology, iCub Facility, Genoa, Italy, [2] Faculty of Technology and Center of Cognitive Interaction Technology (CITEC), Bielefeld University, Bielefeld, Germany, [3] Biosciences Institute, Newcastle University, Newcastle upon Tyne, United Kingdom, [4] International Centre for Neuromorphic Systems, The MARCS Institute, Western Sydney University, Sydney, NSW, Australia

Attentional selectivity tends to follow events considered as interesting stimuli. Indeed, the motion of visual stimuli present in the environment attract our attention and allow us to react and interact with our surroundings. Extracting relevant motion information from the environment presents a challenge with regards to the high information content of the visual input. In this work we propose a novel integration between an eccentric down-sampling of the visual field, taking inspiration from the varying size of receptive fields (RFs) in the mammalian retina, and the Spiking Elementary Motion Detector (sEMD) model. We characterize the system functionality with simulated data and real world data collected with bio-inspired event driven cameras, successfully implementing motion detection along the four cardinal directions and diagonally.

Keywords: attentional selectivity, motion detection, eccentric down-sampling, spiking elementary motion detection, bio-inspired visual system, humanoid robotics, event driven

OPEN ACCESS

Edited by:
Gert Cauwenberghs, University of California, San Diego, United States

Reviewed by:
Cornelia Fermuller, University of Maryland, College Park, United States
Francisco Barranco, University of Granada, Spain

***Correspondence:**
Giulia D'Angelo
giulia.dangelo@iit.it

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *14 December 2019*
Accepted: *14 April 2020*
Published: *08 May 2020*

Citation:
D'Angelo G, Janotte E, Schoepe T, O'Keeffe J, Milde MB, Chicca E and Bartolozzi C (2020) Event-Based Eccentric Motion Detection Exploiting Time Difference Encoding. Front. Neurosci. 14:451. doi: 10.3389/fnins.2020.00451

1. INTRODUCTION

Most modern robotic systems still lack the ability to effectively and autonomously interact with their environment using visual information. Key requirements to achieve this ability are efficient sensory data acquisition and intelligent data processing. Useful information about the environment (e.g., how far away an object of interest is, how big it is, whether it is moving) can be extracted from sensory data. More complex interactions, for example locating and retrieving a particular resource, require an attentive system that allows robots to isolate their target(s) within their environment as well as process complex top-down information.

There are a number of ways for autonomous robots and natural organisms alike to gather information about their surroundings. Teleceptive sensors, for example those using ultrasound or infra-red light, are common in engineered systems, and are also exploited by some natural organisms for navigation and object tracking (Nelson and MacIver, 2006; Jones and Holderied, 2007). However, a closer relationship between attention and activation in the visual cortex has been observed by Maunsell and Cook (2002), showing the importance of vision when interacting and being attentive within an environment whilst performing a task. Motion detection, in particular, represents one of the important attentional cues for facilitating agent-environment interactions (Cavanagh, 1992), and is used by natural organisms to avoid obstacles, respond quickly and coherently to an external stimulus within a scene, or to focus attention to a certain feature of a scene (Abrams and Christ, 2003). Due to its wide range of applications, motion detection

has been an area of research for decades and has produced a number of different detection models, ranging from gradient-based algorithms (Lucas and Kanade, 1981; Benosman et al., 2012), over local-plane fitting (Brosch et al., 2015; Milde et al., 2015) and time-to-travel methods (Kramer, 1996) to correlation-based approaches (Horiuchi et al., 1991). Gradient-based methods utilize the relationship between the velocity and the ratio between the temporal and the spatial derivative. Hence, to determine the speed and direction of the motion, the derivation of the spatial and temporal intensity for each pixel is needed. All correlation-based models share the linear and spatio-temporal filtering of measured intensities, which are functions of time and location. The best-known correlation motion detectors are the biologically derived Hassenstein–Reichardt and the Barlow–Levick models (Hassenstein and Reichardt, 1956; Barlow and Levick, 1965). The Hassenstein–Reichardt model was derived from behavioral experiments with beetles, while the Barlow–Levick model was inspired by motion detection in the rabbit's retina. In both cases one elementary motion detection unit is selective to motion in one cardinal direction (preferred direction) and suppresses output to motion in the opposite direction (anti-preferred direction) (Barlow and Levick, 1965). The models themselves (from 1956 and 1964, respectively), are still assumed to describe motion detection in organisms such as fruit flies (Borst et al., 2010; Maisak et al., 2013; Mauss et al., 2014; Borst and Helmstaedter, 2015; Strother et al., 2017). A limitation of correlation-based detectors is that, depending on the time-constant of the filters used, the detector is only receptive to a limited range of velocities. This range can be shifted by varying the parameters but always remains limited.

Environment analysis using traditional frame-by-frame visual processing generally requires a robot to extract and evaluate huge amounts of information from the scene, much of which may be redundant, which hinders the real-time response of the robot. The computational resources required for visual processing can be significantly reduced by using bio-inspired event-based cameras (Lichtsteiner et al., 2008; Posch et al., 2011), where the change in temporal contrast triggers asynchronous events. Event-based cameras perceive only the parts of a scene which are moving relative to themselves. Thus, they are idle until they detect a change in light intensity above a relative threshold. When this happens, the pixel reacts by producing an event characterized by its time of occurrence. Address Event Representation (AER) protocol allows the asynchronous readout of active pixels while providing information on the event polarity and the pixel location. As such, the camera's output are ON-events for increments in temporal contrast and OFF-events for decrements. Optical flow, the vector representation of the relative velocity in a scene, has a wide range of uses, from navigation (Nelson and Aloimonos, 1989; Milde et al., 2015), to predicting the motion of objects (Gelbukh et al., 2014). We propose that these models can also be used to direct attention toward moving objects within a scene. Recent studies have developed event-based motion detection for optical flow estimation both relying on conventional processing architectures (Benosman et al., 2012, 2014; Gallego et al., 2018, 2019; Mitrokhin et al., 2018) and unconventional neuromorphic processing architectures (Giulioni et al., 2016; Haessig et al., 2018; Milde et al., 2018). Even though the former mechanisms, which leverage standard processing capabilities, show real-time optic flow estimation with very high accuracy, they are not suited for spiking neural networks and neuromorphic processors. This is due to the way information is represented, using real values in these algorithms. Additionally, the power consumption and computational complexity in Gallego et al. (2018, 2019) is too high for constrained robotic tasks. The neuromorphic approaches on the other hand can naturally interact with spiking networks implemented on low-power neuromorphic processing architectures as information is encoded using events.

In the last decade a number of spike-based correlation motion detectors have been introduced (Giulioni et al., 2016; Milde et al., 2018). Of particular interest to this work is the spiking elementary motion detector (sEMD) proposed by Milde et al. (2018). The sEMD encodes the time-to-travel across the visual field as a number of spikes (where time-to-travel is inversely proportional to velocity). The sEMD's functionality has been evaluated in Brian 2 simulations and on SpiNNaker using real-world data recorded with the Dynamic Vision Sensor (DVS) (Milde et al., 2018; Schoepe et al., 2019). Furthermore, the model has been implemented on a neuromorphic analog CMOS chip and tested successfully. The implementation on chip presents a low latency and low energy estimate of locally occurring motion. It further offers the advantage of a wider range of encoded speeds as compared to the Hassenstein-Reichardt model, and it can be tuned to different working ranges in sympathy with the desired output. Event-driven cameras, compared with classic frame-based cameras, dramatically reduce the computational cost in processing data, however they produce a considerable amount of output events due to ego-motion. Previous implementations of the sEMD have applied a uniform down-sampling across the camera's visual field. However, recent studies have found that motion detection performance depends strongly on the location of the stimulus on the retina, due to the non-uniform distribution of photoreceptors throughout the mammalian retina (Traschütz et al., 2012). Rod and cone density in the mammalian retina is high at the fovea, and decreases toward the periphery. The non-uniform distribution of photoreceptors in the retina has a strong role in speed discrimination, and it should be taken into account as an important factor in motion estimation. Taking inspiration from the mammalian visual system (Freeman and Simoncelli, 2011; Wurbs et al., 2013), where Receptive Fields (RFs) linearly decrease in size going from the retinal periphery toward the fovea (Harvey and Dumoulin, 2011), we propose an *eccentric*, space-variant, down-sampling as an efficient strategy to further decrease computational load without hindering performances. A good approximation of the mammalian space-variant down-sampling is the log-polar mapping, describing each point in the 2D space as logarithm of the distance from the center and angle. Given its formalized geometrical distribution, the log-polar mapping provides algorithmic simplification and computational advantages, for example for tasks such as moving a robot's cameras toward a desired vergence configuration (Panerai et al., 1995), or binocular tracking Bernardino and Santos-Victor (1999). Recently, the log-polar approach has been studied also

for event-driven cameras, with the proposal of the Distribution Aware Retinal Transform (DART) (Ramesh et al., 2019). Although the log-polar representation would better suit the implementation of the eccentric down-sampling, the results in polar dimension would not be comparable with the classic down-sampling of the sEMD with Cartesian coordinates. For benchmarking purposes, in this paper we use an approximate implementation of the mammalian space-variant resolution, based on Cartesian coordinates.

In this work, we propose a novel approach to spiking elementary motion detection, exploiting the non-uniform retina model as a down-sampling of the visual field. By combining the sEMD with eccentric down-sampling, this work aims to improve the computational efficiency of the motion computation and take a step toward a bio-inspired attention model where information at the center of the field of view is of higher resolution and more heavily weighted than information at the periphery, allowing robots to exploit visual information to effectively interact with their environments in real time. The proposed architecture is suitable for simulation on neuromorphic platforms such as SpiNNaker (Furber et al., 2014), and offers the possibility to be easily implemented for recorded and live input data. To the authors' knowledge, artificial motion detectors with eccentric filtering of the visual field are a novel approach to motion detection. Link to the authors' repository containing the model and the data: https://github.com/event-driven-robotics/sEMD-iCub.

2. METHODOLOGY

The proposed work integrates bio-inspired eccentric down-sampling with the sEMD (Milde et al., 2018). Our aim is to further decrease the computational resources required, by filtering the number of incoming events into the visual field, while maintaining a fine resolution in the center of the visual field.

2.1. Eccentric Down-Sampling

Several physiological studies have explored the mammalian retina topography such as the blind spot, fovea and eccentricities (Wässle and Riemann, 1978), showing that receptive fields are uniformly overlapped in the mammalian retina (Devries and Baylor, 1997). The proposed eccentric down-sampling approximates the two-dimensional circular retina onto a square, maintaining a quadrilateral camera resolution (**Figure 1B**), where each RF spatio-temporally integrates the information within its area of sensitivity. The RF size of the squared

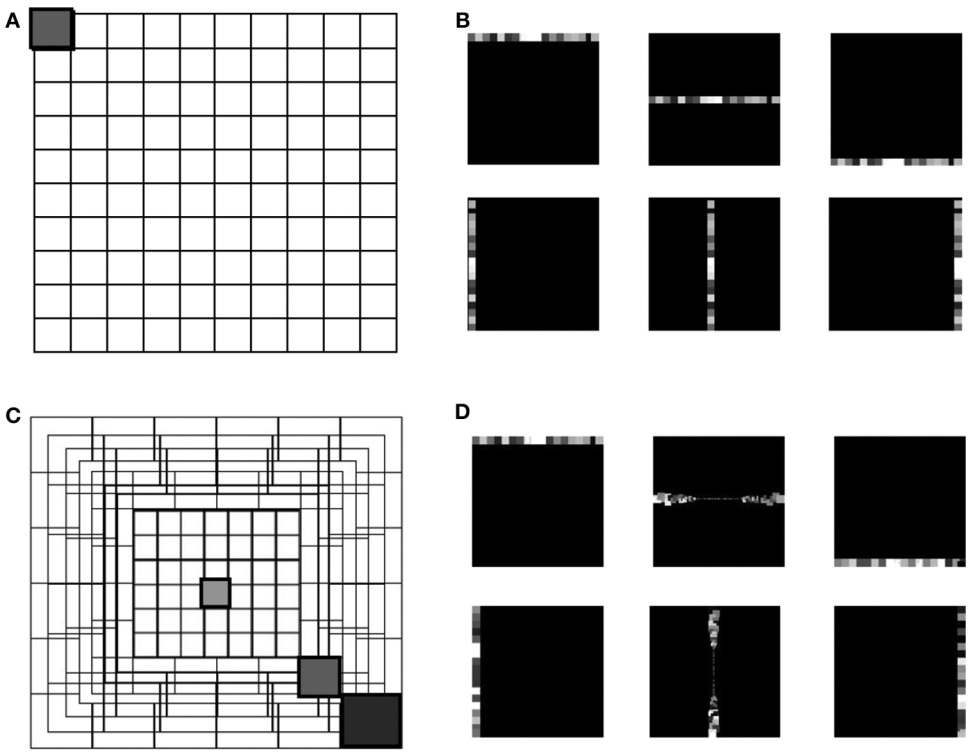


FIGURE 1 | The grid in **(A)** represents the uniform down-sampling of the visual field in equal matrices of *n* by *n*. **(C)** Represents the eccentric down-sampling decreasing the size of the matrices going to the center of the visual field (fovea). This implementation does not include the blind spot present in the mammalian visual system. The three gray squares with varied hues represent three RF sizes at different eccentricities: 0, 39, 70 pixels distant from the center. The square with the same hue in both grids **(A,C)** represents a matrix with equal size in the two down-samplings. Panels **(B,D)** represent the encoding in horizontal and vertical trajectories of the uniform down-sampling **(B)** and the eccentric down-sampling **(D)**. On both top rows of **(B,D)**, an example of the RFs belonging to the first, middle and last horizontal trajectories, and on the bottom row the vertical trajectories is given. All RFs are represented with different gray-scale for the reason of visualization.

approximation decreases linearly toward the foveal region, where each RF is defined by one pixel. All RFs of the same size create a square ring around the foveal region, with each successive ring framing the previous one. The eccentric down-sampling reproduces the RF overlap between RFs of consecutive rings ensuring the robustness in response all over the retina. However, the proposed model does not include the central blind spot present in mammalian retina.

Equations (1) and (2) describe the relationship between the receptive field size (R^s) and its distance from the foveal region, where (R_i^c) is the center of the top left RF of each squared ring and $i = [1, .., n]$ is the number of squared rings over the retinal layer. The term x in Equation (1) represents the x axis of the camera where the origin is placed in the top left corner, $max[\mathbf{R}^s]$ is the maximum kernel size of the outermost peripheral ring, and d_{fovea} is the total distance from the periphery to the edge of the fovea.

$$\mathbf{R}^s(x) = -\frac{max[\mathbf{R}^s]}{d_{fovea}}x + max[\mathbf{R}^s] \quad (1)$$

$$\mathbf{R}_i^c = R_{i-1}^c + \frac{R_{i-1}^c}{2} \quad (2)$$

$$\mathbf{M}_t = M_{t-1}e^{-\frac{dt}{\tau}} + \frac{1}{R_{nf}} \quad (3)$$

Each RF is a matrix of input pixels from the sensor. Every RF is modeled as a leaky integrate and fire (LIF) neuron integrating the information in space and time (Equation 3), where M is the membrane potential of the RF, t represents the temporal information of the incoming event into the RF, dt the difference in time with the previous event in the RF, and τ is the time constant of the exponential decay ($\tau = 1,000ms$). The membrane potential of every RF integrates incoming spikes until it reaches the threshold ($threshold = 1$), which is the same for all RFs. The contribution of each event to the increase in membrane potential of a neuron is normalized with the dimension of the RF. As the activity of the ATIS is sparse, the normalization factor (R_{nf}) is expressed as a percentage of the area of the RF. Every incoming event triggers the updating of the membrane potential by calculating the temporal decay of the membrane since the last event. In addition, the membrane potential is increased by the normalization factor. This way, the response from all RFs is normalized by their occupied space over the visual field. Finally, if the threshold is reached, the neuron emits an output spike. Hence, the response from each RF coherently encodes the input information in relationship with the distance from the fovea.

2.2. The Spiking Elementary Motion Detector (sEMD)

The spiking Elementary Motion Detector (sEMD) depicted in **Figure 2** has been designed for the purpose of encoding optic flow using event-based visual sensors (Milde et al., 2018). The use of event-based sensors is suited to perceiving motion. The edge of an object moving from the receptive field of one pixel to the adjacent one generates a spike in the two pixels with a given time difference, depending on the velocity of the edge and its distance from the pixels. The relative motion or optic flow is inversely proportional to this time-to-travel. An sEMD is composed of two pixels and a time difference encoder (TDE). The TDE encodes the time difference between two pulses into the number of output spikes produced in response to the second input pulse. The number of output spikes encodes the motion flow of objects moving in front of the two pixels.

The synapses connecting the inputs to the TDE are of two types - facilitator and trigger (see **Figure 2** fac and trig). The facilitator synapse gates the activity of the TDE neuron. The trigger synapse elicits a response from the TDE neuron only if its input event occurs after the event from the facilitator synapse (compare **Figures 2B,D**). The output current of the trigger synapse increases the TDE neuron's membrane potential as shown in **Figure 2C**). The strength of the current depends on the exponentially decaying gain variable of the facilitator synapse. Therefore, the TDE not only detects the direction of motion but also encodes the velocity of the stimulus in the number of output spikes and time to first spike. The faster the stimulus propagates, the more spikes are produced by the TDE. In order to mitigate the noise present at the output of a silicon retina, a pre-processing filtering stage is used. It consist of neural spatio-temporal filters (SPTCs) used to detect correlated events. Two uniform neighborhoods, of n by n pixels, are connected to a LIF neuron each. The neurons fire once only if within a specific time, defined by their time constant, 66% of the pixels in the neighborhood produce events. The proposed implementation exploits the eccentric down-sampling (Chapter 2.1) replacing the uniform filtering stage previously used with the sEMD model by Milde et al. (2018).

2.3. Experiments

The objective of this work is to quantitatively and qualitatively characterize the output of the TDE population receiving input from the eccentricity filtering layer and to compare it with the TDE population receiving input from a uniform resolution filtering layer. This characterization aims to demonstrate the advantages of our proposed model, namely a decrease in computational load whilst maintaining the ability to estimate the velocity of moving entities within the visual field. To this purpose we characterized and compared the model using moving bars with 1D and 2D motion. In the following, we will refer to the two different implementations as "sEMD with uniform down-sampling" and "sEMD with eccentric down-sampling." The characterization of the proposed motion detection system (**Figure 3**) is achieved using simulated data. Furthermore, additional experiments are undertaken using real input[1] collected with ATIS cameras (Posch et al., 2011) mounted on the iCub robot (see **Supplementary Materials** for real-world data). The simulated data used in this work reproduces the activity of an event driven sensor in response to a bar moving horizontally [Left to Right (LR), Right to Left (RL)], vertically [Top to Bottom (TB),

[1]We explored the real-world applicability of the underlying motion detection mechanism prior to this work in which we demonstrated the functionality of the underlying given variable contrast and event-rates in natural environments (Milde et al., 2015, 2018; Schoepe et al., 2019).

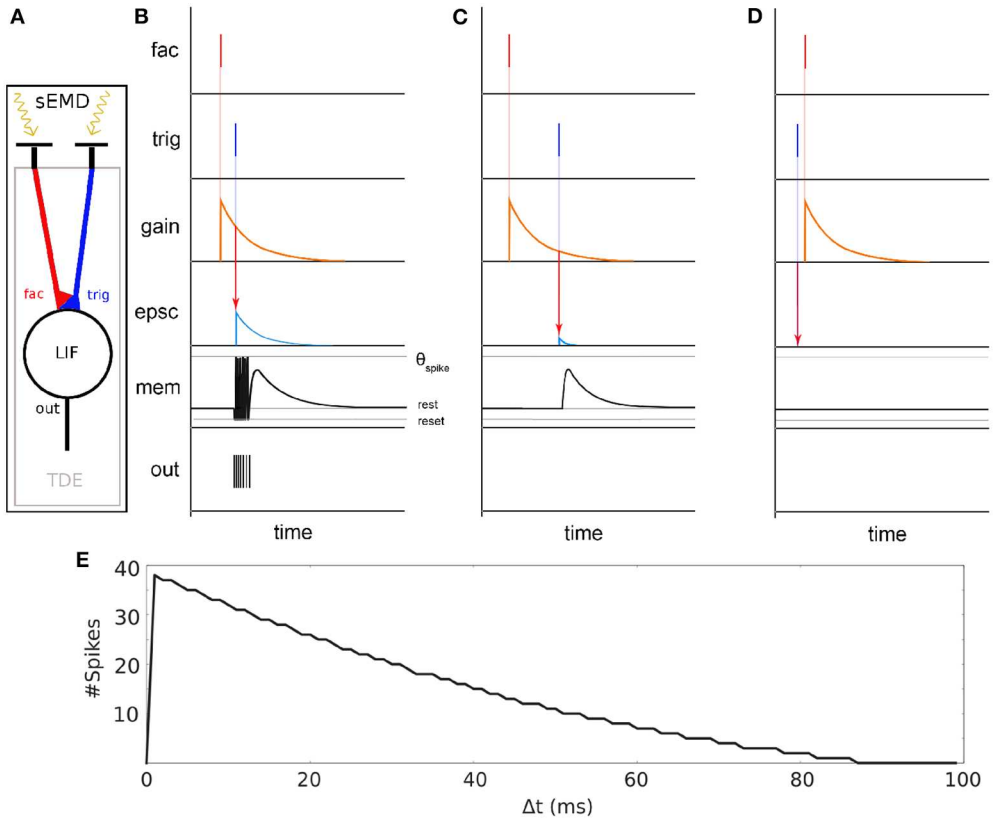


FIGURE 2 | Basic principle of the sEMD (Milde et al., 2018). **(A)** The model consists of an event-based retina sending events into the Time Difference Encoder (TDE). Two adjacent RFs are connected to the facilitation synapse and the trigger synapse, respectively. **(B)** TDE computation for a small time difference Δ t between facilitation event and trigger event. An event at the facilitation synapse generates an exponentially decaying factor called gain. A trigger pulse at the trigger synapse shortly after causes an exponentially decaying Excitatory Post Synaptic Current (EPSC). The EPSC amplitude depends on the gain factor. The EPSC integrates onto the membrane potential (mem). Every time the membrane potential reaches the spiking threshold (τ_{Spike}) an output digital pulse is produced. **(C)** Similar to **(B)** part but with high Δ t. **(D)** Similar to **(C)** but the trigger pulse arrives before facilitation pulse. No output spikes are produced for negative time differences. **(E)** TDE output spike response over time difference Δ t between facilitation event and trigger event.

Bottom to Top (BT)] and transversely, i.e., along the diagonal of the Cartesian plane.

Firstly, we recorded the activity of the sEMD with uniform down-sampling and eccentric down-sampling model, while the speed of the input bar ranges from 0.01 to 1 px/ms, in accordance to the experiments of Giulioni et al. (2016). This ideal input allows a comparison of the two model's spike raster plots and mean population activities.

We first analyzed the selectivity of all sEMDs tuned to the same movement direction, measuring the mean firing rate (MFR) of the whole population. Given the symmetrical connectivity of the sEMD neurons along the eccentric visual field, the responses from the population of LR, RL, TB, and BT sEMD neurons are expected to be comparable, responding with a large MFR to a stimulus moving along their preferred direction and being unresponsive to a stimulus moving along their anti-preferred direction.

Further investigations focus on a single population and its response to its preferred stimulus direction (from left to right, or top to bottom), assuming transferable responses for the other directions.

A deeper understanding of the temporal response from the neurons was achieved by collecting the spike raster plots for nine speeds of the chosen range: (0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 1 px/ms), respectively.

For each speed, we analyzed the response of each sEMD in the population, mapping its MRF onto the Cartesian space and visualizing spatial rather than temporal information. We analyzed how the Mean Firing Rate (MFR) of each sEMD changes with speed and distance from the center of the field of view. Additional experiments have been performed changing the length of the stimulus, by recruiting more sEMDs, should increase the MFR of the whole population tuned to the corresponding stimulus direction. Eventually, we analyzed the response of the model to a bar moving transversally exploring the response from the population to 2D motion. In such a case, the stimulus does not elicit the maximum response of any sEMD, rather, it elicits intermediate activity in more than one sEMD population, that need to be combined to decode the correct input direction.

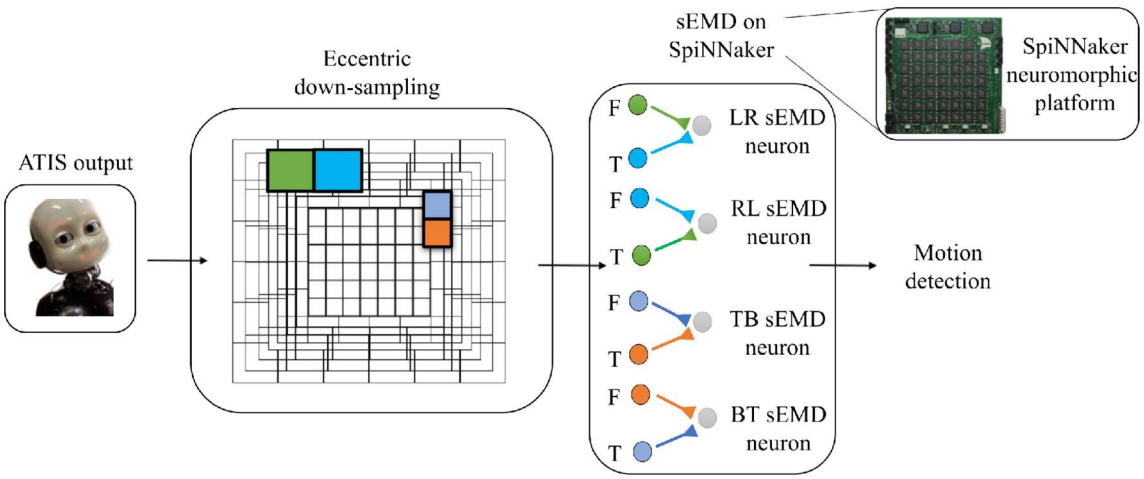


FIGURE 3 | Basic scheme of the pipeline. From left to right the ATIS output is processed by the eccentric down-sampling model and sent to the sEMD model, hosted on SpiNNaker neuromorphic hardware. The sEMD model represents the layer of neurons producing spikes and encoding the motion detection. The eccentric down-sampling and the sEMD model representation show the spatio-temporal filter neurons (green, blue, violet, and orange square), the facilitator and the trigger, both synaptically connected to the sEMD neuron. Facilitators (F) and triggers (T) are shown for LR sEMD neuron, RL sEMD neuron, TB sEMD neuron, and BT sEMD neuron.

2.4. Experimental Setup

In all experiments the model was simulated on a SpiNNaker 5 board hosting 48 ARM-chips, each with 18 cores. The SpiNNaker architecture supports highly parallelized asynchronous simulation of large spiking neural networks in almost real-time. The aspect of real-time computation is of utmost importance for the interaction of the robot with the environment. For the implementation of the SNN we chose 160×160 pixels as a retinal layer resolution, to limit the number of neurons to be simulated on SpiNNaker and to further minimize the impact of the residual distortion in the fringes of the camera after calibration. The output of the retinal layer serves as input to the uniformly and eccentrically down-sampled filtering layer, respectively. For the uniform down-sampling sEMD, we chose a non-overlapping neighborhood matrix size of 4×4 ATIS pixels to represent one RF. This filtering layer is simulated on SpiNNaker and consists of 1600 LIF neurons. It receives input from a SpikeSourceArray, containing the respective ATIS pixel spike times. The synaptic weight of the connections is 0.3. In contrast, the fovea (1 RF = 1 pixel) of the eccentric down-sampling covers 10% of the total retinal layer, and the biggest receptive field has a dimension of 10×10 pixels with a normalization factor of 60% (Equation 3). The population is made up of 8836 LIF neurons. The eccentric down-sampling occurs locally before the spike times of the respective receptive fields are transferred to SpiNNaker in a SpikeSourceArray. The final layer of the network consists of four sEMD populations sensitive to local motion in one cardinal directions, respectively, using sEMD neuron model included in the extra models of the pyNN library. The sEMD populations were connected to the filtering layers along the trajectories as shown in **Figure 3**. The combination of the output of the four populations allows the encoding of transversal stimuli. Each population shares the size of the down-sampling population. For both down-sampling approaches all sEMD neuron and synapse parameters are the same. The connectivity of the respective sEMD populations are displayed in **Figure 3**. The synaptic weights are 0.3 and the synaptic time-constants τ_{ex} and τ_{in} are both 20 ms. The neuron parameters amount to: a membrane capacitance of 0.25 nF, and time-constants τ_m and τ_{rf} of 10 ms and 1 ms, respectively. The reset, resting and threshold voltage of the neurons are defined as −85, −60, and −50 mv, respectively. To avoid a response of the sEMD-populations perpendicular to the preferred direction, in case of a bar moving their facilitator and trigger synapses receive input at the same time, the input to the facilitator synapse was delayed by 1 ms.

3. RESULTS

Our investigation starts with the characterization of the eccentric down-sampling sEMD's response to a simulated bar moving in the four cardinal directions with a speed of 0.3 px/ms: left to right, right to left, top to bottom and bottom to top. **Figure 4** shows the response to stimuli moving in the preferred and anti-preferred directions at fixed velocity 0.3 px/ms (the middle of the regarded velocity range). In particular, **Figure 4A** shows the mean instantaneous firing rates of the preferred and anti-preferred direction populations. The preferred directions are colored in red and the anti-preferred directions in blue. As expected, the preferred direction population's response is significantly higher than the response of the anti-preferred direction population. Furthermore, as expected the response from all the populations to the respective preferred direction is similar in terms of instantaneous firing rate and mean firing rate, and comparable among each other, thus validating the assumption that the response to stimuli in the preferred direction is similar for all of the populations. Assuming a bar moving across the retina at a constant speed, the high variances in

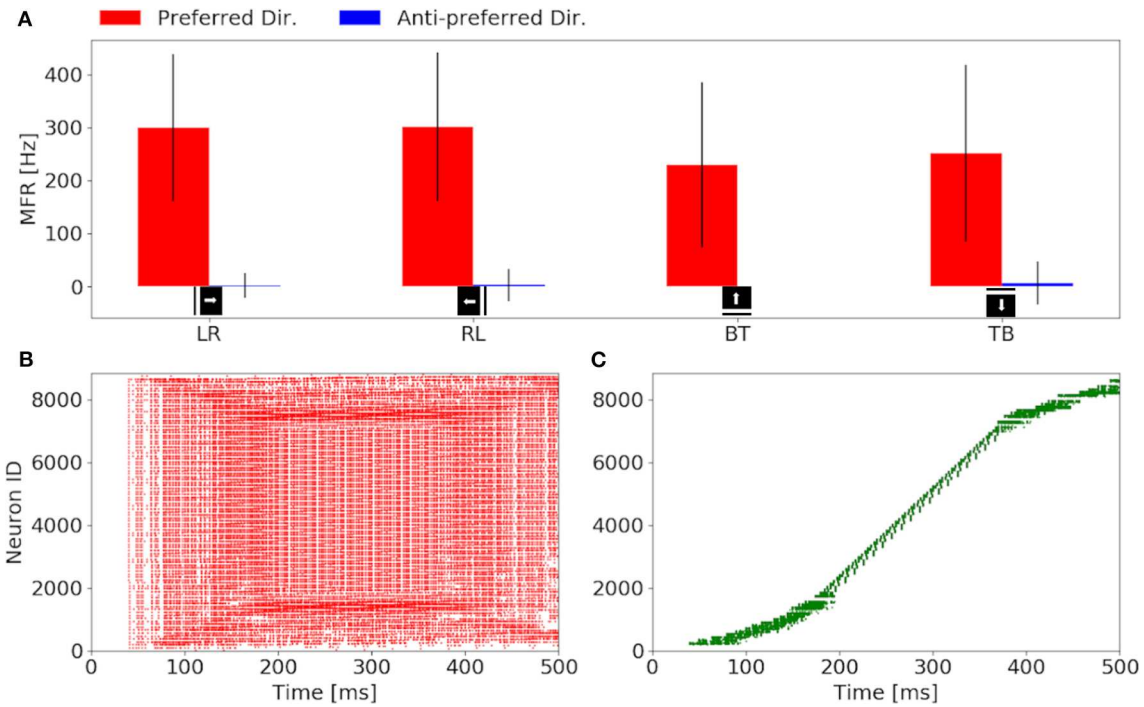


FIGURE 4 | Response of the sEMDs with eccentric down-sampling to a simulated bar moving with a speed of 0.3 px/ms: **(A)** Instantaneous MFR and variance of the four sEMD-populations, each tuned to one of the four cardinal directions, to the preferred and anti-preferred stimulus. Similarly to the sEMD with uniform down-sampling, the response to the anti-preferred stimulus is negligible with respect to the response to the preferred direction stimulus. **(B)** Raster plot of the left to right (LR) population in response to a vertical bar moving from left to right. In the first 100 ms, the difference in the size of the RF can be seen, as the active neurons spike with different spike rates and the number of active neurons increases with time, when the bar moves closer to the fovea. **(C)** Raster plot of the top to bottom (TB) population in response to an horizontal bar moving from top to bottom. The sigmoidal shape arises from the geometry of the eccentric down-sampling and the neurons' indexing.

preferred and anti-preferred directions can be explained by the difference in receptive field sizes in our proposed model (see **Figure 1**). Depending on the stimulus speed, the size of the RF determines a period of time in which the stimulus moves over the RF. Thus, for the same stimulus speed, a peripheral RF takes more time to respond than one in the foveal region, leading to a different RF rings having a different sensitivity to stimulus speed. Only the RFs along the same squared ring have the same sensitivity to the same speed. If a bar is moving across the visual field at a certain speed, only neighbor RFs, that produce spikes able to trigger the TDE neurons, will detect the stimulus. Consequently, due to the varying RF sizes and varying speed sensitivities, the size of the RF relative to its neighbor affects the response of the TDE. This causes the visual field to respond non-uniformly. **Figures 4B,C** show examples of characteristic raster plots of the preferred direction populations, in response to a bar stimulus moving horizontally and vertically, respectively. The color-coding indicates the direction sensitivity of the population: left to right (red) and top to bottom (green). The first response to the horizontal and vertical bar movement (**Figures 4B,C**), is delayed by 40 ms. This is due to the stimulus taking 30 ms (speed of 0.3 px/ms) to travel over the first peripheral RF (10 × 10 px), before reaching the RF connected to the trigger. In the first 50 ms of reaction to the stimulus, the resulting spike density is rather sparse, caused by a lower response from the peripheral RFs (sensitive to higher speeds). Conversely, from 150 to 400 ms, the time where the stimulus is expected to cross the fovea, the spike density is higher because the RFs at the fovea are of a size more suited to the stimuli velocity. The impact of the proposed model is more clearly visible in response to the vertically moving stimulus (**Figure 4C**). The mapping from the eccentric receptive fields to the neuron IDs transforms the time sequence of a vertical bar response to a sigmoid. By contrast, the output of the sEMD with uniform down-sampling resembles the shape of stairs, with each row activated after one another, spiking with the same rate. The non-uniform size of the RFs in our proposed model is again the cause for the different spike densities produced in response to the stimulus moving at constant velocity. In this experiment the sEMDs successfully encode the direction of the bar stimulus moving across the visual field in all the four cardinal directions, showing a negligible response to the anti-preferred direction. This therefore shows that the eccentric down-sampling preserves the ability of the sEMD populations to encode optic flow of moving stimuli.

A comparison of the MFR for all populations of the uniform down-sampling model and the eccentric down-sampling model in response to a simulated stimulus moving from left to right at different velocities is shown in **Figure 5**. The color-coding

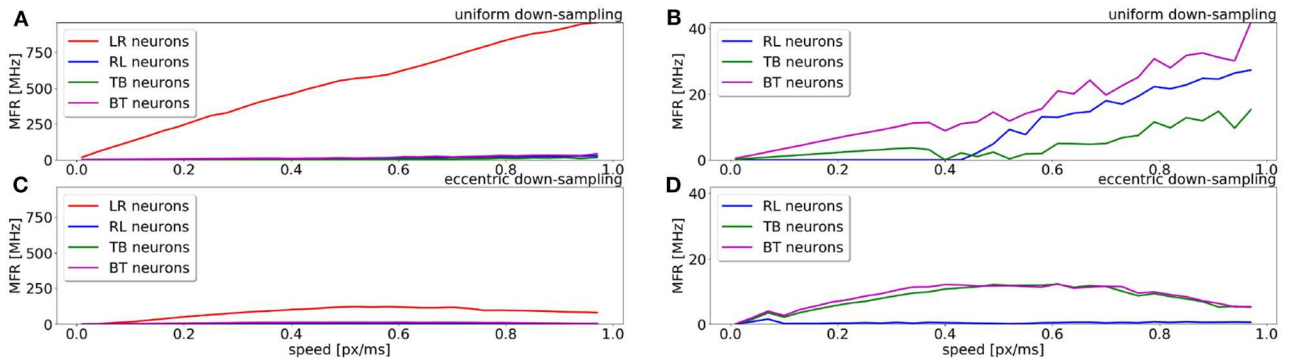


FIGURE 5 | Comparison of the sEMD model with the uniform down-sampling **(A,B)** and the eccentric down-sampling **(C,D)** in response (MFR) to a left to right moving bar (simulated data). The preferred direction is displayed in red (LR), with the anti-preferred direction in blue (RL). The response for the top to bottom (TB) and bottom to top (BT) populations are displayed in green and magenta, respectively. Panels **(B,D)** are a magnification for the anti-preferred direction (right to left) and the incorrect directions (top to bottom and bottom to top) of the panels **(A,C)**. The Figure compares the behavior from the populations of the two approaches to the same stimulus and over the same range of speeds.

remains the same as in **Figures 4B,C**, additionally the response of the populations selective to stimuli from right to left and bottom to top is depicted in blue and magenta, respectively. **Figure 5A** shows the behavior of the uniform down-sampling model, and **Figure 5C** depicts the behavior of the eccentric down-sampling model. Both methods show a trend of increasing MFR until target velocity reaches 0.6 px/ms. While the response from the sEMD with uniform down-sampling keeps increasing after 0.6 px/ms, the firing rate of the population with eccentric down-sampling gradually reduces as the target velocity approaches 1.0 px/ms. The same trend can also be seen for targets moving in the anti-preferred direction. **Figure 5** shows that, while the sEMD response of the anti-preferred (right to left) and the incorrect directions (top to bottom and bottom to top) of the uniform down-sampling model (**Figure 5B**) linearly increases until 1.0 px/ms, the output firing rate of the proposed eccentric down-sampling model (**Figure 5D**) increases for target speeds up to 0.5 px/ms and decreases thereafter. Despite the number of sEMDs required for the proposed model (8,836 per population) being significantly higher than for the uniform down-sampling (1,600 per population) under the same setup conditions, the eccentric sEMDs' down-sampling shows an overall significant decrease in the mean output firing rate of the whole population in response to the same stimulus. Differently from frame-based systems, where the number of operations—and hence power consumption—depend on the number of filters, in event-driven spiking architectures, filters are active (and consume power) only when they receive input spikes and produce output spikes. **Figure 5** shows that the proposed eccentric down-sampling model is able to differentiate between stimulus in preferred and anti-preferred directions more efficiently than a model with uniform down-sampling, without sacrificing performance. The proposed model still maintains an order of magnitude difference between MFR for stimulus in the preferred direction vs. anti-preferred direction. Although the eccentric down-sampled model does not allow for an inference of stimulus velocity to be made based on the MFR of the entire population, the same information can be extracted based on the eccentricity of the RFs with the greatest MFR.

The response from sEMDs selected at different eccentricities (at 0, 39, and 70 pixels distant from the center) is examined in **Figure 6** in relation to the same speed range. In the original model (Milde et al., 2018) the MFR of all three neurons would increase proportionally to the target speed. **Figure 6** shows that the speed encoding for our proposed model depends on the RF size, because the integration time for each RF size corresponds to a specific range of velocities. This leads to a specific range of time-differences between two connected RFs. Each sEMD has a speed limit, which depends on its tuning, above which it will be unable to detect motion. **Figure 2E** shows the TDE output spikes over time difference. If a trigger event occurs before the output of the facilitation event has had time to reach the minimum threshold required, the sEMD will not fire. Due to the varying sensitivity of different RF sizes and enhanced by the 1 ms synaptic delay of the facilitator synapse, while the response from the foveal region (0 px distance) drops to zero for speeds higher than 0.7 px/ms, the response from the neuron with a middle eccentricity (39 px distance) begins to decrease dramatically at 0.9 px/ms. The response from the peripheral neuron keeps increasing until the end of the examined speed range (1.0 px/ms). A possible explanation for the relatively low MFR of the peripheral neuron is the increased number of events needed to trigger the RF and its specific sensitivity to higher speeds. **Figure 6** shows how the RF size affects the behavior of the correspondent neuron, obtaining a wider operative range from the whole population. In comparison, uniform down-sampling where all the RF sizes are the same provides a comparatively limited operative range.

The spike raster plots (**Figures 4B,C**) provide the temporal response from the population but they do not provide any spatial information. The visualization in **Figure 7** maps the response of the sEMDs to the corresponding x and y locations for three different speeds: slow (0.03 px/ms, **Figure 7A**), medium (0.3 px/ms, **Figure 7B**) and fast (1.0 px/ms, **Figure 7C**). The data displayed in **Figure 7B** corresponds to the spike raster

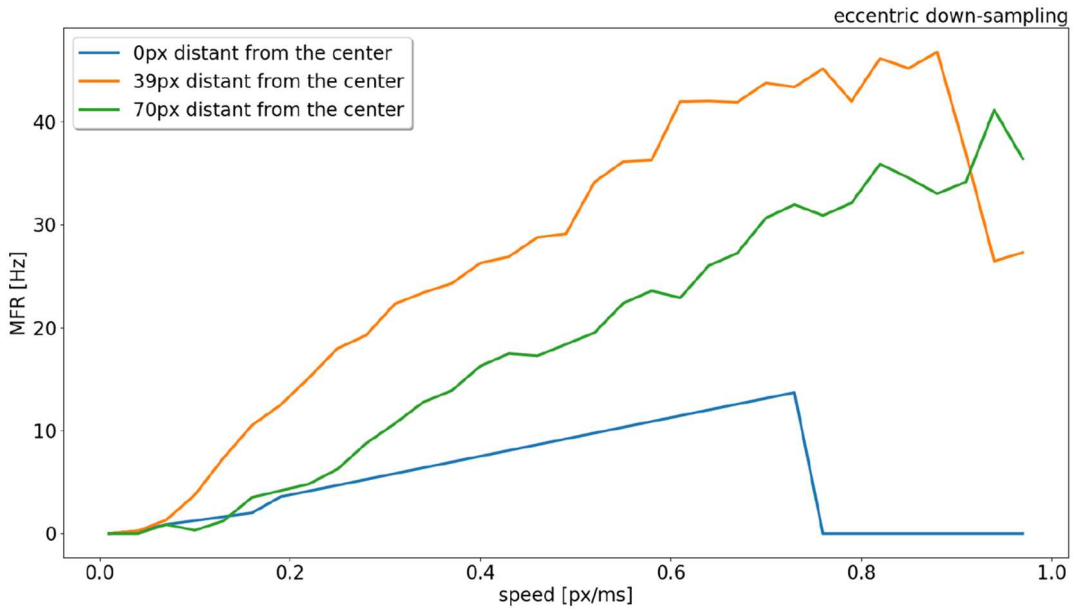


FIGURE 6 | Response (MFR) to a left to right moving bar (simulated) from RFs (eccentric down-sampling) of the central horizontal line of the visual field at different eccentricities (distances from the center of the field of view). In blue, orange and green at 0, 39, and 70 pixels distant from the center, respectively (see **Figure 1**).

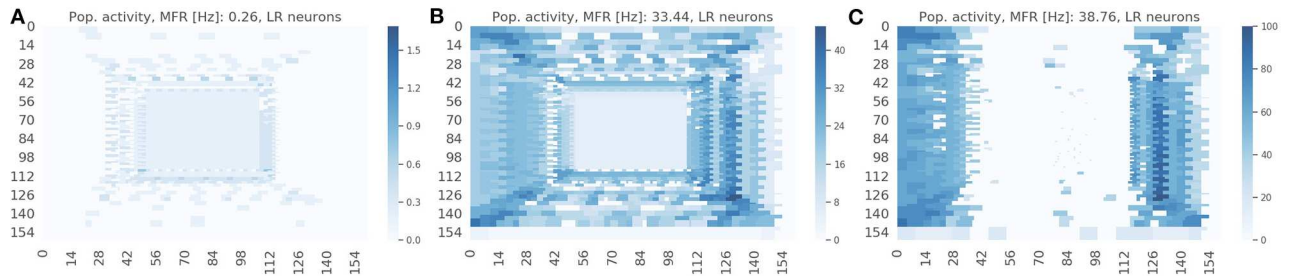


FIGURE 7 | Response from the population of sEMDs with the eccentric down-sampling mapped into the cartesian space with a camera resolution of 160 × 160 pixels. The color-code heatmap represents the MFR of each RF. The stimulus was a bar moving (simulated data) from left to right with constant speed: 0.03 **(A)**, 0.3 **(B)**, 1.0 **(C)** px/ms, respectively.

plot in **Figure 4B**. **Figure 7** shows that the MFR of the whole population increases in relation to the speed: 0.26, 33.44, 38.76 Hz, respectively. The spatial visualization highlights the function of the eccentric down-sampling. As proposed by Traschütz et al. (2012), the slow speeds are detected primarily in the foveal region, where RFs have the smallest dimension and are closest to one another (**Figure 7A**). As the stimulus speed increases, the peripheral region starts responding from the first squared ring around the foveal region (**Figure 7B**) to the rings with the largest RF size for the fast speed (**Figure 7C**).

The response for each RF square ring is different for horizontal and vertical components (most obvious example being in **Figure 7C**. This is because the sEMDs in this case are only connected horizontally (as we are working with left-right motion). Therefore, at the left and right peripheries, there is a descending and ascending scale of RF sizes approaching and moving away from the foveal region, respectively. A concentrated region of diverse, overlapping connected RFs improves the likelihood of the sEMDs picking up the stimulus motion. This does not exist in the regions above and below the fovea, in which each RF will only be connected to horizontally adjacent RFs of the same size, hence the relatively low MFR in these regions.

The response on the right side of the visual field is attenuated in **Figures 7B,C** because the sEMDs from the last RF ring are not connected with any subsequent facilitator (although this does not cause a problem in detecting stimuli entering the scene).

As shown in **Figure 7**, the RF-ring of maximal response appears to move toward the periphery with increasing velocities. **Figure 8** shows the mean and variance of the MFRs at different eccentricities for velocities 0.03, 0.3, and 1.0 px/ms, **Figures 8A–C**, respectively. It is clearly distinguishable, that the maximal response in MFR shifts toward the periphery with increasing velocities.

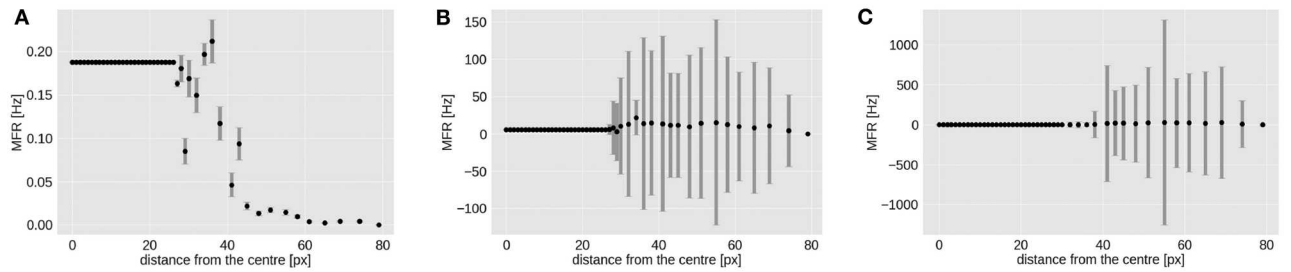


FIGURE 8 | Mean and variance in MFR of RFs at different distances from the center of the visual field. The stimulus is a moving bar (simulated data) going from left to right at speeds of: 0.03 **(A)**, 0.3 **(B)**, 1.0 px/ms **(C)**.

The higher variances observed at greater eccentricities (distance from the center) in **Figures 8B,C**, can be explained by the different RFs response from the horizontal and vertical component of the squared rings (which can be seen in **Figure 7**). The low MFR at 29 pixels (**Figure 8A**) from the center (fovea region from 0 to 28 px) can be explained by the connections between RFs of the first peripheral squared ring (about 3 × 3 px) and the fovea, where each RF has a dimension of 1 px. This sudden increase in size leads to a delay in response from the TDE receiving input to the trigger synapse from the larger receptive field.

To compare the trend of the RFs' peak response increasing in eccentricity with increasing stimulus speed, the center of mass of the RFs response is plotted in relation to the speed range, from 0.01 to 1.0 px/ms (see **Figure 9**). **Figure 9** shows that for low speeds (0.01–0.06 px/ms) the center of mass of the RFs' response shifts from 0 to 27 pixels (distance from the center). The center of mass then plateaus from 0.06 to 0.6 px/ms, where only the RFs of the edges of the foveal region respond to the stimulus. For higher speeds (from 0.6 to 1.0 px/ms), the eccentricity of the center of mass of RF responses starts to increase again, due to a lack of activity in the fovea. The center of mass of RF responses eventually shifts to the periphery, reaching a distance of 49 px from center.

A comparison of the MFR of the sEMD with uniform down-sampling and eccentric down-sampling has been explored with simulated data. **Figure 10** shows the difference in response, normalized for the total number of neurons, from all populations of sEMD neurons with uniform down-sampling and eccentric down-sampling. Even though the uniform down-sampling model has fewer neurons than the eccentric down-sampling model (1,600 compared to 8,836 neurons, respectively) the MFR from the eccentric down-sampling is considerably less at each explored speed, increasing computational and power efficiency.

Figure 11 shows the MFR from the population of LR sEMD neurons in response to a stimulus moving from left to right, at a medium speed of 0.3 px/ms, with bars of varying lengths: 10, 50, 100, and 160 pixels, respectively. The plot shows a positive correlation between the size of the bar and the response from the neurons sensitive to the corresponding direction. **Figure 11** shows that the MFR increment decays as the length of the bar increases - most noticeable when comparing the difference in MFR between the 50 and 100 px bar, and that between the 100 and 160 px bar. This is because the bar is vertically centered in the visual field, and so longer bars cover more of the peripheral region—where each RF requires a greater number of events in order to be activated. Finally, **Figure 12** shows the behavior of the population to a bar moving transversely, revealing the response of the model to 2D motion. **Figure 12A** shows the response to a bar moving from the top left corner to the bottom right, **Figure 12B** from the top right corner to the bottom left, **Figure 12C** from the bottom left to the top right corner and **Figure 12D** from the bottom right corner to the top left.

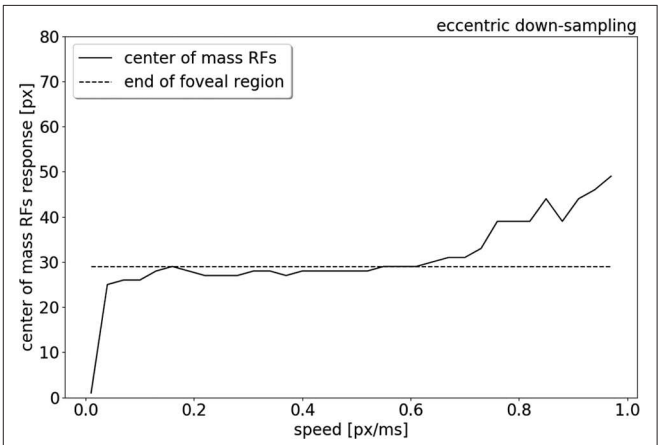


FIGURE 9 | Center of mass (solid line) of the neurons response location to a left to right moving bar (simulated data), from 0.01 to 1.0 px/ms. The dash line indicates the end of the foveal region.

All the explored cases report a similar response from two kind of sEMD populations and a response close to zero from the other neurons. The combination of the responding sEMD neurons successfully detects the transverse motion, showing similar MFR values of the neurons that actively respond.

4. DISCUSSION

The biological role of detecting temporal changes comprise two mechanisms: the detection of fast and slow movements. The first one to identify an entering stimulus into the scene and the latter one to recognize its spatial structure (Murray et al., 1983). Sudden

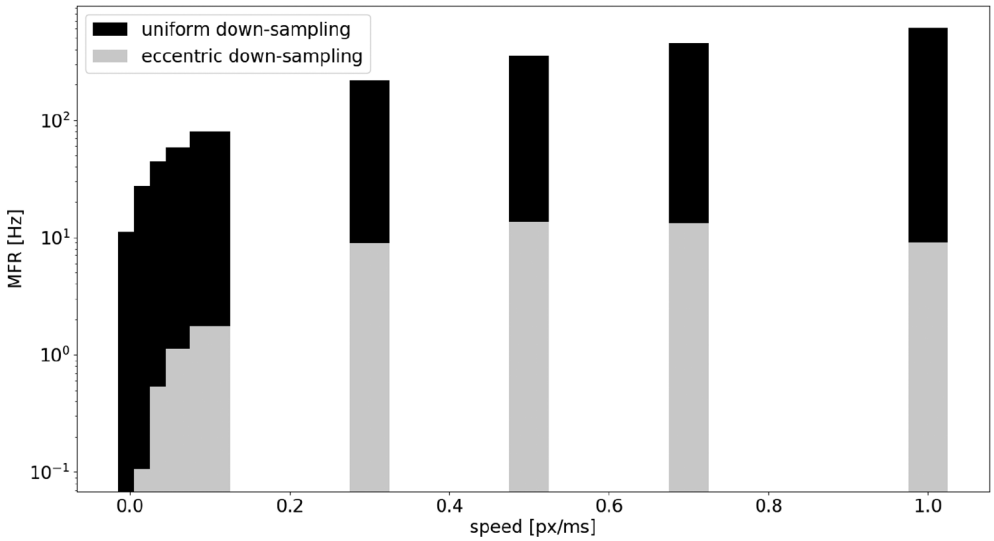


FIGURE 10 | Comparison, between the sEMD model with the uniform down-sampling (1,600 neurons) and the eccentric down-sampling (8,836 neurons), of MFR from the LR sEMD neurons in response to a left to right moving bar.

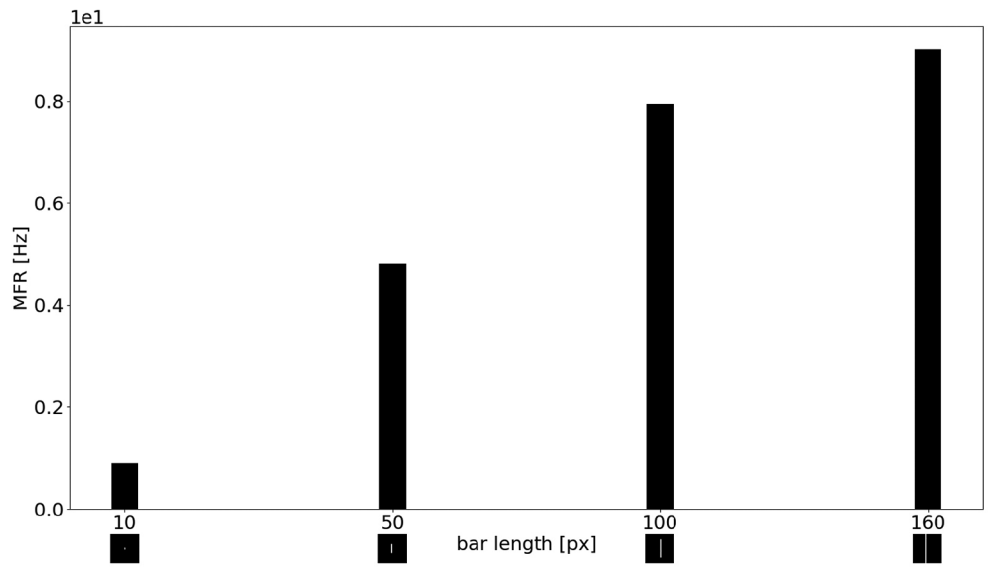


FIGURE 11 | MFR response of the sEMD the LR sEMD neurons for a left to right moving bar at 0.3 px/ms with different bar lengths: 10, 50, 100, 160 pixels, respectively.

onset of motion can attract our attention (Abrams and Christ, 2003, 2005, 2006). Hence, fast movements, speed and acceleration similarly increase our perception of a threat—making it a noticeable stimulus and grabbing our attention (Howard and Holcombe, 2010). Thus, motion detection collaborates with attentional mechanisms to react on time and interact with the surrounding.

In this paper, we have presented a novel implementation of motion detection based on the use of spiking elementary motion detectors coupled with non-uniform down-sampling inspired by the mammalian retina. The proposed model successfully detects the correct direction of an edge moving in the field of view at speeds ranging from 30 to 1,000 px/s, being suitable for the coarse motion processing of robots interacting with the environment (Giulioni et al., 2016).

With respect to the uniform down-sampling implementation presented in the original work (Milde et al., 2018), the eccentricity model significantly decreases the overall activation of each motion detector at every investigated speed. The reduced spiking activity makes this implementation more power efficient even

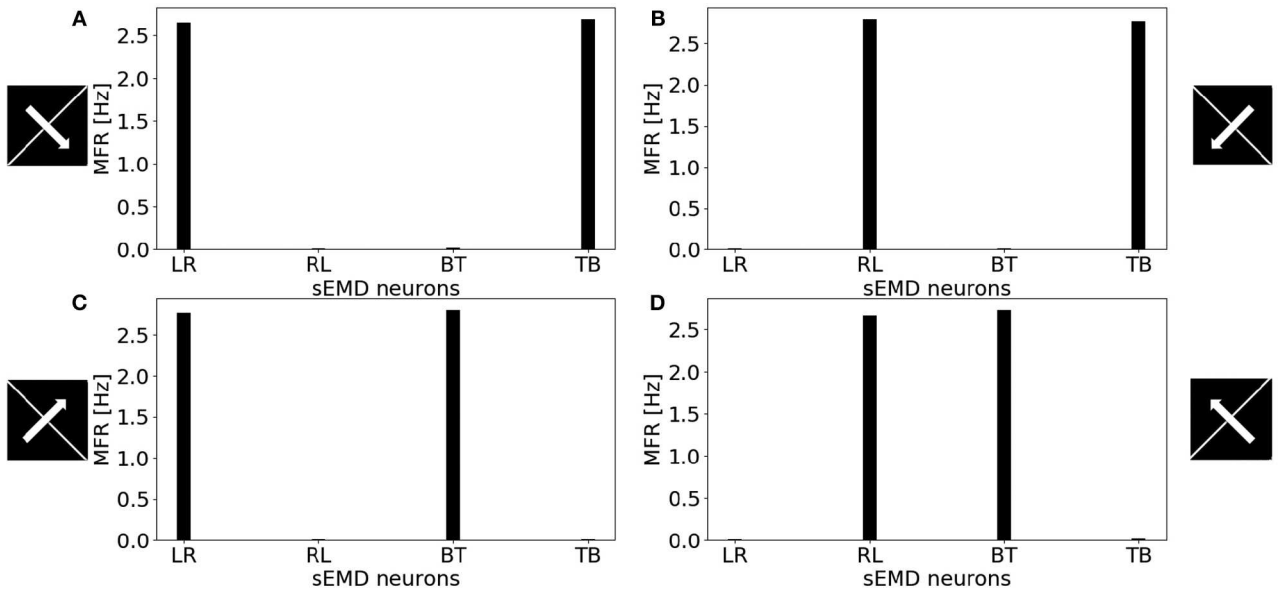


FIGURE 12 | MFR response of the sEMD neurons reacting to a bar moving transversely at 0.3 px/ms. **(A)** Bar moving from the top left corner to the bottom right corner, **(B)** bar moving from the top right corner to the bottom left corner, **(C)** bar moving from the bottom left corner to the top right corner and **(D)** bar moving from the bottom right corner to the top left corner. The length of the bar covers the whole visual field.

in face of an increased number of elementary motion detectors. To achieve the same result in the uniform down sampling implementation, the size of the spatio-temporal filters should be increased, at the cost of a coarser resolution in the whole visual field and a reduced sensitivity to low velocities. The eccentricity implementation overcomes this issue maintaining the sensitivity for low and fast speed – distributed over different regions of the field of view – while significantly reducing the number of incoming events to be processed by the down-stream computational layers.

In the proposed non-uniform down sampling, the elementary motion detectors are tuned to different ranges of speed depending on their position in the field of view. The peripheral sEMDs are characterized by large receptive fields and are hence tuned to higher speeds, that progressively decreases toward the fovea. Hence, the proposed implementation encodes the speed based on the location of the active sEMD. RFs with similar size work in a similar range of speed producing redundant information, and making the decoding of the population activity robust. Moreover, thanks to the sensitivity to high speeds of the peripheral RFs, the detection of objects moving into the visual field is immediate. The sEMDs in periphery will trigger a response to a fast stimulus entering the field of view with extremely low latency. This behavior is desirable in our target scenario, where a robot shall react quickly to fast approaching objects suddenly entering the field of view, and attracting its attention. Furthermore, the combination of RFs with different size, processing events on the same field of vision, allows working with a wider operative range of speeds. In the final application, this motion detection module will be used as one of the feature maps used to compute the salience of inputs in the field of view, directing the attention of the robot to potentially relevant stimuli that will be further processed once a saccadic eye motion will place the salient region in the fovea. A strong and low latency response of peripheral sEMDs to fast stimuli could override the salience of static objects. The characterization of the response of the sEMDs in the non-uniform down sampling shows the same qualitative overall behavior for real-world stimuli, showing robustness to noise and to changing the overall spiking activity of the input. The analysis of the individual responses of the sEMDs at different distance from the fovea shows variability that depends on the discretisation of the receptive fields and on the uneven distribution of the receptive field sizes. This effect possibly depends on the Cartesian implementation of the eccentricity, that approximates the distribution of the receptive fields with a rectangular symmetry. A polar implementation of the same concept will reduce the effects of discretisation and improve the overall population response. In a polar implementation, the direction of each sEMD will be aligned along the polar coordinates (radius and tangent), rather than along the Cartesian directions, further improving the variability in the overall response of individual modules and allowing decoding of stimulus direction beyond the cardinal ones.

DATA AVAILABILITY STATEMENT

The datasets generated for this study can be found in the https://github.com/event-driven-robotics/sEMD-iCub.

AUTHOR CONTRIBUTIONS

GD'A: main author of the manuscript and developer of the software. CB, EC, and MM: supervision

assistance and review. JO'K and TS: review assistance. EJ: assistance during experiments and writing the manuscript.

ACKNOWLEDGMENTS

We thank our colleagues from Italian Institute of Technology, Luca Gagliardi and Vadim Tikhanoff, who provided insight and expertise assisting the research. We would also like to show our gratitude to Jay Perrett for sharing his accurate review. Further thanks are issued to the Faculty of Technology and Center of Cognitive Interaction Technology (CITEC) at Bielefeld University, for financial support in the form of their InterAct scholarship.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00451/full#supplementary-material

REFERENCES

Abrams, R. A., and Christ, S. E. (2003). Motion onset captures attention. *Psychol. Sci.* 14, 427–432. doi: 10.1111/1467-9280.01458

Abrams, R. A., and Christ, S. E. (2005). The onset of receding motion captures attention: comment on Franconeri and Simons (2003). *Percept. Psychophys.* 67, 219–223. doi: 10.3758/BF03206486

Abrams, R. A., and Christ, S. E. (2006). Motion onset captures attention: a rejoinder to Franconeri and Simons (2005). *Percept. Psychophys.* 68, 114–117. doi: 10.3758/BF03193661

Barlow, H., and Levick, W. R. (1965). The mechanism of directionally selective units in rabbit's retina. *J. Physiol.* 178, 477–504. doi: 10.1113/jphysiol.1965.sp007638

Benosman, R., Clercq, C., Lagorce, X., Ieng, S.-H., and Bartolozzi, C. (2014). Event-based visual flow. *IEEE Trans. Neural Netw. Learn. Syst.* 25, 407–417. doi: 10.1109/TNNLS.2013.2273537

Benosman, R., Ieng, S.-H., Clercq, C., Bartolozzi, C., and Srinivasan, M. (2012). Asynchronous frameless event-based optical flow. *Neural Netw.* 27, 32–37. doi: 10.1016/j.neunet.2011.11.001

Bernardino, A., and Santos-Victor, J. (1999). Binocular tracking: integrating perception and control. *IEEE Trans. Robot. Autom.* 15, 1080–1094. doi: 10.1109/70.817671

Borst, A., Haag, J., and Reiff, D. F. (2010). Fly motion vision. *Annu. Rev. Neurosci.* 33, 49–70. doi: 10.1146/annurev-neuro-060909-153155

Borst, A., and Helmstaedter, M. (2015). Common circuit design in fly and mammalian motion vision. *Nat. Neurosci.* 18, 1067–1076. doi: 10.1038/nn.4050

Brosch, T., Tschechne, S., and Neumann, H. (2015). On event-based optical flow detection. *Front. Neurosci.* 9:137. doi: 10.3389/fnins.2015.00137

Cavanagh, P. (1992). Attention-based motion perception. *Science* 257, 1563–1565. doi: 10.1126/science.1523411

Devries, S. H., and Baylor, D. A. (1997). Mosaic arrangement of ganglion cell receptive fields in rabbit retina. *J. Neurophysiol.* 78, 2048–2060. doi: 10.1152/jn.1997.78.4.2048

Freeman, J., and Simoncelli, E. P. (2011). Metamers of the ventral stream. *Nat. Neurosci.* 14:1195. doi: 10.1038/nn.2889

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

Gallego, G., Gehrig, M., and Scaramuzza, D. (2019). "Focus is all you need: loss functions for event-based vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Long Beach, CA), 12280–12289. doi: 10.1109/CVPR.2019.01256

Gallego, G., Rebecq, H., and Scaramuzza, D. (2018). "A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 3867–3876. doi: 10.1109/CVPR.2018.00407

Gelbukh, A., Espinoza, F. C., and Galicia-Haro, S. N. (2014). "Human-Inspired Computing and its Applications," in *13th Mexican International Conference on Artificial Intelligence, MICAI2014* (Tuxtla Gutiérrez: Springer). doi: 10.1007/978-3-319-13647-9

Giulioni, M., Lagorce, X., Galluppi, F., and Benosman, R. B. (2016). Event-based computation of motion flow on a neuromorphic analog neural platform. *Front. Neurosci.* 10:35. doi: 10.3389/fnins.2016.00035

Haessig, G., Cassidy, A., Alvarez, R., Benosman, R., and Orchard, G. (2018). Spiking optical flow for event-based sensors using IBM's truenorth neurosynaptic system. *IEEE Trans. Biomed. Circ. Syst.* 12, 860–870. doi: 10.1109/TBCAS.2018.2834558

Harvey, B. M., and Dumoulin, S. O. (2011). The relationship between cortical magnification factor and population receptive field size in human visual cortex: constancies in cortical architecture. *J. Neurosci.* 31, 13604–13612. doi: 10.1523/JNEUROSCI.2572-11.2011

Hassenstein, B., and Reichardt, W. (1956). Systemtheoretische analyse der zeit-, reihenfolgen-und vorzeichenauswertung bei der bewegungsperzeption des rüsselkäfers chlorophanus. *Z. Naturforschung B* 11, 513–524. doi: 10.1515/znb-1956-9-1004

Horiuchi, T., Lazzaro, J., Moore, A., and Koch, C. (1991). "A delay-line based motion detection chip," in *Advances in Neural Information Processing Systems 3 (NIPS 1990)* (San Mateo, CA: Morgan Kauffmann), 406–412.

Howard, C. J., and Holcombe, A. O. (2010). Unexpected changes in direction of motion attract attention. *Attent. Percept. Psychophys.* 72, 2087–2095. doi: 10.3758/BF03196685

Jones, G., and Holderied, M. W. (2007). Bat echolocation calls: adaptation and convergent evolution. *Proc. R. Soc. B Biol. Sci.* 274, 905–912. doi: 10.1098/rspb.2006.0200

Kramer, J. (1996). Compact integrated motion sensor with three-pixel interaction. *IEEE Trans. Pattern Anal. Mach. Intell.* 18, 455–460. doi: 10.1109/34.491628

Lichtsteiner, P., Posch, C., and Delbruck, T. (2008). A 128x128 120 db 15us latency asynchronous temporal contrast vision sensor. *IEEE J. Solid State Circ.* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Lucas, B. D., and Kanade, T. (1981). "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: Proceedings of the 7th international joint conference on Artificial intelligence,* Vol. 2 (San Francisco, CA: Morgan Kaufmann Publishers Inc.), 674–679.

Maisak, M. S., Haag, J., Ammer, G., Serbe, E., Meier, M., Leonhardt, A., et al. (2013). A directional tuning map of drosophila elementary motion detectors. *Nature* 500, 212–216. doi: 10.1038/nature12320

Maunsell, J. H., and Cook, E. P. (2002). The role of attention in visual processing. *Philos. Trans. R. Soc. Lond. Ser B Biol. Sci.* 357, 1063–1072. doi: 10.1098/rstb.2002.1107

Mauss, A. S., Meier, M., Serbe, E., and Borst, A. (2014). Optogenetic and pharmacologic dissection of feedforward inhibition in drosophila motion vision. *J. Neurosci.* 34, 2254–2263. doi: 10.1523/JNEUROSCI.3938-13.2014

Milde, M. B., Bertrand, O. J., Benosmanz, R., Egelhaaf, M., and Chicca, E. (2015). "Bioinspired event-driven collision avoidance algorithm based on optic flow," in *2015 International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)* (Krakow: IEEE), 1–7. doi: 10.1109/EBCCSP.2015.7300673

Milde, M. B., Bertrand, O. J., Ramachandran, H., Egelhaaf, M., and Chicca, E. (2018). Spiking elementary motion detector in neuromorphic systems. *Neural Comput.* 30, 2384–2417. doi: 10.1162/neco_a_01112

Mitrokhin, A., Fermüller, C., Parameshwara, C., and Aloimonos, Y. (2018). "Event-based moving object detection and tracking," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Madrid: IEEE), 1–9. doi: 10.1109/IROS.2018.8593805

Murray, I., MacCana, F., and Kulikowski, J. (1983). Contribution of two movement detecting mechanisms to central and peripheral vision. *Vis. Res.* 23, 151–159. doi: 10.1016/0042-6989(83)90138-4

Nelson, M. E., and MacIver, M. A. (2006). Sensory acquisition in active sensing systems. *J. Comp. Physiol. A* 192, 573–586. doi: 10.1007/s00359-006-0099-4

Nelson, R. C., and Aloimonos, J. (1989). Obstacle avoidance using flow field divergence. *IEEE Trans. Pattern Anal. Mach. Intell.* 11, 1102–1106. doi: 10.1109/34.42840

Panerai, F. M., Capurro, C., and Sandini, G. (1995). "Space-variant vision for an active camera mount," in *Visual Information Processing IV*, Vol. 2488 (Orlando, FL: International Society for Optics and Photonics), 284–296. doi: 10.1117/12.211981

Posch, C., Matolin, D., and Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE J. Solid State Circ.* 46, 259–275. doi: 10.1109/JSSC.2010.2085952

Ramesh, B., Yang, H., Orchard, G. M., Le Thi, N. A., Zhang, S., and Xiang, C. (2019). Dart: distribution aware retinal transform for event-based cameras. *IEEE Trans. Pattern Anal. Mach. Intell.* doi: 10.1109/TPAMI.2019.2919301

Schoepe, T., Gutierrez-Galan, D., Dominguez-Morales, J., Jimenez-Fernandez, A., Linares-Barranco, A., and Chicca, E. (2019). "Neuromorphic sensory integration for combining sound source localization and collision avoidance," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Nara), 1–4. doi: 10.1109/BIOCAS.2019.8919202

Strother, J. A., Wu, S.-T., Wong, A. M., Nern, A., Rogers, E. M., Le, J. Q., et al. (2017). The emergence of directional selectivity in the visual motion pathway of drosophila. *Neuron* 94, 168–182. doi: 10.1016/j.neuron.2017.03.010

Traschütz, A., Zinke, W., and Wegener, D. (2012). Speed change detection in foveal and peripheral vision. *Vis. Res.* 72, 1–13. doi: 10.1016/j.visres.2012.08.019

Wässle, H., and Riemann, H. (1978). The mosaic of nerve cells in the mammalian retina. *Proc. R. Soc. Lond. Ser B Biol. Sci.* 200, 441–461. doi: 10.1098/rspb.1978.0026

Wurbs, J., Mingolla, E., and Yazdanbakhsh, A. (2013). Modeling a space-variant cortical representation for apparent motion. *J. Vis.* 13:2. doi: 10.1167/13.10.2

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 Neuroscience

ORIGINAL RESEARCH
published: 12 May 2020
doi: 10.3389/fnins.2020.00421

Biologically Relevant Dynamical Behaviors Realized in an Ultra-Compact Neuron Model

Pablo Stoliar[1†], Olivier Schneegans[2†] and Marcelo J. Rozenberg[3†]*

[1] National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan, [2] CentraleSupélec, CNRS, Université Paris-Saclay, Sorbonne Université, Laboratoire de Génie Electrique et Electronique de Paris, Gif-sur-Yvette, France, [3] Université Paris-Saclay, CNRS, Laboratoire de Physique des Solides, Orsay, France

We demonstrate a variety of biologically relevant dynamical behaviors building on a recently introduced ultra-compact neuron (UCN) model. We provide the detailed circuits which all share a common basic block that realizes the leaky-integrate-and-fire (LIF) spiking behavior. All circuits have a small number of active components and the basic block has only three, two transistors and a *silicon controlled rectifier* (SCR). We also demonstrate that numerical simulations can faithfully represent the variety of spiking behavior and can be used for further exploration of dynamical behaviors. Taking Izhikevich's set of biologically relevant behaviors as a reference, our work demonstrates that a circuit of a LIF neuron model can be used as a basis to implement a large variety of relevant spiking patterns. These behaviors may be useful to construct neural networks that can capture complex brain dynamics or may also be useful for artificial intelligence applications. Our UCN model can therefore be considered the electronic circuit counterpart of Izhikevich's (2003) mathematical neuron model, sharing its two seemingly contradicting features, extreme simplicity and rich dynamical behavior.

Keywords: spiking neural networks, neuron models, leaky-integrated-and-fire, artificial intelligence, neuromorphic electronic circuits, neuromorphic computers

OPEN ACCESS

Edited by:
Emre O. Neftci, University of California, Irvine, United States

Reviewed by:
Zhijun Yang, Middlesex University, United Kingdom
Khaled Nabil Salama, King Abdullah University of Science and Technology, Saudi Arabia

****Correspondence:***
Marcelo J. Rozenberg
marcelo.rozenberg@universite-paris-saclay.fr

† These authors have contributed equally to this work

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *15 November 2019*
Accepted: *07 April 2020*
Published: *12 May 2020*

Citation:
Stoliar P, Schneegans O and Rozenberg MJ (2020) Biologically Relevant Dynamical Behaviors Realized in an Ultra-Compact Neuron Model. Front. Neurosci. 14:421. doi: 10.3389/fnins.2020.00421

INTRODUCTION

In his 2003 landmark paper Izhikevich (2003) emphasized that to develop a large-scale model of the brain one faces seemingly mutually exclusive requirements: on one hand the model had to be simple enough to allow for efficient computation and, on the other, it had to be able to produce a rich variety of biologically relevant firing patterns. Interestingly, the same dilemma is encountered for the implementation of neurons in neuromorphic circuits – i.e., circuits that perform computations based on the architecture of the brain. Recently we proposed an ultra-compact neuron (UCN) circuit that realizes the leaky integrate and fire model (Rozenberg et al., 2019). That circuit complies with the first requirement, as it was simply based on only three active devices, two transistors and a thyristor, or SCR, plus a capacitor and a few resistors. Here we shall show that the UCN model also complies with the second requirement. Specifically, we shall demonstrate that the UCN is a circuit block that, with minimal variations, may realize at least 12 out of the 20 biological relevant behaviors highlighted by Izhikevich (2004). To reproduce those behaviors has become a *de facto* standard to demonstrate the relevance of a spiking neuron model implemented on different physical supports. The literature is very diverse and growing fast, so we shall only cite a few examples here and refer the

readers to further references in those works and in the review of Indiveri et al. (2011): the digital processor chips TrueNorth developed by IBM (Cassidy et al., 2013; Merolla et al., 2014) and the more recent ODIN by ICTEAM (Frenkel et al., 2019); the compact neuron circuit, with only 14 MOSFET transistors proposed by Wijekoon and Dudek (2008); or the radically different spiking neuron based on vanadium dioxide (Yi et al., 2018), a Mott insulator memristive material (del Valle et al., 2018, del Valle et al., 2019). Other interesting proposals, which aimed at a faithful physical implementation of the Izhikevich mathematical model equations are: a compact circuit of MOS transistors in the subthreshold regime, simulated with MOSIS libraries (Rangan et al., 2010); a CMOS digital neuron for event-driven computation, simulated in Spice (Imam et al., 2010).

Silicon Neuron (SiN) Circuits

The UCN belongs to the class of SiN circuits (Indiveri et al., 2011), which are electronic hardware implementations of systems that aim to emulate the electric behavior of biological neurons. These SiN blocks may then be integrated to construct larger circuits (Qiao et al., 2015), such as to emulate neural network for artificial intelligence applications, or brain-like systems for basic neuroscience research. The SiN circuits are inspired from a multiplicity of mathematical neuronal models that range from the simplest integrate and fire to the realistic Hodgkin-Huxley (Gerstner et al., 2014). Depending on the desired goal, SiN implementations may favor different features, such as low power dissipation, circuit simplicity, low variability, realistic behavior, tunability, etc. Typically, they are implemented using CMOS technology and VLSI (Indiveri et al., 2011), and they can be broadly classified as sub-threshold or above-threshold depending on the conduction mode of the transistors. The sub-threshold systems follow from the pioneer work of Mead (Mead, 1989, Mead, 1990) and of Mahowald and Douglas (1991) that emphasized the similarity between the exponential behavior of carrier conduction in transistor channels with that of ionic channels in neurons, and coined the concept of "neuromorphic behavior." The systems in the sub-threshold regime have the additional attractive features of low power dissipation, which follows from the small currents, and time constants that are compatible with the biological ones (Indiveri et al., 2011). However, a main drawback is the so called device mismatch, which is a relatively large variability between cells (Indiveri et al., 2011). As an example of this approach we may mention that of Yu and Cauwenberghs (2010) that implemented a SiN to realize the realistic Hodgkin-Huxley model. The above-thershold implementations avoid mismatch and thus have the precision needed to faithfully recreate the mathematical models that motivate them. These SiN circuits also operate a time-constants that are much faster (10^3–10^4) than the biological ones, unless they adopt off-chip larger capacitors. One example of above-threshold systems is the implementation of a tunable Hodgkin-Huxley model by Saighi et al. (2011).

A different approach is to develop SiN circuits that are motivated on generalizations of the simple integrate and fire model (Gerstner et al., 2014). Some examples are the AdEx (Brette and Gerstner, 2005), the Izhikevich (Izhikevich, 2003, 2004) and the Mihalas-Niebur neuron (Mihalas and Niebur, 2009). These models do not necessarily have a biological underpinning as the Hodgkin-Huxley, but nevertheless were shown to capture the relevant spiking patterns observed in biological neurons. Their main attractive is that their relative simplicity allow for more efficient implementations in both software and hardware (Wijekoon and Dudek, 2008; Folowosele et al., 2009; Livi and Indiveri, 2009; Indiveri et al., 2010; Rangan et al., 2010; van Schaik et al., 2010a,b; Qiao et al., 2015).

The Ultra-Compact Neuron

In this context the UCN that we introduced recently (Rozenberg et al., 2019) opens a different paradigm. Similar to sub-threshold systems and faithful to the concept of neuromorphic engineering, it exploits an intrinsic non-linearity of an electronic device. Namely the threshold switching of the SCR conductance emulates the firing of biological neurons. As we discussed in Rozenberg et al. (2019), this features permits a drastic reduction to the number of components to implement a basic leaky-integrate-and-fire (LIF) SiN. So in this regard it may be considered as belonging to the class of Compact SiN circuits (Indiveri et al., 2011). However an attractive feature of the UCN is that they can be directly interconnected, therefore need not *a priori* require an additional address-event representation off-chip system. Despite the fact that the thyristor was introduced in the very beginnings of semiconductor electronics, it is currently not a conventional CMOS device. Its development in microelectronics is mostly restricted to protection circuits (Ker and Hsu, 2005), which nevertheless demonstrates that there are no *a priori* impediments for its CMOS implementation. The time-constants associated to the switching of a SCR are short, thus in this regard the operation of the UCN follows similar features as the above-threshold SiN as we mentioned above. Therefore, if the goal is to achieve biological time-scales one may need large "membrane" capacitors, hence our UCN should not be considered compact in regard of the wafer real estate.

In the present work we shall describe how the functionality of the basic UCN block can be extended to realize a variety of biologically relevant spiking patterns, without a sacrifice of circuit simplicity. The paper is organized as follows: In section Materials and Methods we shall describe our recently introduced UCN circuit (Rozenberg et al., 2019). We shall demonstrate how the basic behavior of the UCN can be very precisely captured by means of numerical simulations obtained with LTspice (LTspice®, 2020 that we validate against actual circuit measured data. Section Results contain the main results of the present work. In the first part, we exploit the simplicity of the simulation package capabilities to explore extensions of the basic UCN circuit block, searching for different types of biologically relevant dynamical behaviors. We shall demonstrate that small variants of a basic circuit allow us to capture at least 12 out of the 20 dynamical behaviors, including some inhibition ones. In the second part of this section, we use the simulation results to achieve the main goal of this work, namely to provide the explicit circuits and measure them to demonstrate that the thyristor-based UCN can realize the complex firing patterns observed in biological systems. Our simulations inform and guide the implementation

of the actual electronic circuits that we construct with out-of-the-shelf components. This feature underscores the relative ease for the reproducibility of our work and may prompt other research groups to embark along the present line of work. The circuits details and the list of components are described in the **Supplementary Material**. In section Discussion we finally discuss some specific technical aspects of our work in regard of different open challenges in the field.

MATERIALS AND METHODS

The Ultra-Compact Neuron Model

In a recent paper (Rozenberg et al., 2019) we introduced the ultra-compact spiking neuron model that only requires two transistors and a thyristor (SCR). We follow the terminology of Indiveri et al. (2011), where a *compact* model refers to a simple electronic circuit with few components. As was argued in Rozenberg et al. (2019), the UCN has a *minimal* number of components, just one capacitor for the *integrate*, one resistor for the *leak*, and one thyristor for the *fire* functionalities. Therefore, one may consider the electronic circuit as an *ultra compact* realization of the LIF neuron. The thyristor is a standard electronic device, often used in high power applications, which consists of a tri-junction *pnpn* device that can be implemented in VLSI (Tong et al., 2012, 2014).

The behavior of a thyristor can be considered as similar to that of a diode where the access to the conduction state is controlled by an adjustable threshold, whose value is set by the voltage at the gate electrode. Once conducting, the SCR remains in a low resistance state till the current becomes smaller than a small holding value I_{hold}. Thus, the SCR polarized in direct has *two resistive states*: a high resistance that switches to a low resistance one when the threshold V_{th} is overcome, and the low resistance that switches back to high when the current is below I_{hold}. Therefore, one may consider the SCR as a *memristor* since it is a resistive device with a hysteresis or memory effect. In fact, the I-V characteristics of the SCR are qualitatively similar to a type of memristive device with *volatile resistive switching*, which are based on transition metal oxide materials that display Mott insulator-metal transitions (del Valle et al., 2018; Rozenberg et al., 2019). Work along these lines was recently reported in Yi et al. (2018), where a device made of two VO_2 memristors was conceived to realize a Hodgkin-Huxley type neuron model. Using variants of that basic device, the authors demonstrated a large variety of biologically relevant spiking patterns.

In **Figure 1** we show the schematic circuit of the UCN that can be easily implemented using out-of-the-shelf components. Details of the circuit elements and their values are provided in the **Supplementary Material**. The memristive effect of the SCR can be straightforwardly exploited in the "soma" block of the UCN circuit (see **Figure 1**) to achieve the basic LIF behavior. The SCR is initially in a high-resistance state during the *integrate* phase, providing a small *leakage* to the capacitor charge. Then, upon reaching the voltage threshold, the SCR switches to the low-resistance state and *fires* a pulse of current. The current is due to the charge accumulated in the capacitor that rapidly discharges through the SCR. The small holding current ensures the almost full discharge of the capacitor.

The UCN circuit has also a second bock, the "axon" block with two transistors (see **Figure 1**), which strengthens the current pulse. This feature is key to enable that the output spikes of one (upstream) neuron may excite a second (downstream) neuron. This enables the UCN to be interconnected as modular blocks of a *neural network circuit* as it was shown in a previous publication (Rozenberg et al., 2019).

Validation of Numerical Simulations of the UCN Circuit

We may exploit the fact that our UCN is simply implemented with standard electronic components to reproduce the spiking behavior using the standard electronic circuit simulation package LTspice, which is freely available (freeware) (LTspice®, 2020). In order to represent the actual SCR that we adopted in our circuits, we found convenient to modify the default parameters of the EC103D1 thyristor (SCR) model (LITTELFUSE, 2019). Specifically, we changed the value of the parameter BF from 6.10 to 2.90, and all the other parameters were left unchanged (further details are provided in the **Supplementary Material**).

The main results of the simulations of the basic UCN block are shown in **Figure 1**. The validation of the numerical data is done by comparison to the data measured in the actual circuit. As can be observed in the figure, the agreement is excellent. We may note a minor difference in a transient effect when the input reached the threshold voltage to excite spikes. Besides the small difference, the remarkable agreement validates the LTspice package simulations as an efficient method to explore variants of the basic UCN circuit and search for biologically relevant dynamical behaviors. The results of the numerical exploration will then serve us to inform the actual circuit implementations, which is the ultimate goal of our work.

RESULTS

In this section we present our results. We firstly describe the biologically relevant behaviors obtained through the numerical simulations study of circuits that are variations from the UCN basic block. Then, secondly, we describe the results of the measurements made on actual electronic circuits, whose implementation was informed by the numerical simulations study.

Numerical Simulation Study

In **Figure 2** we present the results of the exploration of circuits variants. We were able to obtain 12 out of the 20 biologically relevant behaviors captured initially by the Izhikevich neuron model (Izhikevich, 2003, 2004). However, we do not exclude the possibility of capturing the totality of those behaviors, a task that we are leaving for future work.

The behaviors that we obtained include some of the most significant ones. For instance, we find *class 1 and class 2 excitability*, panels (f) and (g), which were originally identified by Hodgkin as the two prototypical ways an individual neuron

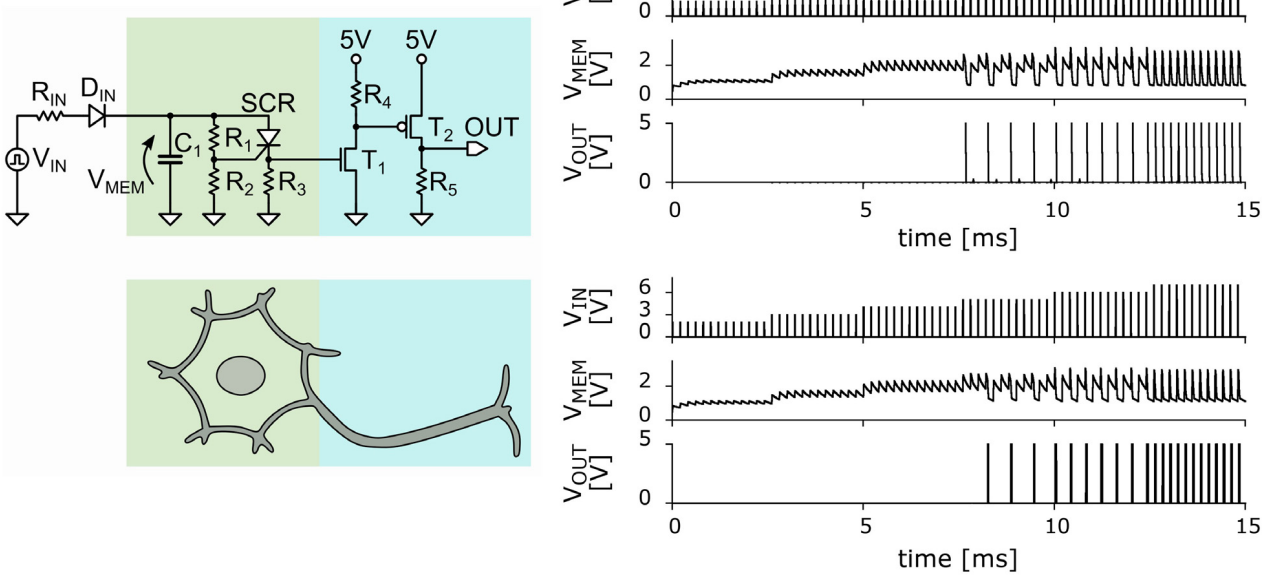


FIGURE 1 | Left panels: Basic circuit of the ultra-compact LIF neuron. The portion of the circuit representing the cell body or soma is in green, and that representing the axon is in blue (Rozenberg et al., 2019). **Right panels:** Comparison of the measured (top) and simulated (bottom) basic LIF spiking behavior of the neuron circuit. V_{IN} (measured in volts) denotes the incoming applied voltage pulses from the voltage pulse generator as function of time (in milli-sec). V_{MEM} denotes the voltage at the "membrane" capacitor as function of time. V_{OUT} denotes the output voltage showing a spiking pattern. The existence of a threshold value of V_{IN} for the spiking behavior is clearly observed. It is also observed that as incoming voltage is increased the output spiking rate increases. The agreement between measured data and the simulations is excellent, with the exception of a small transient near the threshold input voltage value.

can start spiking when excited by an external current source (Hodgkin, 1948). In addition, we also obtained the most basic behaviors, such as *phasic spiking and phasic bursting,* **Figures 2B,C**, which are considered to be associated to a neuron signaling or flagging the beginning of an activity or the presence of a stimulus. Another biologically relevant behavior that we captured is *spike frequency adaptation,* **Figure 2E**, which is key to habituation. Importantly this behavior is also a key feature of neuronal circuits that can reproduce some basic global brain behaviors, such as asynchronous irregular and regular oscillatory spiking states (Destexhe, 2009; di Volo et al., 2019).

We also obtained *accommodation,* **Figure 2J**, where the neuron reacts to the rate of change of the input potential. This behavior is associated to a threshold in the time domain, as the neuron gets excited by sudden changes in the environment. It can also be considered as the neuromorphic equivalent of a high-pass filter. We should note, however, that our second input pulse is of the same strength as the first, as small variation with respect to the respective Izhikevich pattern (Izhikevich, 2004). *Delayed phasic spiking* or spike latency was also captured, **Figure 2H**, which is a behavior that may allow the system to adjust the timing of its reaction to a given input.

The two last behaviors displayed in **Figures 2K,L**, show that the UCN can also be adapted to accept negative polarity input and produce both, the *rebound spike* and *inhibition-induced tonic spiking*. The negative polarity excitability is associated to inhibition and hyperpolarization of the cell body. For further discussion on the biological relevance of the behaviors we refer the reader to the work of Izhikevich (2004) and the references therein.

All these behaviors were obtained with variants of the basic LIF circuit that involved only one neuron. However, some additional complex behaviors may require the combination of two or more neurons. This interesting possibility lies beyond the scope of the present work.

Simulated Circuits

In **Figure 3** we provide some examples of the simulated circuits. As one can see, they are in fact small variants of the basic LIF model block, which is indicated by a red dotted box in the panels. That portion of the circuit is the UCN block that we introduced and discussed in the previous section and in **Figure 1**.

Figure 3A, we observe that the small addition made to the basic UCN block is to merely supplement it with an (RC) passive differentiator circuit at the input. This enables to obtain either phasic spiking or phasic bursting behaviors, as the input capacitor gets charged with the first (or few first) incoming pulses and then prevents further DC excitation of the UCN block. On the other hand, in the circuit of **Figure 3B** we observe that the UCN is supplemented with a feed-back loop. The key feature here is that the feed-back is connected to the gate of the SCR, regulating the effective resistance that grounds the gate. This, in turn, lowers the value of the anode-cathode threshold voltage for the resistive switch of the SCR, and hence one may obtain the spike-frequency adaptation behavior.

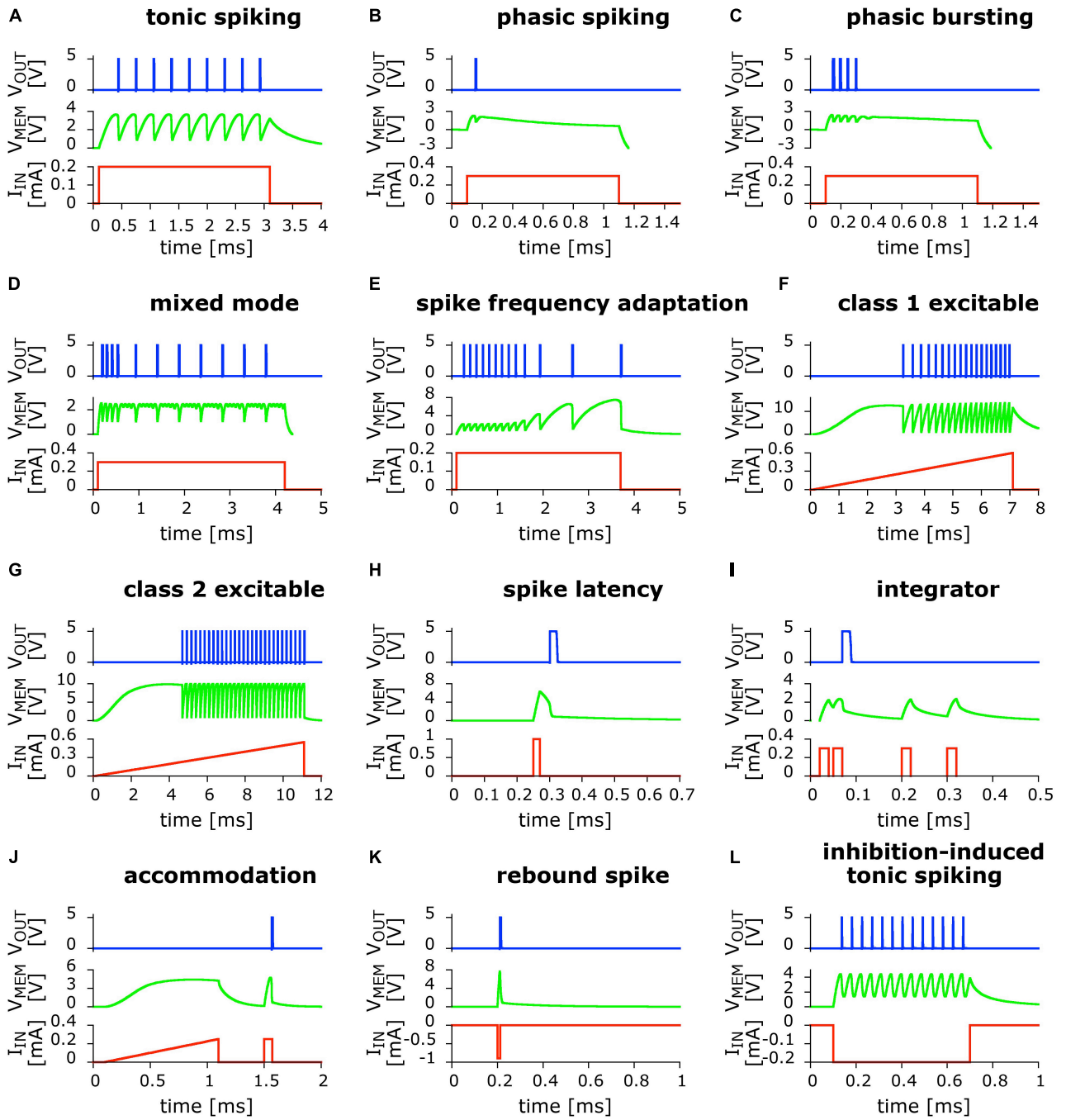


FIGURE 2 | Twelve biologically relevant behaviors obtained from numerical simulations of small variants of the basic UCN LIF circuit, using the LTspice code. In red line is the applied input current excitation I_{IN} as function of time. In green line is the "membrane" voltage V_{MEM} at capacitor C_1. In blue line is the output voltage V_{OUT} that shows the dynamical spiking behavior. From left to right and top to bottom: **(A)** Tonic spiking; **(B)** phasic spiking; **(C)** phasic bursting; **(D)** mixed (burst then spiking); **(E)** spike frequency adaptation; **(F)** class 1 excitable; **(G)** class 2 excitable; **(H)** spike latency; **(I)** integrator; **(J)** accommodation; **(K)** rebound spike (negative polarity input); and **(L)** inhibition-induced (negative polarity input) tonic spiking.

As shown in **Figure 3C**, the implementation of the inhibition induced behaviors can be achieved by simply exchanging the positions of the SCR with R_3, and of T_1 with R_4. In this case, the neuron is excited with negative polarity input pulses. In principle, we no longer need to use the second transistor T_2 of the axon block, which was only meant to invert the polarity of the outgoing pulse. Therefore, the UCN for inhibition-induced input excitation requires even less components than the original UCN block (red box). However, T_2 would be required if one desires to generate negative polarity pulses on output.

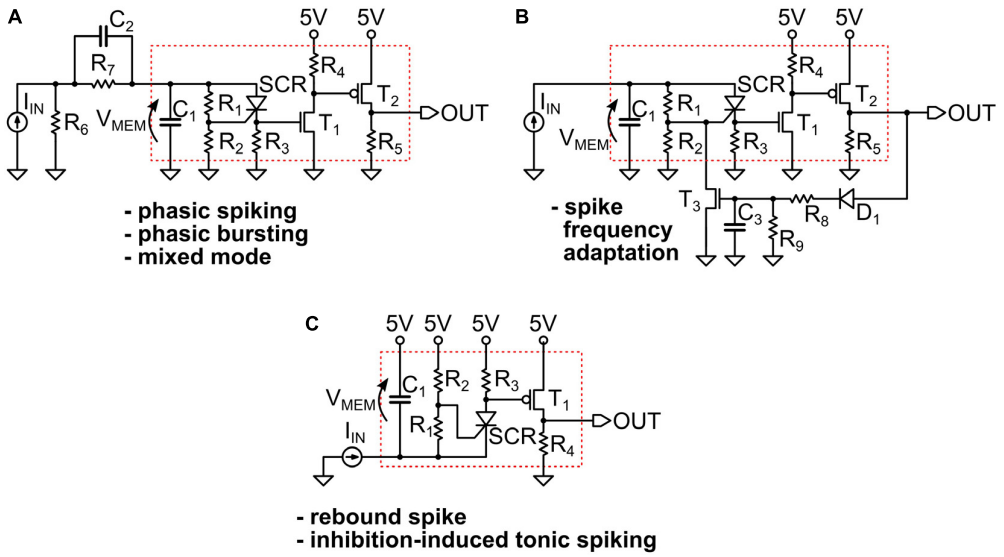


FIGURE 3 | (A,B) The circuits that implement the simulated spiking behaviors shown in the **Figure 2** of the main text. They all represent small variants of a basic UCN circuit block that is indicated by a red dotted line box. We indicate with an arrow the definition of the "membrane" voltage V_{MEM} on the capacitor C_1. The circuit adapted for negative input is shown in **(C)**, where we indicate with a red box the basic UCN block that is slightly modified with respect to the original. Note that the rebound spike may also be called inhibition-induced phasic spike. The specific values of the components are provided in **Supplementary Material**.

These simulated circuits will serve as a basis to inform the implementation of the actual electronic circuits that we describe in the next subsection.

Biologically Relevant Behaviors Realized in Actual UCN Electronic Circuits

Our stated strategy was to use simulations to rapidly explore circuit variants, but the ultimate goal is of course to implement the actual electronic circuits. The soundness of this approach is demonstrated by the diversity of the behaviors that we were able to realize, as shown in **Figure 4**.

Eleven behaviors, shown in **Figures 4A–J,L**, are close implementations of the previously simulated circuits. They include both, positive and negative input excitation. We note that a twelfth behavior, *rebound spike*, can be trivially obtained from behavior (l) by simply reducing the duration of the input pulse, therefore is not shown. Interestingly, during our experimental circuits study, we also found an additional biologically plausible behavior, shown in **Figure 4K** that corresponds to *delayed bursting* (Zeldenrust et al., 2018). Other interesting spiking patterns were also observed, including negative input, which we leave to future work.

Variations of the UCN Electronic Circuit Block

In **Figure 5** we show some significant examples of the actually implemented circuits (the specific values of the components are provided in the **Supplementary Material**). Similarly, as was the case of numerical simulations, all the circuits are rather small variants of the basic UCN block that we indicate with a red dotted box in the figure.

The excellent agreement between the spiking patterns of the simulations and the actual circuits is quite apparent upon comparison of the many panels of **Figures 2**, **4**. However, we should also mention that while the simulated circuits always provided a good starting point, it was necessary to make further modifications in the actual circuit implementation in order to achieve the desired spiking pattern. One of the most significant differences that illustrates the point is in **Figure 5D**, which shows the circuit for phasic spiking, mixed mode and accommodation, and can be compared to that of **Figure 3A**. Actually, both circuits provide valid solutions, however, upon implementation of the simulated 3-a, we found that it imposed too high a demand of current from the input signal generator. Hence, we search for a circuit variant. This was obtained by exploiting the additional freedom of tuning the gate of the SCR. This small example illustrates the benefits and shortcoming of simulations, with respect to the ultimate goal, which is the circuit implementation.

We also note that the implementation of negative input pulses (inhibition), shown in **Figure 5E**, also required some small modifications. The rest of the circuits are almost identical to the simulated ones. Besides the small circuit changes, it was also sometimes necessary to adapt the input strengths, such as comparing (**Figure 2F**) and (**Figure 4F**); or different spiking frequency were obtained, such as in **Figures 2G**, **4G**. In any case, we want to emphasize that exact agreement was not the goal, except in the initial validation of the numerical model described in Section Materials and Methods. For the whole variety of obtained behaviors the differences remained merely quantitative, and within an order of magnitude, and the qualitative agreement was always satisfactory.

Some additional details on the choice of electronic components and the implementation of the measurement system are given in the **Supplementary Material**.

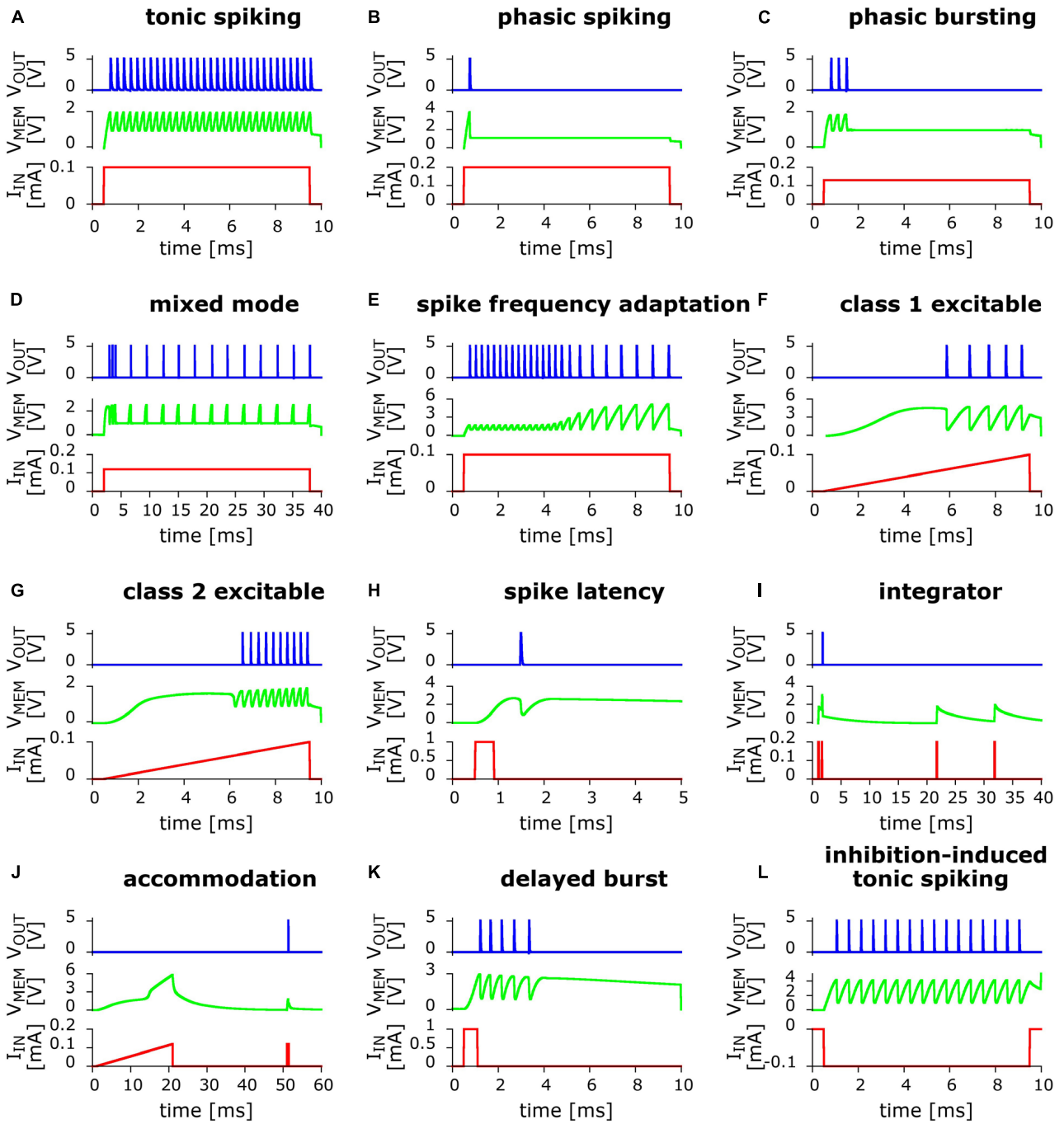


FIGURE 4 | Twelve biologically relevant behaviors measured in small variants of the basic UCN circuit. In red line is the applied input current excitation I_{IN} as function of time. In green line is the "membrane" capacitor voltage V_{MEM}. In blue line is the output voltage V_{OUT} that shows the dynamical spiking behavior. From left to right and top to bottom: **(A)** Tonic spiking; **(B)** phasic spiking; **(C)** phasic bursting; **(D)** mixed (burst then spiking); **(E)** spike frequency adaptation; **(F)** class 1 excitable; **(G)** class 2 excitable; **(H)** spike latency; **(I)** integrator; **(J)** accommodation; **(K)** delayed burst (note: this behavior is biologically plausible but not among the Izhikevich set); and **(L)** inhibition-induced (negative polarity input) tonic spiking (note: the "rebound spike" behavior can be trivially obtained from this one via a shorter time input). Further details on the measured circuits along with the values of their components are described in the **Supplementary Material**.

DISCUSSION

In this work we have illustrated the versatility of the UCN circuit to capture a significant number of biologically relevant neuronal behaviors. We have not attempted to demonstrate the totality of the 20 behaviors identified by Izhikevich (2004). Rather, our goal was to demonstrate that the minimal UCN circuit is a sound basis to implement a new type of spiking neuron model

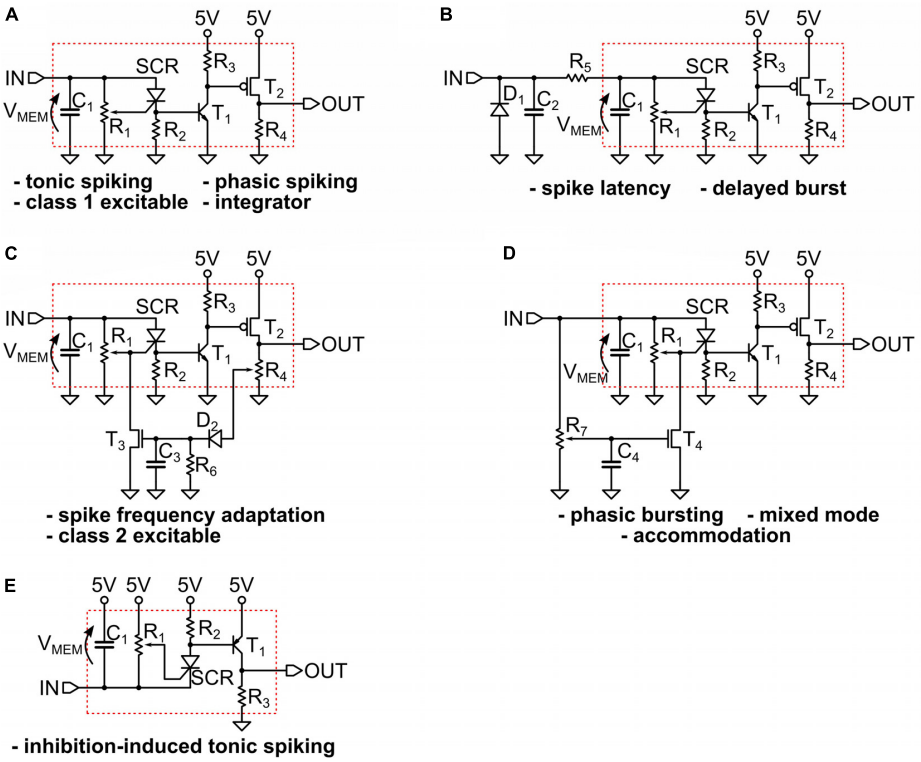


FIGURE 5 | (A–D) The circuits that implement the measured spiking behaviors shown in the **Figure 3**. We indicate with an arrow the definition of the "membrane" voltage V_{MEM} on the capacitor C_1. The circuit adapted for negative input is shown in **(E)**, where we indicate with a red box the basic UCN block that is slightly modified with respect to the original. The specific values of the components are provided in **Supplementary Material**.

of remarkable simplicity. Another important result of our study was the successful implementation of the numerical simulation package to efficiently search for the complex spiking patterns. This required the implementation of the SCR model. More generally, reliable numerical simulations methods become an essential tool to implement small neural sub-circuits counting tens or hundreds of spiking neurons.

The key feature that enables this circuit simplicity is the memristive behavior of the SCR, which is a conventional electronic component that may be implemented in CMOS technology (Ker and Hsu, 2005; Tong et al., 2012). However, we should also note that although the UCN model and the extensions proposed in the present work only require a reduced number of electronic components, they are of different types, which may pose a challenge for the integration into a single technology. Nevertheless, this may be achieved by Bi-CMOS (Alvarez, 1990), or the more recent BCD8sP technology (Roggero et al., 2013). Simulation of our UCN circuits to implement actual chips is an exciting prospect that is beyond the scope of the present work.

Also in regard of the prospects for microelectronic implementation, one should be aware that, similarly to all compact neuron model circuits based on standard electronics, the UCN also requires a "membrane" capacitor to integrate charge. This feature remains a significant problem for miniaturization as the capacitors still require a relatively large physical space in VLSI.

The ultimate solution for a low-power and low-footprint spiking neuron device may therefore require memristors based on Mott materials (del Valle et al., 2018). However, achieving a reliable control and a theoretical understanding of the metal-insulator transitions in those compounds still represent a significant challenge (Yi et al., 2018; del Valle et al., 2019).

To conclude, the UCN model is a simple modular block that can be used to implement spiking neuron circuits. The present work demonstrates that its simplicity does not prevent the realization of complex spiking patterns, beyond the integrate and fire paradigm.

Our work opens a new way for the implementation of large neuronal networks with biological plausibility and of unprecedented simplicity.

DATA AVAILABILITY STATEMENT

All the information required to obtain the datasets generated for this study is included in the article/**Supplementary Material**.

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct and intellectual contribution to the work, and approved it for publication.

FUNDING

MR acknowledges support from ANR Grant MoMA. This work was supported in part by the JSPS KAKENHI Grant No. JP18H05911.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00421/full#supplementary-material

REFERENCES

Alvarez, A. R. (1990). *Introduction to BiCMOS. BiCMOS Technology and Applications*. Berlin: Springer Science & Business Media. 1–20.

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

Cassidy, A. S., Merolla, P., Arthur, J. V., Esser, S. K., Jackson, B., Alvarez-Icaza, R., et al. (2013). "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Proceedings of the 2013 International Joint Conference on Neural Networks (IJCNN)*, (Piscataway, NJ: IEEE), 1–10.

del Valle, J., Ramírez, J. G., Rozenberg, M. J., and Schuller, I. K. (2018). Challenges in materials and devices for resistive-switching-based neuromorphic computing. *J. Appl. Phys.* 124:211101

del Valle, J., Salev, P., Tesler, F., Vargas, N. M., Kalcheim, Y., Wang, P., et al., (2019). Subthreshold firing in Mott nanodevices. *Nature* 569, 388–392. doi: 10.1038/s41586-019-1159-6

Destexhe, A. (2009). Self-sustained asynchronous irregular states and Up-Down states in thalamic, cortical and thalamocortical networks of nonlinear integrate-and-fire neurons. *J. Comput. Neurosci.* 27, 493–506. doi: 10.1007/s10827-009-0164-4

di Volo, M., Romagnoni, A., Capone, C., and Destexhe, A. (2019). Biologically realistic mean-field models of conductance-based networks of spiking neurons with adaptation. *Neural Comput.* 31, 653–680. doi: 10.1162/neco_a_01173

Folowosele, F., Etienne-Cummings, R., Hamilton, T. J. (2009). "A CMOS switched capacitor implementation of the mihalas-niebur neuron," in *Proceedings of the Biomedical Circuits and Systems Conference, BIOCAS 2009* (Beijing: IEEE), 105–108.

Frenkel, C., Legat, J. D., and Bol, D. (2019). MorphIC: a 65-nm 738k-Synapse/mm2 Quad-Core binary-weight digital neuromorphic processor with stochastic spike-driven online learning. *IEEE Trans. Biomed. Circuits Syst.* 13, 999–1010. doi: 10.1109/TBCAS.2019.2928793

Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Reading: Cambridge University Press.

Hodgkin, A. L. (1948). The local electric changes associated with repetitive action in a non-medullated axon. *J. Physiol.* 107, 165–181. doi: 10.1113/jphysiol.1948.sp004260

Imam, N., Wecker, K., Tse, J., Karmazin, R., Manohar, R. (2010). "Neural spiking dynamics in asynchronous digital circuits" in *Proceedings of the 2013 International Joint Conference on Neural Networks (IJCNN)*, Dallas, TX, 1–8.

Indiveri, G., Stefanini, F. and Chicca, E. (2010). Spike-based learning with a generalized integrate and fire silicon neuron. in *Proceedings of the International Symposium on Circuits and Systems, ISCAS 2010* (Paris: IEEE), 1951–1954.

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al., (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Trans. Neural Netw.* 14, 1569–1572. doi: 10.1109/TNN.2003.820440

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw.* 15, 1063–1070. doi: 10.1109/TNN.2004.832719

Ker, M. -D., and Hsu, K. -C. (2005). Overview of on-chip electrostatic discharge protection design with SCR-based devices in CMOS integrated circuits. *IEEE Trans. Device Mat. Res.* 5, 235–249.

LITTELFUSE (2019). *Link to the EC103D1 SCR SPICE Model Specifications*. Available online at: https://www.littelfuse.com/technical-resources_old/spice-models/thyristor-spice-models.aspx

Livi, P., Indiveri, G. (2009). "A current-mode conductance-based silicon neuron for address-event neuromorphic systems," in *Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS 2009* (Taipei: IEEE), 2898–2901.

LTspice® (2020). *Analog Devices Inc., USA. (Freeware)* Available online at: https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html

Mahowald, M., and Douglas, R. (1991). A silicon neuron. *Nature* 354, 515–518.

Mead, C. A. (1989). *Analog VLSI and Neural Systems*. Reading: Addison-Wesley Longman Publishing Co.

Mead, C. A. (1990). Neuromorphic electronic systems. *Proc. IEEE* 78, 1629–1636.

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

Mihalas, S., 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

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al., (2015). A reconfigurable online learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Front Neurosci.* 9:141. doi: 10.3389/fnins.2015.00141

Rangan, V., Ghosh, A., Aparin, V., and Cauwenberghs, G. (2010). "A subthreshold aVLSI implementation of the Izhikevich simple neuron model," in *Proceedings of the 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, (Piscataway, NJ: IEEE), 4164–4167. doi: 10.1109/IEMBS.2010.5627392

Roggero, R., Croce, G., Gattari, P., Castellana, E., Molfese, A., Marchesi, G., et al., (2013). "BCD8sP: an advanced 0.16 m technology platform with state of the art power devices," *Proceedings of the 25th International Symposium on Power Semiconductor Devices & IC's (ISPSD)*, Kanazawa, 361–364.

Rozenberg, M. J., Schneegans, O., and Stoliar, P. (2019). An ultra-compact leaky-integrate-and-fire model for building spiking neural networks. *Sci. Rep.* 9:11123. doi: 10.1038/s41598-019-47348-5

Saighi, S., Bornat, Y., Tomas, J., Le Masson, G., Renaud, S. (2011). A library of analog operators based on the Hodgkin-Huxley formalism for the design of tunable, real-time, silicon neurons. *IEEE Trans. Biomed. Circuits Syst.* 5, 3–19. doi: 10.1109/TBCAS.2010.2078816

Tong, X., Wu, H., Liang, Q., Zhong, H., Zhu, H., Chen, D., et al., (2012). "On the design of 2-port SRAM memory cells using PNPN diodes for VLSI application," in *Proceedings of the SISPAD*, Denver, CO, 316–319

Tong, X., Wu, H., Liang, Q., Zhong, H., Zhu, H., Zhao, C., et al., (2014). Design of two-terminal PNPN diode for high-density and high-speed memory applications. *J. Semicond.* 35, 14006–14003.

van Schaik, A., Jin, C. and Hamilton, T. J. (2010a). "A log-domain implementation of the Izhikevich neuron model," in *Proceedings of the International Symposium on Circuits and Systems, ISCAS 2010*, (Paris: IEEE), 4253–4256.

van Schaik, A., Jin, C., Hamilton, T. J., Mihalas, S. and Niebur, E. (2010b). *International Symposium on Circuits and Systems, ISCAS 2010*, (Paris: IEEE), 4249–4252.

Wijekoon, J. H., and Dudek, P. (2008). Compact silicon neuron circuit with spiking and bursting behaviour. *Neural Netw.* 21, 524–534. doi: 10.1016/j.neunet.2007.12.037

Yi, W., Tsang, K. K., Lam, S. K., Bai, X., Crowell, J. A., and Flores, E. A. (2018). Biological plausibility and stochasticity in scalable VO2 active memristor neurons. *Nat. Commun.* 9:4661. doi: 10.1038/s41467-018-07052-w

Yu, T., and Cauwenberghs, G. (2010). Analog VLSI biophysical neurons and synapses with programmable membrane channel kinetics. *IEEE Trans. Biomed. Circuits Syst.* 4, 139–148. doi: 10.1109/TBCAS.2010.2048566

Zeldenrust, F., Wadman. W. J., and Englitz, B. (2018). Neural coding with bursts-current state and future perspectives. *Front. Comput. Neurosci.* 12:48. doi: 10.3389/fncom.2018.00048

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 Neuroscience

ORIGINAL RESEARCH
published: 19 May 2020
doi: 10.3389/fnins.2020.00420

Event-Based Computation for Touch Localization Based on Precise Spike Timing

Germain Haessig[1], Moritz B. Milde[2], Pau Vilimelis Aceituno[3,4], Omar Oubari[5], James C. Knight[6], André van Schaik[2], Ryad B. Benosman[5,7,8] and Giacomo Indiveri[1]*

[1] Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich, Switzerland, [2] International Centre for Neuromorphic Systems, MARCS Institute, Western Sydney University, Penrith, NSW, Australia, [3] Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany, [4] Max Planck School of Cognition, Leipzig, Germany, [5] Institut de la Vision, Sorbonne Université, Paris, France, [6] Centre for Computational Neuroscience and Robotics, School of Engineering and Informatics, University of Sussex, Brighton, United Kingdom, [7] University of Pittsburgh, Pittsburgh, PA, United States, [8] Carnegie Mellon University, Pittsburgh, PA, United States

Precise spike timing and temporal coding are used extensively within the nervous system of insects and in the sensory periphery of higher order animals. However, conventional Artificial Neural Networks (ANNs) and machine learning algorithms cannot take advantage of this coding strategy, due to their rate-based representation of signals. Even in the case of artificial Spiking Neural Networks (SNNs), identifying applications where temporal coding outperforms the rate coding strategies of ANNs is still an open challenge. Neuromorphic sensory-processing systems provide an ideal context for exploring the potential advantages of temporal coding, as they are able to efficiently extract the information required to cluster or classify spatio-temporal activity patterns from relative spike timing. Here we propose a neuromorphic model inspired by the sand scorpion to explore the benefits of temporal coding, and validate it in an event-based sensory-processing task. The task consists in localizing a target using only the relative spike timing of eight spatially-separated vibration sensors. We propose two different approaches in which the SNNs learns to cluster spatio-temporal patterns in an unsupervised manner and we demonstrate how the task can be solved both analytically and through numerical simulation of multiple SNN models. We argue that the models presented are optimal for spatio-temporal pattern classification using precise spike timing in a task that could be used as a standard benchmark for evaluating event-based sensory processing models based on temporal coding.

Keywords: temporal coding, event-based sensors, spatio-temporal patterns, spike-based computing, touch localization

OPEN ACCESS

Edited by:
Emre O. Neftci, University of California, Irvine, United States

Reviewed by:
Timothée Masquelier, Centre National de la Recherche Scientifique (CNRS), France
Arindam Basu, Nanyang Technological University, Singapore

****Correspondence:***
Germain Haessig
germain@ini.uzh.ch

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *15 December 2019*
Accepted: *07 April 2020*
Published: *19 May 2020*

Citation:
Haessig G, Milde MB, Aceituno PV, Oubari O, Knight JC, van Schaik A, Benosman RB and Indiveri G (2020) Event-Based Computation for Touch Localization Based on Precise Spike Timing. Front. Neurosci. 14:420. doi: 10.3389/fnins.2020.00420

1. INTRODUCTION

Information transmission in neural networks is often described in terms of the rate at which neurons emit action potentials. Neurons are typically assumed to encode values—such as the orientation of a bar—using their mean firing rate, with individual spikes emitted using a Poisson process (Dean, 1981). Neurons in higher processing areas of the brain (e.g., in primary visual cortex) have been shown to demonstrate variable spike timing in response to repetitions of identical stimuli (Hubel and Wiesel, 1962). This variability is commonly interpreted as being the result of

noise (or noisy background activity) which can be assumed to be an additive signal to the sensory input one (Baudot et al., 2013). This linear separation of signal and noise has been used to justify rate- and/or population-coding by averaging across time and/or neuronal populations (Shadlen and Newsome, 1998; Dayan and Abbott, 2001). These observations led to the common assumption that the main mode of information transmission in most brain areas is encoded in the neurons average spike-frequency. This assumption, supported by many experimental investigations (Softky and Koch, 1993; Dayan and Abbott, 2001), continues to be used in the field of machine learning.

However, the time at which spikes are emitted might also carry additional information. If this is the case, the temporal-correlation of such events can then be used an extra source of information for models of computation (Dayan and Abbott, 2001; Thorpe et al., 2001). This type of signal representation is described as a *temporal code*. In the last three decades, the advantages of temporal coding have been demonstrated in computational models of fast visual processing (Thorpe et al., 2001); for the classification of time-varying signals and balance (Gütig and Sompolinsky, 2006; Deneve and Machens, 2016); in temporal interval discrimination (Buonomano and Merzenich, 1995); in state-dependent computation (Buonomano and Maass, 2009), and even for fine motor control (Laje and Buonomano, 2013). Furthermore, experimental findings have shown that the information carried in the timing of spikes can be used by the brain to discriminate textures (Hipp et al., 2006; Wolfe et al., 2008), classify temporal patterns (Mainen and Sejnowski, 1995; Wehr and Zador, 2003; Baudot et al., 2013; Goel and Buonomano, 2016) or localize an animal in its environment (O'Keefe and Recce, 1993). This evidence demonstrates that neural networks—whether biological or artificial—can use spike timing information to extract relevant cues for behavior and generate events with precise timing precision in response to time-varying input patterns (Mainen and Sejnowski, 1995).

While both rate- and temporal-codes are used to convey information in the brain, conventional ANNs, for the most part, are based only on rate-codes. The contexts and tasks in which temporal-coding can outperform rate-coding remain elusive, especially as the performance in many tasks is measured purely in terms of classification accuracy and ignores additional metrics such as latency, energy consumption and computational complexity.

In this paper, we first describe a well-constrained spatio-temporal pattern classification task inspired by the sand scorpion: localizing the source of a vibration induced by tapping on a surface, using the spatio-temporal pattern detected by an array of sensors. We then present a step-by-step analysis of conventional algorithms and five different models based on spiking neural networks for classifying the data-set of spatio-temporal patterns using both supervised and unsupervised learning rules.

2. BACKGROUND

2.1. Sand Scorpion Prey Localization

Sand scorpions, such as the specimen shown in **Figure 1A**, are nocturnal predatory arachnids which, despite their primitive visual systems, can accurately locate prey such as crickets up to 50 cm away (Brownell, 1977). Brownell (1977) discovered that sand scorpions perform this feat using time-based computation based on two types of information propagated through the sand of their desert habitat: Transverse Rayleigh waves and compressional waves. Rayleigh waves travel slowly across the surface of the sand at a velocity of $\approx 50\,\mathrm{m\,s^{-1}}$ and are sensed by the scorpion's Basitarsal Compound Slit Sensilla (BCSS). Compressional waves diverge spherically from their source—traveling through the sand at $\approx 150\,\mathrm{m\,s^{-1}}$ and attenuating much more quickly than the Rayleigh waves (Brownell, 1977). Sand scorpions detect these waves using sensory hairs on their legs.

Both types of sensory organ are located on the ends of the scorpion's legs, maximizing the distance between the sensors and thus the difference in arrival time between signals measured at each one. While, theoretically, the arrival time of either type of wave could be used by the scorpion to detect the direction of its prey, Rayleigh waves travel and attenuate slower than compressional waves resulting in better range and larger time differences (1 ms rather than 0.4 ms). This intuition was supported by an ablation study in which Brownell and Farley (1979a) found that the BCSS was required for sensing direction.

As well as being able to detect the *direction* of their prey, sand scorpions can also estimate how far away it is. Brownell and Farley (1979b) suggested that the difference in amplitude of the signals received by the sensory hairs on different legs could be used to perform this computation. Here, the faster attenuation of the compressional waves is advantageous as it results in larger differences in amplitude between near and distant stimuli.

2.2. Computational Models of Spatio-Temporal Pattern Recognition

The ability to learn and recognize spatio-temporal sequences is a hallmark of biological neural information processing. Understanding spatio-temporal sequences is at the heart of object recognition, navigation and, in more general terms, all neuron-to-neuron communication. Each neuron receives a spatio-temporal pattern of pre-synaptic action potentials or spikes at its dendrites and sends output spikes to its post-synaptic partners. In the case of a single input channel, the problem of spatio-temporal sequence learning can be addressed by temporal coincidence detection (Carr and Konishi, 1990) or by temporal correlation detection (Krammer and Koch, 1997). The former approach provides binary outputs, whereas the latter approach provides a continuous output. In both cases, information is encoded in the timing of the incoming spike. On the other hand, if multiple input channels are present, spatio-temporal patterns can be represented by detecting coincidence or correlation of spikes arriving via the different input channels (Roy et al., 2016). Additionally, neurons have more options for capturing spatio-temporal patterns when multiple input channels are present. A neuron can use synaptic weight plasticity to emphasize certain channels over others, synaptic delay plasticity to delay certain input channels compared to others, or any combination of the two. To recognize spatio-temporal patterns, Gütig and Sompolinsky (2006) proposed the tempotron model in which synaptic weights are adjusted in a supervised manner, based on the deviation of the maximum (post-synaptic) voltage from the spiking threshold for wrongly

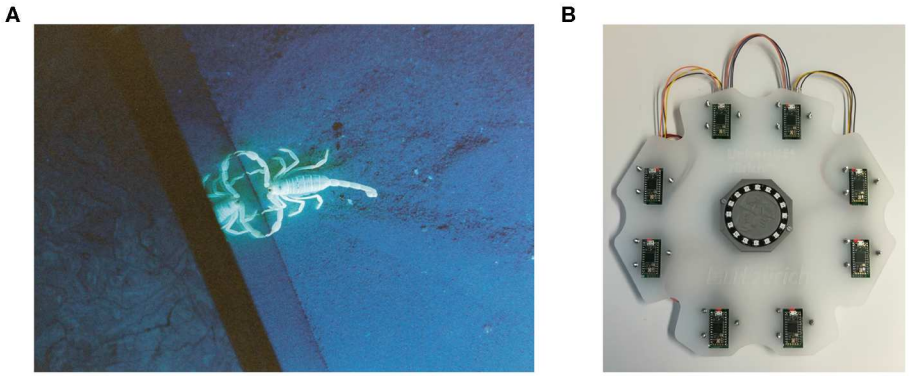


FIGURE 1 | (A) A sand scorpion in the lab. Image courtesy of Martin Reichert and Wolfgang Stuerzl. **(B)** Our prototype.

classified patterns. Roy et al. (2016) extended the tempotron approach by using an online structural plasticity mechanism in a competitive winner-takes-all (WTA) network relying on binary synapses. Alternatively, Izhikevich (2006) proposed a learning framework in which Spike-Time Dependent Plasticity (STDP) is used to adjust synaptic weights and synaptic propagation delays are randomly sampled at the beginning of the simulation and subsequently fixed. Both approaches lead to the learning of *polychronous* neural ensembles, each encoding a different spatio-temporal pattern. Wang et al. (2013) presented a hardware implementation of polychronous networks in which propagation delays are learned in a supervised manner, based on the expected firing time of the post-synaptic neuron. Another approach to learning synaptic delays is to sample synaptic time constants from a distribution and select relevant time constants via an STDP mechanism (Gerstner et al., 1996). Thus only synapses with fitting delays which trigger post-synaptic spikes are selected. In the next sections, we will present both biological and event-based mechanisms for synaptic and neuronal plasticity to learn spatio-temporal patterns.

2.3. Biological Mechanisms for Synaptic Delay Plasticity

Spikes are delivered to a neuron's post-synaptic partners through its axon with a transmission delay dictated by the axon's conduction velocity. The conduction velocity is dependent on both the diameter of the axon and the thickness of the *Myelin* sheath around it (Swadlow and Waxman, 2012). Myelin is a phospholipid substance formed by glial cells and its presence increases the conduction velocity of axons by wrapping around them and acting as an electrical insulator. Furthermore, it has recently been shown that the myelination of axons can be influenced by neural activity(Markram et al., 1997; Fields, 2015; Koudelka et al., 2016) suggesting that a form of "myelin plasticity" is also at work—something that should be taken into consideration when developing learning algorithms for spiking neural networks (Baldi and Atiya, 1994; Maass, 2001).

By optimizing conduction delays, a myelin plasticity-based model opens the way to directly learning the time dynamics of incoming spikes and extracting meaningful spatio-temporal patterns. Previous conduction delay-based algorithms have not often been tested with practical tasks such as pattern recognition and clustering (Eurich et al., 1999, 2000). The DELTRON (Hussain et al., 2012) uses the tempotron model (Gütig and Sompolinsky, 2006) to adjust conduction delays through gradient descent dynamics. Paugam-Moisy et al. (2008) extended the polychronization model developed by Izhikevich (2006) to include learnable conduction delays for classification and Wang et al. (2013) applied this approach to pattern storage. Matsubara (2017) developed a probabilistic delay learning model which adjusts conduction delays and synaptic weights. However, Matsubara used this to classify time-invariant databases such as MNIST, which have no temporal structure making them a poor choice for evaluating computation based on spike timing.

2.4. Event-Based Spatio-Temporal Pattern Recognition

The task solved by the sand scorpion can be described more generally as spatio-temporal pattern classification and recently, two complementary approaches, specifically designed for event-based sensory signals, were proposed. Both approaches feature homogeneous and fixed synaptic time constants and adapt synaptic weights to cluster spatio-temporal patterns. In the following subsections, we detail these two approaches.

2.4.1. HOTS: A Hierarchy of Event-Based Time-Surfaces

Lagorce et al. (2017) proposed an algorithm in which events are converted into a continuous-valued time surface. This approach can be understood as convolving events within a pre-defined region of interest (ROI) with an exponential decaying kernel, with the reference time being the time of the central event in the ROI. These spatio-temporal contexts are then matched to learned features using online learning, offline clustering or other methods. HOTS can be employed in a hierarchical fashion, with an increasing ROI size and time-constant for the exponential kernels and has been successfully applied to variety of classification tasks (Cohen et al., 2016; Afshar et al., 2019a).

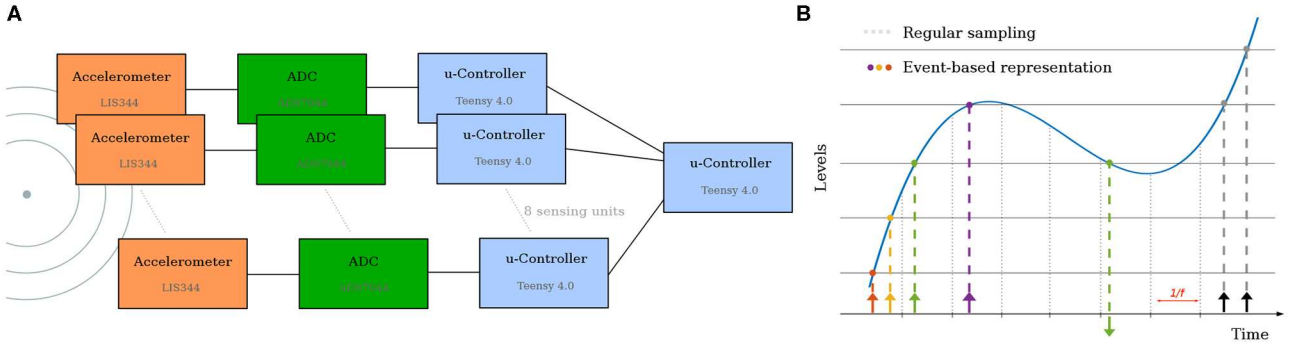


FIGURE 2 | (A) Architecture of the electronic solution. Each accelerometer is read by its own Analog-to-Digital Converter. The local microcontroller receives the samples, applies level-crossing detection and send the master controller the spikes. **(B)** The level crossing sampling method employed, and comparison to regular sampling. In this work, only the first spike is used.

2.4.2. FEAST: Event-Based Feature Extraction Using Adaptive Selection Thresholds

To guarantee that all feature detector neurons are used equally when clustering time-surfaces, Afshar et al. (2019b) extended HOTS to feature coupled, adaptive thresholds. Every time a given feature detector emits a spike, its threshold is increased. This results in non-updated feature detectors being more likely to capture the next time-surface and means that all feature detector neurons are equally active across the data set[1]. If no feature detector captures the present time-surface, however, all thresholds are decreased. The adaptation of firing thresholds can be understood as a homeostatic plasticity mechanism (Turrigiano and Nelson, 2004; Qiao et al., 2016, 2017). In the context of continual learning, this "global"[2] threshold adaptation might prevent convergence if unrecognized patterns are common (Afshar et al., 2014).

3. METHODS

3.1. Neuromorphic Tactile Sensor Design

The problem of spatially localizing a stimulus on a 2D surface is well-defined with 5 sensors (Mahajan and Walworth, 2001; Hu and Yang, 2010). However, having an array of more than 5 sensors adds robustness to the system, so we developed the prototype shown in **Figure 1** with (arbitrarily) 8 sensors. A circular configuration of the sensor array would lead to badly conditioned cases—as depicted by Mahajan and Walworth (2001)—so our 8 sensors are arranged in the non-circular manner shown in **Figure 1B**. An acrylic plate makes a rigid connection between the 8 sensors. As the system is statically overconstrained (5 redundant contact points), a slightly flexible acrylic plate was chosen to ensure that all 8 sensors could still touch the surface if there was any fabrication misalignment.

Each sensing unit consists of a Piezoelectric accelerometer for sensing vibrations and a local microcontroller-based processing unit (Teensy 4.0, ARM Cortex-M7) which reads samples from the Analog to Digital converter at 1 MHz, and then applies a level crossing detection to generate events (Astrom and Bernhardsson, 2002) (**Figure 2B**. All 8 sensors then transmit these events to an additional central processing unit which solves the analytical problem using the approach described in section 3.2.1 and saves the data for dataset creation (**Figure 2A**).

While in desert sand, a 1 ms resolution would be sufficient (Brownell, 1977), in order to work on more common mediums—which typically have faster propagation speed—we need higher temporal accuracy. Depending on the surfaces used in our experiments, a wave propagates at a speed between 200 and 300 $m\,s^{-1}$, which result in a propagation time between 1 and 1.5 ms in our setup, between radially opposite sensors. However, standard accelerometers with digital output are limited to sampling rate of only a few kHz, so we decided to use an accelerometer with an analog output (*STMicro LIS344*), combined with a separate 1 MHz Analog to Digital converter (*Texas Instruments ADS7044*).

Figure 2 illustrates the architecture of the electronic solution, as well as the spike generation method. This approach of fast sampling followed by level-crossing detection was chosen for it flexibility (different encoding schemes could be tested). However, other approaches such as the one introduced by Lee et al. (2017), or the VLSI event-generator proposed by Corradi and Indiveri (2015) could be used. While these might reduce the complexity of the sensing unit and (possibly) increase the time precision, this would come at the cost of reduced flexibility.

Using this sensor, we recorded a dataset consisting of 10 repetitions of 32 stimuli (8 different angles, every 45°, and 4 distances (200, 400, 600, 800 mm). The stimuli, i.e., the surface vibrations, were induced by tapping with the index finger on a wooden surface.

3.2. Algorithms

In the following section we will present five different solutions to the problem of localizing the position of a vibration induced by tapping on a rigid surface. Not all approaches are entirely successful but, nonetheless, we hope to provide interesting concepts and ideas which try to emphasize how to extract task relevant information from the timing of incoming events.

[1]The occurrences of examples in the data set need to be balanced.
[2]Global in this context refers to the population of feature detectors at a given level in a hierarchy, not across a hierarchy of feature detectors.

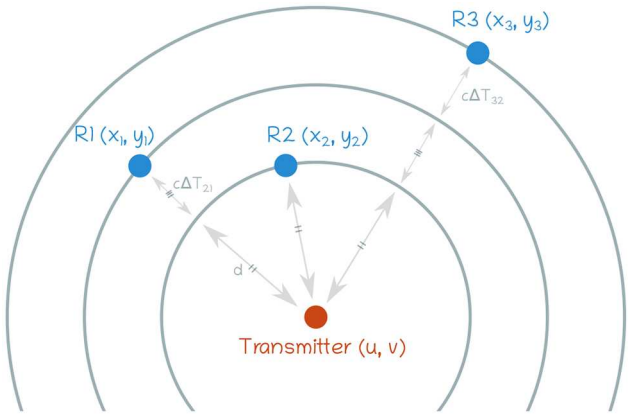


FIGURE 3 | Definition of the problem. For the sake of simplicity, only 3 sensors are shown here. The transmitter can be seen as being in the center of concentric circles.

We selected these algorithms to represent varying levels of complexity and biological plausibility as well as because they each require different amounts of information. Specifically, in section 3.2.1 we will first demonstrate how to localize the position of the tap analytically if the geometry of sensory array and the propagation speed are known. Then in sections 3.2.2 to 3.2.6, we will present more and more biological plausible implementations which try to solve the task with less and less external information.

While these algorithms do not represent a complete list of possible solutions, we still hope to provide the reader with a thorough analysis of several approaches for computation based on the precise timing of spikes as well as outlining some of the challenges the community needs to overcome to perform such computation using event-driven SNNs. More importantly however, we hope to provide a starting point for the development of novel algorithms, as well as providing a benchmark task for further comparison and evaluation.

3.2.1. Analytic Solution

The position of the source can be estimated based on the Time Difference Of Arrival (TDOA) between each pair of sensors. The 2D problem is shown in **Figure 3** and, given the sensor spatial positions $R_i(x_i, y_i)$ and the TDOA for each pair of sensors, the source localization (u, v) and the propagation speed c in the chosen medium can be retrieved as follows:

$$\underbrace{\begin{bmatrix} x_1 - x_2 & y_1 - y_2 & -\Delta T_{12} & -\Delta T_{12}^2 \\ x_1 - x_3 & y_1 - y_3 & -\Delta T_{13} & -\Delta T_{13}^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_1 - x_n & y_1 - y_n & -\Delta T_{1n} & -\Delta T_{1n}^2 \end{bmatrix}}_{A} * \underbrace{\begin{bmatrix} u \\ v \\ cd \\ c^2 \end{bmatrix}}_{X} = \underbrace{\begin{bmatrix} x_1^2 + y_1^2 - x_2^2 - y_2^2 \\ x_1^2 + y_1^2 - x_3^2 - y_3^2 \\ \vdots \\ x_1^2 + y_1^2 - x_n^2 - y_n^2 \end{bmatrix}}_{B} \tag{1}$$

for N sensors where $A \in \mathbb{R}^{N\times 4}$, $X \in \mathbb{R}^{4\times 1}$ and $B \in \mathbb{R}^{N\times 1}$. This equation can then be solved using the pseudoinverse A^+ of A. Because A^+ has to be evaluated every time a stimulus is presented, we can exploit the fact that our matrices are well defined [rank(A) being equal to the number of columns of A] and therefore:

$$A^+ = (A^T A)^{-1} A^T \tag{2}$$

allowing us to minimize the required computation and find an analytic solution to the problem, using bloc decomposition for the inverse $(A^T A)^{-1}$, given the fact that $A^T A$ is a squared matrix ($A^T A \in \mathbb{R}^{4\times 4}$). An alternative approach would be to iteratively estimate the pseudoinverse, following the method described by Tapson and van Schaik (2013).

3.2.2. Temporal Coincidence Detection

A simple way to detect a particular position is to have a have a neuron associated with every target position, connected to each receptor with delayed synapses. In this set-up, each neuron receives one spike from every receptor and must only spike if the input came from the right place. The natural way to ensure that the receptive neuron will indeed cross the threshold is to have the spikes arrive at the same time, so that all the incoming spikes coincide and create a large increase in membrane potential.

Specifically, a decoding neuron at position p has N_s synapses, each with a corresponding delay $d_{p(k)}$, and parameters τ and θ, corresponding to the decay constant and the firing threshold of the neuron. For all input spikes to arrive simultaneously, we must associate the vibration wave generate at each position p to a delay vector $\mathbf{d_p} \in \mathbb{R}^{N_s}$. The sub-threshold membrane potential of the decoding neuron is then

$$v_p(t) = \sum_{k=1}^{N_s} e^{-\frac{t - t(k) - d_p(k)}{\tau}} \Theta(t - t_k - d_p(k)) \tag{3}$$

where the exponential corresponds to the decay of the membrane potential and Θ is a step function that ensures that the input is only relevant after it arrived at the detector neuron at time $t(k) + d_p(k)$, where $t(k)$ is the time of arrival of the ground vibration at the detector k and $d_p(k)$ is the delay associated with synapse k. The leaky integrate-and-fire neuron will spike, indicating a stimulus at position p, if

$$v_{\max} > \theta, \quad v_{\max} = \max_t v_p(t). \tag{4}$$

Hence, we will try to maximize the value of $v_{\max}$. Since the exponential decay term in Equation (3) implies that each input spike is strongest at its arrival so, if we want to maximize the membrane potential, we must make sure that all those spikes arrive simultaneously, which can be achieved by setting

$$d_p(k) = d_p^* - t_p(k) \tag{5}$$

where $t_p(k)$ is the kth entry of the vector $\mathbf{t}_p \in \mathbb{R}^{N_s}$ which corresponds to time of arrival of the ground vibration from position p to each sensor k, and d_p^* is a value that ensures that

all the transmission delays are positive. Naturally, there are many possible values of d_p^*, but for simplicity we will set

$$d_p^* = \max_k \ t_p(k), \tag{6}$$

which implies that all the spikes arrive when the last spike is detected—as it is impossible to receive all of them earlier than that.

In the ideal case—where the spikes from a given position are perfectly timed—$\theta = N_s$, so that, when the spikes arrive exactly at t_k, they will all add up and the membrane potential will cross the threshold. However, if the spike times vary even slightly, the fast decay will result in a sub-threshold membrane potential and the neuron would not fire.

In the real world, the ideal case is unlikely so we must account for the possibility of jitter by associating a target detector with an area rather than a point. Assuming that τ is fixed, we must simply select the value of θ that minimizes our classification error

$$e = \Pr\left[v_{\max} < \theta | p\right] + \Pr\left[v_{\max} > \theta | \neg p\right], \tag{7}$$

which is simply the sum of the probabilities of false negatives and false positives. The simplest way to do this is to realize that $\Pr\left[v_{\max} < \theta | p\right]$ increases monotonically with θ, while $\Pr\left[v_{\max} > \theta | \neg p\right]$ decreases monotonically with θ. Hence, the computation of the optimal θ from a sample of m examples, where m_p examples were from position p and $m_{\neg p}$ were not, can be done through a simple algorithm:

for all m examples **do**
  compute $v_{\max}$
  if example position = p **then**
    add the tuple $\left(v_{\max}, d = 1\right)$ to list L
  else
    add the tuple $\left(v_{\max}, d = -1\right)$ to list L
  end if
end for
Sort L by $v_{\max}$, high to low.
$e \leftarrow m_p$
for every tuple in L: **do**
  $e \leftarrow e - d$
  Add the tuple $(v_{\max}, e)$ to list M
end for
select the tuple with lowest e in M
Set $\theta \leftarrow v_{\max}$ from the tuple with lowest error

This approach gives us a simple way of using the leaky integrate-and-fire nature of neurons to achieve the desired detection as long as we can compute the appropriate delays *a priori*.

3.2.3. Complex Weights and Delays

The previous approach, while fundamentally correct, requires precise knowledge of the delays. If sensors or synapses have systematic measurement errors or there is significant jitter, it could be impossible to find delays $d_p(k)$ that would be able to fully compensate for the effects of noise. Furthermore, unreliable sensors or synapses should be given less importance than if perfect noiseless sensors or synapses. In this section we present a statistical method for computing the delays and associated weights to address this issue (State, 2019).

First we must redefine our leaky integrate and fire neuron model, described in Equation (13), to include synaptic weights:

$$v_p(t) = \sum_{k=1}^{N_s} w_p(k) e^{-\frac{t - t(k) - d_p(k)}{\tau}} \Theta(t - t_k - d_p(k)), \tag{8}$$

While our new synapses now have two parameters ($w_p(k)$ and $d_p(k)$), the logic from the previous section remains the same and our goal is to force spikes to arrive as synchronously as possible. In order to manipulate the spike times algebraically, we encode the input spike train – here consisting of a single spike per neuron – into N_s variables that can be studied using linear algebra. We do this by encoding spikes as phases of a complex number so each spike

$$s(k) = e^{\frac{j\pi(t(k) - t_0)}{T}} \tag{9}$$

where t_0 is the time at which the first spike of a wave arrives (so that the time of the input wave is not considered) and j is the imaginary unit. Encoding time in the phase of a complex number is a known trick when dealing with spikes, often used in *phasor* networks (Hirose, 1992; Reichert and Serre, 2013; Frady and Sommer, 2019). T is the maximum time during which we can receive spikes and is given by

$$T = \frac{2 r_{\max}}{c} \tag{10}$$

where c is the wave speed and $r_{\max}$ is the radius of the sensors, meaning that the numerator is two times the maximum distance between two sensors. The value of T ensures that the phase of $s(k)$ is in the interval $[0, \pi]$, which is necessary to avoid geometric inconsistencies (State, 2019).

Now we can use least squares regression to obtain the delays and weights associated to each synapse. Thus, for every position p we will have

$$\epsilon = \frac{1}{N_e} \sum_{i=1}^{N_e} \| y_i - \sum_{k=1}^{N_s} \hat{w}_p(k) s_i(k) \|^2 \tag{11}$$

where N_e is the number of examples, indexed over i and y_i corresponds to the desired output of the perceiving neuron: one if the spikes were generated by a tap at position p and zero if the spikes come from a tap somewhere else.

Once we find the weight vector for position p, $\hat{\mathbf{w}}_p = \left[\hat{w}_{p(1)}, \hat{w}_{p(2)}, ..., \hat{w}_{p(n)}\right]$, it will give us weights with complex entries. Naturally, this is not something we can put on a synapse, but rather a complex number that somehow relates s to its appropriate synapse. To obtain the delays and weights, we inverse the operation done in Equation (9) and obtain the delay from the phase and the weight from the absolute value,

$$\hat{w}_p(k) \rightarrow w_p(k) = |\hat{w}_p(k)|, \ \ d_p(k) = \frac{T}{\pi} \arg \hat{w}_p(k). \tag{12}$$

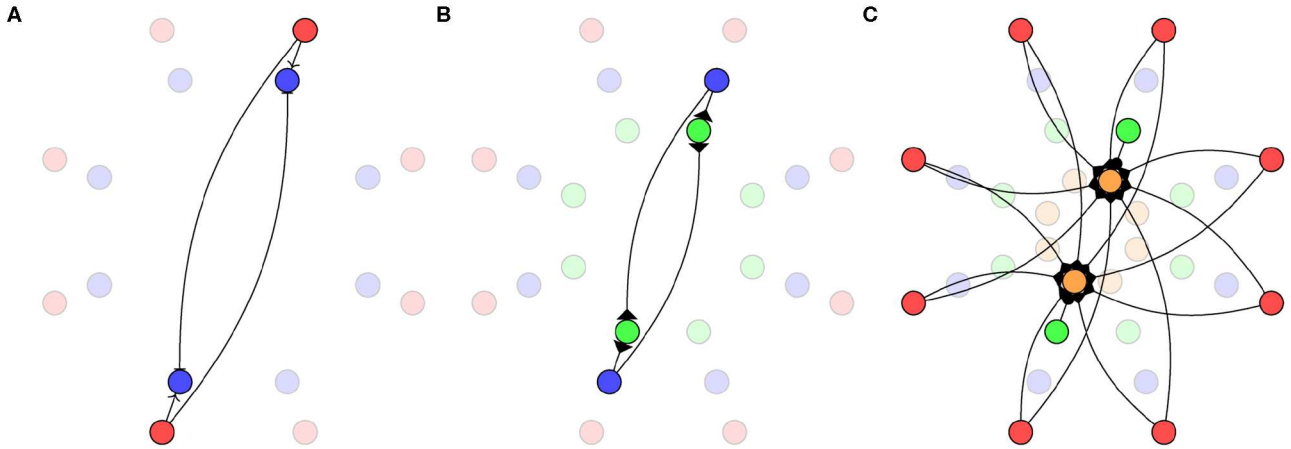


FIGURE 4 | Biomimetic network architecture. **(A)** Connections from sensors (red) to TDE neurons (blue). **(B)** Connections from TDE to inverse direction neurons (green). **(C)** Connections from sensors and inverse direction neurons to direction neurons (orange).

It is worth noticing that the conversion from a complex weight to a weight and a delay used here ensures that all the weights are positive. This means that all synapses are excitatory, and is a natural consequence of the encoding chosen originally in Equation (9).

To understand this procedure, it is useful to look at value of ϵ. When $y_i = 1$, the sum of the input to the neuron $\sum_{k=1}^{n} \hat{w}(k)s_i(k)$ should be real and positive whereas, when $y_i = 0$, it should be close to zero. In the ideal case, $\hat{w}(k)$ will have exactly the same phase as $s_i(k)$ but the opposite sign, meaning that the product $\hat{w}(k)s_i(k)$ must be real and positive and the phases somehow uniformly distributed in $[0, 2\pi]$ when $y_i = 0$ so that $\sum_{k=1}^{n} \hat{w}(k)s_i(k)$ adds up to zero. Just as the delays were converted into phases in Equation (9), the phases must now be converted back into delays so that, when the phases of $\hat{w}_p(k)$ and $s(k)$ cancel each other, the delay of the synapse also cancels out the delay of the spike. The weights are also easy to interpret: the more reliable the value $s(k)$ is for a certain position p, the higher $|\hat{w}_p(k)|$. This is because the least squares regression will "learn" that every time $s(k)$ has a specific value and the product $\hat{w}_p(k)s(k)$—which is already real and positive due to the phase cancellation—should approach $y = 1$ and hence be large.

As in the previous section, this complex conversion trick is simply a way to synchronize the arrival of spikes at the neuron encoding position and therefore we still need to compute the θ_p for every neuron, for which we can, again, use Algorithm 3.2.2. It is also worth noticing that using the complex formulation intrinsically assumes that the spikes have the shape of a cosine, as opposed to a decaying exponential (Reichert and Serre, 2013; State, 2019), meaning that it is more appropriate to use a non-instantaneous synapse such as an EPSP (Takagi, 2000) with a flat value at the maximum such that the first derivative is the same; however, this does not affect our results.

The advantage of this approach compared to similar complex formulations (Reichert and Serre, 2013; Shrestha and Orchard, 2018) lies in the use of classical linear algebra. Besides being very data efficient—as a single example would yield a solution just as well as a combination of examples—this approach easily handles cases where spikes are unreliable (State, 2019), something that is often difficult when using delays directly and it is resistant to over-fitting because the pseudoinverse guarantees that the weights will have the lowest possible modulus. However, its simplicity also make it less flexible, as it does not deal with multi-spike problems (Taherkhani et al., 2015; Shrestha and Orchard, 2018) nor does it work for SNNs with hidden units (Hirose, 1992; Frady and Sommer, 2019) as the linear algebra solution requires specific values as outputs, rather than step-by-step error feedback.

3.2.4. Temporal Difference Encoders

The approaches presented in the previous sections encode target position using individual neurons to represent each point in space. While this encoding allows for precise localization, it requires a large number of neurons. In this section, we will take inspiration from the ring-like neural structures present in the sand scorpion (Stürzl et al., 2000) to develop an alternative solution based on Temporal Difference Encoder (TDE) neurons (Milde et al., 2018) which requires many fewer neurons[3].

In this model, each pair of opposite sensors is connected to an inner ring of 8 TDE neurons (Milde et al., 2018) as shown in **Figure 4A**. The sub-threshold behavior of the TDE neurons is modeled as a leaky integrate-and-fire neuron:

$$\tau_m \cdot \frac{dV(t)}{dt} = E_L - V(t) + R_m \cdot I(t) \tag{13}$$

[3]The TDE was originally named the spiking Elementary Motion Detector (sEMD) as it was designed to extract relative motion cues from spatially adjacent pixels of an event-based vision sensor. However, the computation performed by these neurons is of much more general nature as it calculates the temporal correlation of two events based on the difference in timing, irrespective of the sensory modality.

where E_L denotes the resting potential of the neuron, R_m is the membrane resistance, $I(t)$ is the injected current at time t, and τ_m is a decay constant. These neurons are then driven by an input current $I(t)$ such that:

$$I(t) = \begin{cases} I_{trig} \cdot f & \text{if } f > 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where f represents a dimensionless "facilitating" input and I_{trig} represents the "trigger" input current. Both f and I_{trig} are exponentially shaped such that:

$$\tau_{syn} \cdot \frac{df}{dt} = -f \qquad \tau_{syn} \cdot \frac{dI_{trig}}{dt} = -I_{trig} \tag{15}$$

where τ_{syn} is the time constant of the synaptic dynamics. When a spike is received at a facilitating or trigger synapse, the synaptic weight (w_{fac} and w_{trig} respectively) is added to the appropriate input:

$$f \leftarrow f + w_{fac} \qquad I_{trig} \leftarrow I_{trig} + w_{trig} \tag{16}$$

The dynamics described by Equations (14)–(16) result in an input current which is scaled non-linearly depending on the time difference between spikes arriving at the facilitating and trigger synapses. Additionally, as I_{trig} is "gated" by f, these synapses are also direction-selective. The time difference between the inputs received at the two opposite sensors will be largest when a stimulus is located on the line connecting them and smallest when it is on the line perpendicular to this meaning that the inner ring of TDE neurons will encode the direction of the stimuli as a vector in an over-complete 8-dimensional space.

While the directional information required could be decoded from the activity of the TDE neurons, the desired output for this system is a ring with a single active neuron identifying the direction of the stimuli. In order to achieve this, we connect the TDE neurons to a second ring of "inverse direction" neurons using the excitatory connections shown in **Figure 4B**. Weak connections from the TDE neuron to the adjacent inverse direction neuron and strong connections to the opposite inverse direction neuron result in this population of neurons having a minimum of activity in the direction of the stimulus.

Finally, the inverse direction neurons are connected to the innermost ring of "direction neurons" with inhibitory connections as shown in **Figure 4C**. These neurons are additionally provided with background excitation—direct from the sensors—tuned to produce a "1-hot" encoding of the stimuli direction. In order to maximize the accuracy of this encoding, we use 16 neurons in this ring with inhibitory weights calculated as:

$$w_{ij} = w_{peak}.max(0, cos(\theta_i - \phi_j)) \tag{17}$$

where w_{peak} is the peak inhibitory weight, θ_i is the angle of the sensor adjacent to inverse direction neuron i and ϕ_j is the angle of the direction neuron j. While this approach does not currently provide an estimate of distance, if magnitude information were available, this could be provided in place of the excitatory input to the direction ring.

3.2.5. Synaptic Delay Plasticity

So far we addressed how one can adjust neuronal (Θ), axonal and synaptic (w, τ) parameters if the geometry of the sensor array is known or, in the absence of that information, if a set of training examples is given. Given this information, we have outlined how these parameters can be optimized so that one can localize the position of a vibration source even in the presence of temporal jitter. While we used biologically motivated neuron and synapse models to perform the *computation*, the *optimization* of the parameters was done offline using conventional regression algorithms such as the least square method. Such optimization procedures require non-local information[4] such as the neuronal firing thresholds of other neurons, the onset of the stimulus and the position of the stimulus itself. The decision on where the tap originated from is being made through adapting neuronal firing/spiking thresholds such that the designated spatio-temporal pattern is matched. In the subsequent sections we are going to address the localization task by applying three constraints on our model:

1. Only information which is local to a given pre-post synaptic neuron pair is used to update synaptic parameters.
2. No a priori knowledge of the sensory system is required.
3. The model parameters must be updated in an unsupervised manner.

Drawing inspiration from the myelin plasticity discussed in section 2.3 and from previous work on delay shifts (Hussain et al., 2012; Wang et al., 2015), in this section we will extract temporal features by modulating conduction delays.

The proposed model uses gradient descent dynamics to synchronize spikes emitted by pre-synaptic neurons, by adjusting delays on the most recently active synapses within an experimentally set temporal window. Whenever a neuron fires, mutual inhibition is used to ensure that neurons specialize to a particular temporal pattern.

The delay plasticity model works in conjunction with leaky integrate-and-fire (LIF) neurons described in Equation (13). We chose an exponential excitatory post-synaptic current (EPSC) shape such that the input current $I(t)$ at time t is:

$$I(t) = I_{inj} \cdot \sum_i w_i \cdot e^{-\frac{t-s_i}{\tau_{syn}}} \cdot \mathcal{H}(t) \tag{18}$$

where I_{inj} is the injected current every time a neuron fires, w_i is the synaptic weight of synapse i, τ_{syn} is the synaptic time constant, and $\mathcal{H}(t)$ is the Heaviside step function.

When we study the dynamics of a single synapse i, we remove the discontinuities caused by the input signal by focusing on the range $[s_i, t]$ where $\mathcal{H}(t) = 1$, s_i being the time of arrival of a spike to a post-synaptic neuron. Assuming initial conditions such that $V_i(s_i) = E_L$ as we are restricting the network to only one spike per synapse, and the membrane potential is reset between

[4]Local information is defined as to have direct access from a neuron perspective, e.g. post-synaptic density (estimate of synaptic weight), average firing rate (calcium concentration) or its own spiking threshold.

each training example through a WTA algorithm, the membrane equation now has a solution:

$$V_i(t) = E_L + \frac{R_m \cdot I_{inj} \cdot w_i \cdot \tau_{syn}}{\tau_m - \tau_{syn}} \cdot \left(e^{-\frac{t-s_i}{\tau_m}} - e^{-\frac{t-s_i}{\tau_{syn}}} \right) \quad (19)$$

The time course of the potential follows a bi-exponential model with a finite rising time. In order to maximize the membrane potential of a post-synaptic neuron—and ultimately associate it with a particular temporal pattern—we compute the gradient of the neuron's potential $\frac{\partial V(t,s_i)}{\partial s_i}$ and modulate d_i until all spikes are aligned. The model assumes only one spike per synapse. The partial derivative of $V(t, s_i)$ with respect to s_i can then be written as:

$$\frac{\partial V(t,s_i)}{\partial s_i} = \frac{R_m \cdot I_{inj} \cdot w_i}{\tau_m - \tau_{syn}} \cdot (e^{-\frac{t-s_i}{\tau_m}} - e^{-\frac{t-s_i}{\tau_{syn}}}) \quad (20)$$

The delay update rule can be represented by the following equation:

$$d_i^{t+1} = d_i^t + \eta \cdot \frac{\partial V(t,s_i)}{\partial s_i} \quad (21)$$

where η represents the learning rate of a neuron, with $\eta > 0$. We decay the learning rate across iterations to avoid large gradient steps.

3.2.6. Structural Plasticity

In this section we propose a neurally implemented, self-organizing, balanced network of excitatory and inhibitory (EI) neurons which fulfills the constraints outlined at the beginning section 3.2.5 by combining event-based STDP (Song et al., 2000) with a variant of structural plasticity (Bekkers, 2011; Knoblauch et al., 2014) and adaptive spiking thresholds (Afshar et al., 2019b). It has been demonstrated that the combination of STDP and a competitive EI network (i.e., WTA), in the context static inputs can account for disparity selectivity (Chauhan et al., 2018), in the context of non-static inputs account for the observed development of orientation selectivity (Masquelier, 2012), and even the formation of temporal memory (Kappel et al., 2014). These mechanisms become especially powerful when sensory information is encoded using relative latency, i.e., using a temporal code.

The EI network consists of N neurons of which 80 % are excitatory and 20 % are inhibitory[5]. N depends on the desired spatial resolution the network should be able to decode. Each excitatory neuron is connected via simple alpha synapse (Rall, 1967) with the following dynamics

$$I_{syn}(t) = \bar{I}_{syn} \frac{t-t_0}{\tau_{syn}} \exp\left(1 - \frac{t-t_0}{\tau_{syn}}\right) \quad (22)$$

[5]The excitatory neuron pool has $N \times k$ incoming excitatory plastic synapses (see Equations 24 and 26), whereas the connectivity probability $E \rightarrow I$ and $I \rightarrow E$ is set to 0.7 with random and fixed synapses.

where I_{syn} is the EPSC, $\bar{I}_{syn}$ is the peak EPSC and τ_{syn} is the synaptic time constant. Given a pre-synaptic spike at t_0, I_{syn} is updated as follows

$$I_{syn} = I_{syn} + w, \quad (23)$$

where w is the synaptic weight which is modified according to a STDP protocol:

$$\Delta w = \begin{cases} a_w^+ \cdot e^{\frac{t_{pre}-t_{post}}{\tau^+}}, & if\ t_{pre} \leq t_{post} \\ a_w^- \cdot e^{\frac{t_{pre}-t_{post}}{\tau^-}}, & if\ t_{pre} > t_{post}, \end{cases} \quad (24)$$

where a_w^+ and a_w^- represent the magnitude of increments and decrements to the weight and can be seen as the learning rate of the plasticity mechanism. t_{pre} and t_{post} are the times at which the pre- and post-synaptic neuron emitted a spike and $\tau^{+/-}$ defines the temporal window within which spikes result in weight changes. Such "additive" STDP updates (Song et al., 2000) often result in bimodal weight distributions with all weights ending up either at 0 or their maximum value. There are numerous approaches to addressing this problem (Goodhill and Barrow, 1994; Morrison et al., 2008), but here we implement an event-driven weight decay which is triggered whenever the post-synaptic neurons emits a spike. The weight decay depends on the synaptic weight and is calculated as follows:

$$w = w - (w \cdot \kappa \cdot \eta), \quad (25)$$

where κ is the ratio between weight increment and decrement ($\kappa = \frac{a_w^+}{a_w^-}$, for balanced STDP protocol $\kappa = 1$) and η is the decay rate ($\eta << 1$). The objective of the STDP learning rule is to detect spatio-temporally correlated activity in the input spike trains. Here we pair STDP for synaptic weights with a STDP rule for synaptic time constants. This plasticity rule aims to find a set of synaptic time constants which, given the post-synaptic activity, increase the overlap in synaptic input currents across the 8 input channels. The time constants of channels which transmit spikes early in the sequence are increased, whereas the time constants of channels which transmit spikes late in the sequence are decreased. This plasticity rule has the consequence that a neuron spikes as early as possible to a given spatio-temporal pattern given the provided competition of the other neurons in the EI network. The synaptic time constants τ are updated as follows:

$$\Delta\tau = a_{\tau_{syn}} \cdot \frac{(\Delta t - s) \cdot e^{-\left\| \frac{\Delta t - s}{\tau^* - s} \right\|}}{(\tau^* - s) \cdot e^{-1}}, \quad (26)$$

where a_τ is the learning rate of the synaptic time constant and is set such that the time scale of changes in the synaptic time constants is much slower than the time scale of weight plasticity ($a_\tau << a_w^{+/-}$). Δt is calculated based on pre- and post-synaptic spike timing ($\Delta t = t_{pre} - t_{post}$), τ^* determines the peak in change of the time constant with relative to the offset s ($\tau^* > s$).

The plasticity kernel for synaptic weights and time constants are depicted in **Figure 5C**.

The continuous changes to synaptic weights and time constants are combined with a form of structural plasticity,

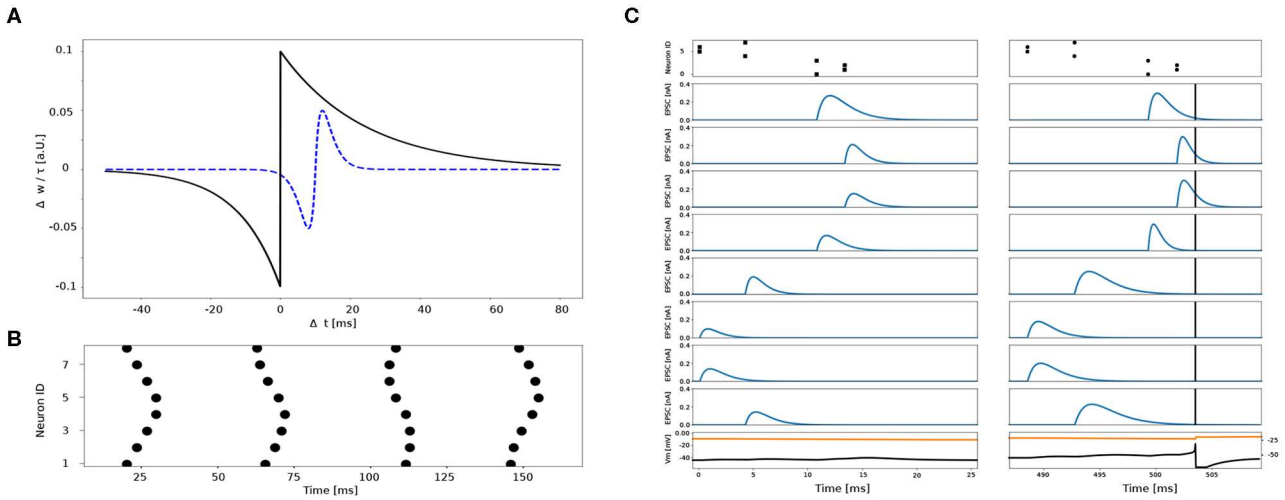


FIGURE 5 | (A) Kernels of plasticity. Change in synaptic weights Δw (black, solid) and time constants $\Delta\tau$ (blue, dashed) as a function of Δt. **(B)** Four example spatio-temporal patterns elicited by eight different sensors. **(C)** Example synaptic traces to the same spatio-temporal pattern before (left) and after (right) training. Top panel shows the spatio-temporal pattern across eight channels. Blue traces show the excitatory post-synaptic currents (EPSCs) of 8 different synapses connected to the 8 sensors. Black vertical bar indicates the post-synaptic spike time. Black trace in the bottom panel is the post-synaptic membrane potential plotted in conjunction with its threshold (orange trace on top).

operating on slower time-scale. If the average weight of all incoming synapses is below a user defined threshold, this rule deletes all (8) synapse and "re-spawns" a new set with a random weights and time constant sampled from a Gaussian distribution. This can be interpreted as modeling the retraction of a dendritic branch and its replacement by another at a different location hence its different time constant and strength (Zito et al., 1999). In this way, the synaptic weights and time constants are sampled and afterwards adjusted according to Equations (24) and (26) until each excitatory neuron within the EI network finds a unique set of w and τ_{syn}. Thus, each neuron is sensitive to particular spatio-temporal pattern which, in the case of this task, represents a particular location (see **Figure 5C**). To prevent each neuron from learning multiple patterns we install adaptive thresholds, similar to (Afshar et al., 2019b). However, the adaption of the firing threshold depends only on the post-synaptic membrane potential, which reflects the networks activity indirectly via the recurrent connections.

4. RESULTS

In the following subsections, we will present the results obtained using each of the algorithms discussed in section 3.2.

4.1. Analytical Solution

Because the matrix A in Equation (1) is not constant, we need to compute its pseudo-inverse at run-time in order to use the approach presented in section 3.2.1. While it would be possible to perform this calculation on the microcontroller (either direct pseudo-inverse computation, or iterative methods), it is beyond the scope of this paper. Instead, we performed the calculation of this pseudo-inverse offline on the host computer. Due to the temporal resolution of our system, the positional error (**Figure 6A**) is significant (73.4% accuracy) while angles are recovered reliably in almost all trials (99.7%). Nonetheless, 80 % of the position error is less than 20 mm.

After differentiating Equation (1), it appears that our mechanical implementation has 2 ill-conditioned points, at 90 and −90 degrees, as shown in **Figure 6B**. As **Figure 6C** illustrates, the angular precision is not affected by this problem.

4.2. Temporal Coincidence Detection

To compute the delays associated with every point and sensor, we first need to calculate the propagation speed of the vibration wave on wood. The simplest way to do this is to use linear regression on the distances between points and sensors as one variable and the arrival times as the second. From our recordings we get 2.560 spike times which, as we show in **Figure 7**, we can match to their corresponding delay. This gives us a speed of 126 m s^{-1}, which we can then use to compute the delays.

By using the delays calculated by dividing the distance by the propagation speed we find that we only able to recover 37.5 % of the positions successfully. Within those errors, the angles are always perfectly recovered, but the distances are not. Although the distance recovery is better than chance—which would be 25 %—the fact that the coincidence detection would mistake different distances implies that there are errors that we are not accounting for. This is not surprising given the variance around the regression line shown in **Figure 7**.

Interestingly, the percentage of errors remains the same when we change the number of measurement per point, hinting that the error in delays is not stochastic but rather systematic. This

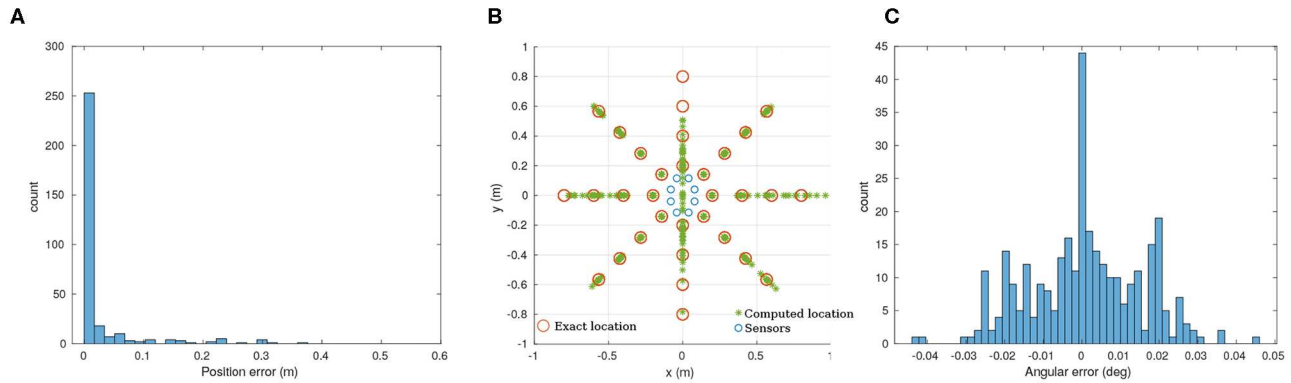


FIGURE 6 | Analytic solution for the recorded data. **(A)** Position error to the ground truth. **(B)** Overview of the results for the full dataset (320 points). **(C)** Angular error. 80% of the datapoints have a position error lower than 20 mm. The angular error is lower that ±0.04 degrees. The mechanical design of our prototype presents two ill-conditioned points at 90 and −90 degrees (along a vertical line in this case).

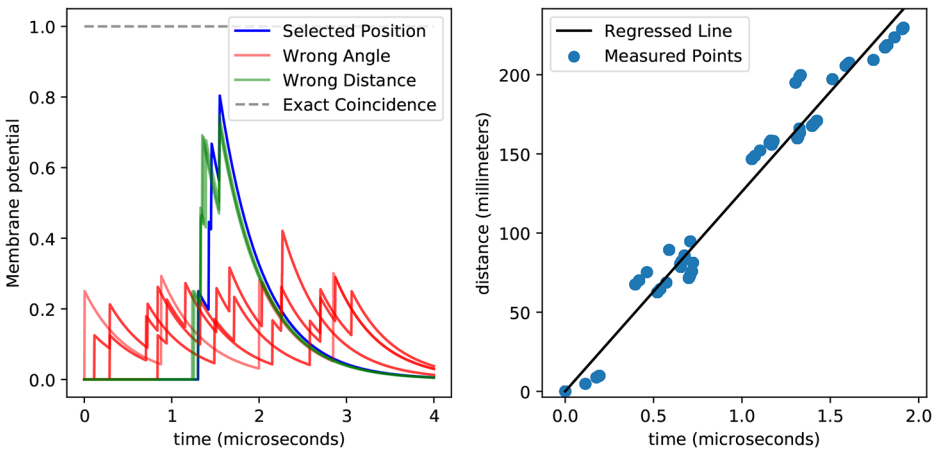


FIGURE 7 | This figure shows the membrane potential for a position detector with temporal coincidence detection **(left)** and the relationship between spiking delays and distances **(right)**. The left plot shows how spikes generated from a tap at the preferred position arrive synchronously at the detector neuron and induce a high membrane potential (blue), while spikes that arrive from taps at other positions (green and red) achieve lower membrane potentials. This plot highlights the difficulty in differentiating taps coming from the same angle but different distances (green vs. blue). Also, spikes do not arrive precisely at the same time, which prevents the membrane potential from reaching the theoretical upper bound (gray). The general relationship between distance and delay is easy to see in the right plot, and can be used to obtain the speed from the slope of the regressed line (black), while the variance around that line accounts for the sub-optimal membrane potentials obtained in the left plot. It is worth noticing that here we displayed 2560 points, yet they all fall into a few dozens of clusters, meaning that the errors are systematic rather than stochastic.

can be observed in **Figure 7**, where despite having 2,560 points we only see a few of them, meaning that the measurements are systematically biased.

4.3. Complex Weights and Delays

Knowing that there is a systematic but unknown bias on the sensors implies that we must resort to techniques drawn from statistics and machine learning rather than purely analytical solutions which could extract the information of the biases automatically. As expected, using the linear regression in the complex domain yields perfect recovery of the points, implying that the systematic biases in the recorded times are not necessarily an impasse.

This can be illustrated in **Figure 8**, where we see that the value obtained by the cumulative weighted representation of the spikes in the complex plane reaches the unit circle only when the right input is presented. This can be interpreted in terms of spikes by saying that for any input spike train that does not correspond to the right input the complex representation of the spikes do not have their phases aligned, and therefore they do not arrive to the perceiving neuron at the same time.

4.4. Temporal Difference Encoders

As the network described in section 3.2.4 is static, we simply presented the 320 sets of input spikes to the network as shown

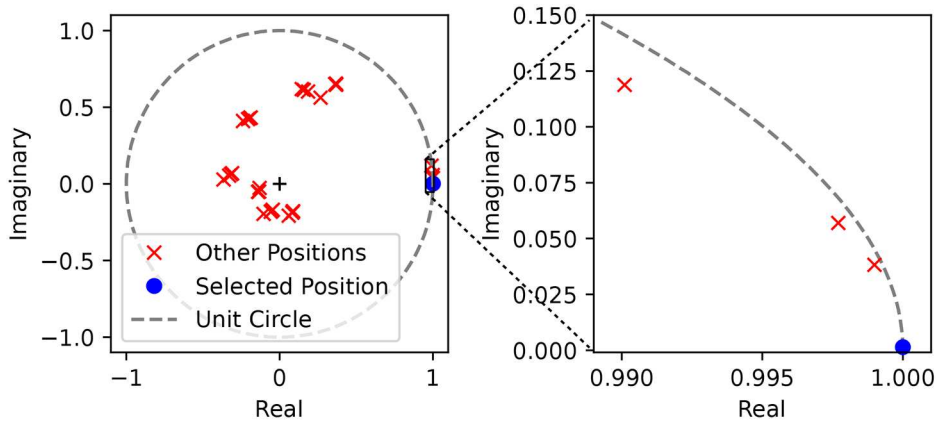


FIGURE 8 | Complex representation of the spikes weighted by their corresponding complex weights. We show the position of the weighted sum of the spikes in the complex plane given by $\hat{y}_p = \sum_k^{N_s} \hat{w}_p(k)s(k)$ for a single position. We used the parameters of a single position detector (angle 135° and distance 400 mm), and then tested the values of $\hat{y}_p$ with spike trains generated from a tap in the preferred position (blue) and in every other position (red). On the left we show the full unit circle and we observe that 32 different positions are clustered together in 8 clusters, corresponding to the 8 different angles. On the right plot we zoom around the solution and verify that the absolute value of $\hat{y}_p$ for other positions within the same cluster—meaning for the same angle but different distances—is indeed lower.

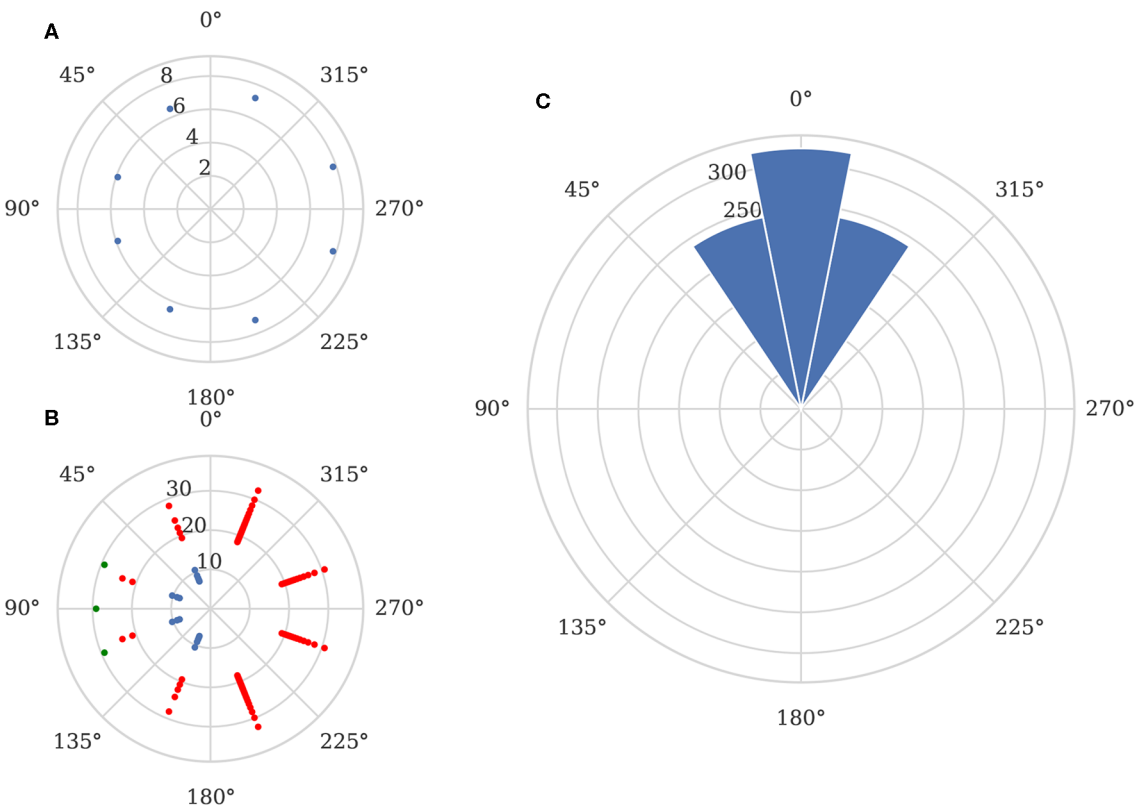


FIGURE 9 | Elementary motion detector network output. **(A)** Example input spikes from sensor when stimuli is located 800 mm away from the sensor at a 90° angle. **(B)** Corresponding spikes from TDE layer (blue), inverse direction layer (red) and direction output (green). **(C)** Direction detection performance across all recorded data.

in **Figure 9A**. Because this approach is only capable of finding the *angle* to the target, we can simply treat each output spikes from a "direction" neurons (green spikes in **Figure 9B**) as a "vote" for the target being at the direction neuron's corresponding angle. We can then subtract the correct stimuli angle from the angle associated with each spike and plot the circular

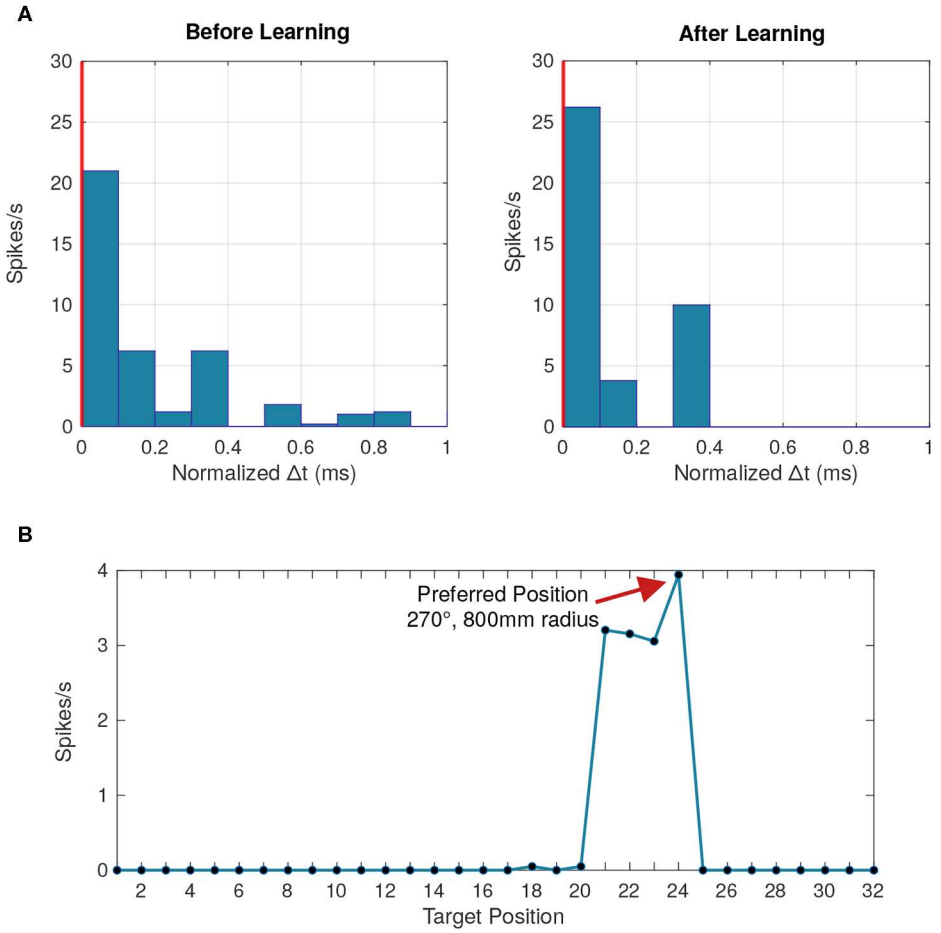


FIGURE 10 | Behavior of a postsynaptic neuron with synaptic delay plasticity. **(A)** Peristimulus time histogram averaged across 50 data points at the beginning and toward the end of the simulation. The vertical red line represents the postsynaptic neuron's firing time, chosen as a reference from which the time difference is calculated. After learning, more presynaptic neurons fire with a lower time difference compared to the postsynaptic firing time, due to the synchronization of input spikes. **(B)** Tuning curve of the postsynaptic neuron averaged across all the data points where the neuron fired, linking the firing rate to each of the 32 different stimuli positions. Positions 21–24 correspond to the distances 200, 400, 600, and 800 mm, respectively, at a 270° angle.

histogram shown in **Figure 9C**. This shows that the mean error of the classified directions is 0°, although a perfect one-hot encoding is not achieved resulting in a circular standard deviation of 17.5°.

Further investigation supports the existence of systematic biases in the sensor as, although the network is entirely symmetrical, the circular standard deviation is 0° for stimuli presented at angles of 0° and 180° whereas, for stimuli presented at all other angles, the circular standard deviation is 18.5°.

We used the GeNN library (Yavuz et al., 2016) to generate optimized CPU simulation code for this model. This simulation can be run 10× faster than real-time on a single ARM Cortex A57 core running at 2 GHz when using a 0.1 ms simulation time step, suggesting that this approach could be used for embedded online processing of spatio-temporal patterns.

4.5. Synaptic Delay Plasticity

The synaptic delay plasticity network consists of eight pre-synaptic neurons, sparsely connected in a random fashion to 50 LIF neurons. Sparsity is achieved by limiting the number of connections toward a LIF neuron $N = 4$. Synaptic delays are randomly initialized according to a normal distribution with a mean of $\mu = 0.5$ ms, and a standard deviation $\sigma = 0.3$ ms and a fixed weight equal to $\frac{w_0}{N}$ with $w_0 = 1$. The resting potential is set to $E_L = -70$ mV. The LIF neuron's decay constant is set to $\tau_m = 2$ ms, and the injected current I_{inj} is set to 180 nA to make sure that each presented spike train is capable of causing a LIF neuron to fire. The learning rate starts at $\eta = 1$ and decays by 10% after every 100 input spike trains to help the network converge toward a local minimum.

Each post-synaptic neuron that responds starts specializing to a particular pattern by synchronizing its input spikes through a change in synaptic delays following Equation (21). The winner-take-all mechanism ensures that no other post-synaptic neurons synchronize their input spikes. With each subsequent presentation of the pattern, the time differences between input spikes gradually converge toward zero (**Figure 10A**).

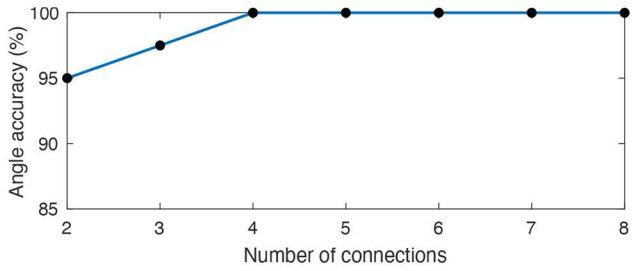


FIGURE 11 | Average angle accuracy for an increasingly sparse synaptic delay network. We connected each LIF neuron to a random subset of 8 pre-synaptic neurons representing the hardware sensors. We varied the size of the subset across 35 different trials in order to assess the smallest number of connections between pre-synaptic and LIF neurons capable of preserving an accurate temporal representation of all 8 angles. A sparse synaptic delay network with only 2 connections per LIF can already represent all angles with an average accuracy of 95% and we can achieve 100% accuracy with only 4 connections.

With a temporal resolution of 0.1 ms, all eight angles were successfully represented by at least one LIF neuron (**Figure 10B**). Previous implementations of learned delays relied on an all-to-all connectivity scheme (Hussain et al., 2012), but we obtained similar performance with fewer connections through a randomly connected sparse network with high redundancy (**Figure 11**). The learned temporal patterns for a network with only two connections per LIF neuron are enough to represent all angles with an accuracy of 95% and we can achieve 100% accuracy with only four connections per LIF neuron. An all-to-all network would work just as well, but, in addition to being more efficient in terms of hardware, a randomly connected and highly redundant sparse network increases robustness against systemic noise or a faulty sensor.

We also wanted to determine whether the synaptic delay network could differentiate between stimuli at different distances. As seen in **Figure 10B**, an individual neuron seems to respond more frequently to a particular position. We expect the membrane potential to be maximized for a particular distance which was not the case as the membrane potential was similar across all distances.

While the delay plasticity network managed to specialize neurons to all directions, due to the slow attenuation of the waves being measured, the temporal signatures across the measured distances are not significantly different. Spike synchronization seems to have a limited impact on the membrane potential beyond a certain level of synchronization. An inhibitory plasticity rule could be explored to further specialize post-synaptic neurons to increasingly precise temporal patterns.

4.6. Structural Plasticity

As a first step, we trained 4 neurons to learn 4 out of the 32 different spatio-temporal patterns (see **Figure 6B**). The network's free parameters, i.e., synaptic weights w, synaptic time constants τ and firing thresholds, are randomly initialized at the beginning of the training. Each spatio-temporal pattern, corresponding to a unique location, is presented 20 times to the network. In the beginning, the neurons sparsely capture incoming spatio-temporal patterns and therefore the thresholds slowly decrease. Each neuron starts to "lock on" to one particular pattern by decreasing the synaptic time constants of late spikes in the sequence and increasing the time constants of spikes early in the sequence following Equation (26) (for visualization of the time constant change see **Figure 6A**, blue dashed trace). The synaptic weights start to increase more for late spikes in the sequence, than for early spikes following Equation (24) (for visualization of the weight change see **Figure 6A**, black solid trace). Therefore the neuron begins to respond earlier to a given spatio-temporal pattern, while the mutual inhibition introduces competition on both the spike itself and the neuronal firing thresholds. After a given neuron's threshold—and thus its other free parameters—starts to stabilize, it reliably spikes in response to a particular pattern (see **Figure 12A**). After 15 stimuli presentations, the thresholds start to stabilize and each neuron locks onto 1 out of 4 different patterns (**Figure 12B**).

In a second step, we use 32 neurons and present all 32 patterns corresponding to different locations. Each pattern is presented 100 times to the network. The neurons fail to respond reliably to the different spatio-temporal patterns which is to due to the jitter present in the data, leading to too similar spatio-temporal patterns. While neurons are capable of learning these patterns, the jitter prevents the stabilization of the firing threshold and the synaptic weights and time constants keep on changing (see **Figure 12C**). This case of failure might also be due to the number of presentation needed by the network to learn a unique set of parameters which scales non-linearly with the number of patterns to learn and number neurons in the network (see Afshar et al., 2014 for statistical analysis of this relation).

Structural plasticity mechanisms or variants thereof within competitive EI networks has been demonstrated before to learn spatio-temporal patterns of activity in static (Gerstner et al., 1996; George, 2018) and time-varying (Masquelier, 2012; Roy and Basu, 2016; Roy et al., 2016) conditions. Unlike Roy et al. (2016), our approach does not need a reference time, but rather relies on relative latency encoding similar to Masquelier (2012). The proposed unsupervised structural plasticity algorithm is designed to operate on time-continuous, event-based sensory data in which there exist no start- or end-point to a pattern, nor one can rely on batch-training. In contrast to the feature extraction approach proposed by Afshar et al. (2019b)—which inspired this work—neurons in our network adapt their neuronal firing threshold solely based on locally available signals. We do so by utilizing the inhibitory interneuron population which indirectly signal the presence of captured spatio-temporal pattern by other neurons by hyperpolarizing the non-spiking excitatory neurons.

Understanding the computational properties and emergent network dynamics resulting from recurrent excitation and inhibition mediated balanced activity is beyond the scope of this paper, but will be subject of future investigations. A promising next step would be to learn the temporal relations of different spatio-temporal patterns by exploiting recurrent excitatory synapses with STDP, as described in Kappel et al. (2014) and Milde (2019).

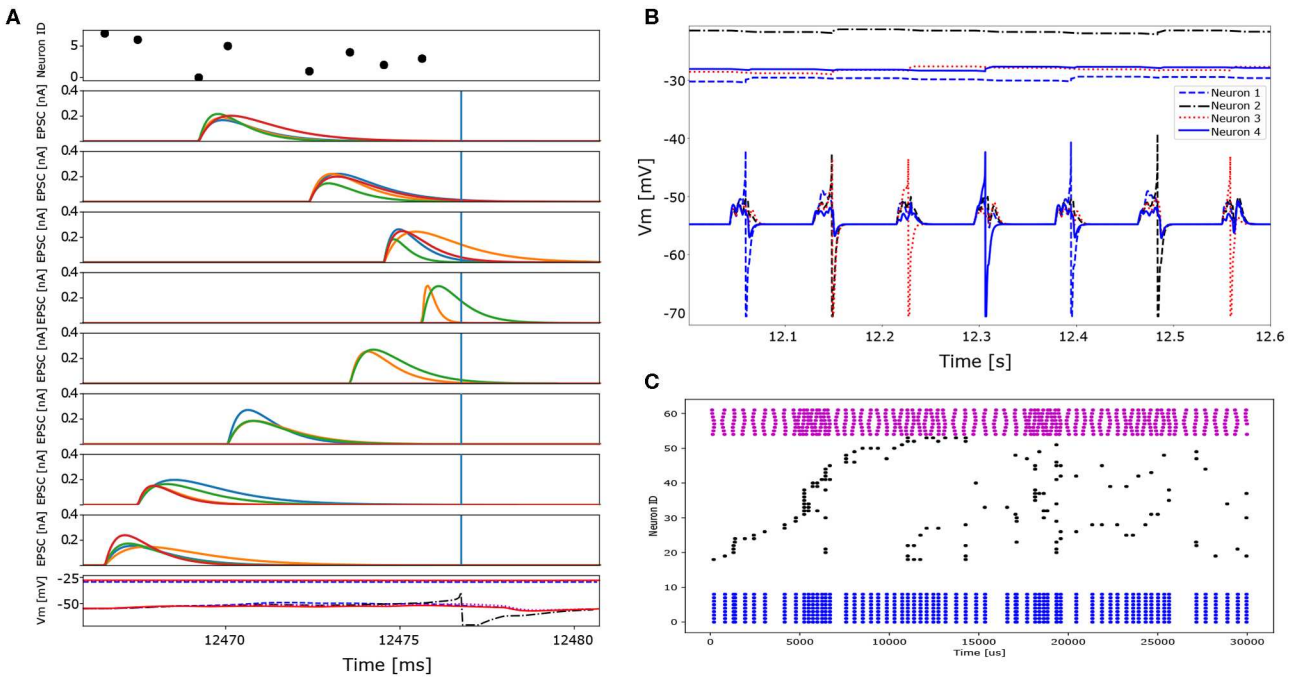


FIGURE 12 | Detailed excitatory post-synaptic current traces and spiking behavior of neurons in the context of two different tasks which vary in the number of patterns presented to the network and the network size. In the first task only four different spatio-temporal patterns (four locations) are presented to a network consisting of four excitatory and 2 inhibitory neurons. In the second task 32 different spatio-temporal patterns (32 locations, i.e., the entire data set) are presented to a network of 32 excitatory and 8 inhibitory neurons. **(A)** Detailed EPSC traces of the eight input synapses to the four excitatory neurons (1 color per post-synaptic neuron) to 1 out of 4 different spatio-temporal patterns (first task). The top plot represents the input pattern, whereas the bottom panel shows the membrane potential traces of the four excitatory neurons. Note that only one neuron generates an action potential and subsequently inhibits the others via the inhibitory interneurons. **(B)** Membrane potentials and corresponding neuronal firing thresholds of 4 neurons learning 4 different spatio-temporal patterns (first task). Each neuron learns to represent a single input pattern and consequently spikes reliably to only one out of the four patterns. Two pattern repetitions are shown. After each neuron locks onto one out of the four patterns the neuronal firing threshold stabilize. **(C)** Spike raster plot of the input and the network's activity. Blue dots represent the inhibitory neuron activity (bottom), black dots indicate excitatory neuron activity (middle) and pink dots represent the different input patterns (top). The network fails to converge and represent each location using a single neuron. The reason for this might be due to the too short training time given the amount of different patterns or due to the too high similarity in the input patterns for the same angle but different distances. The sampling frequency of the ADC is too slow to provide the needed temporal precision to resolve the distance if the stimulus onset is not known.

4.7. Comparison and Extensions

In the preceding sections of this work, we proposed several approaches for tackling the problem of spatio-temporal pattern classification, in the context of touch localization based on the precise timing of input events. **Table 1** shows an overview of the presented results. It is worth noting that, as a community, we lack clear metrics for assessing the performance of spiking networks. While a simple accuracy metric can be used, it fails to consider factors such as power consumption and suitability for real-time simulation as well as not reflecting constraints present in both biological SNNs and neuromorphic hardware (Nowotny, 2014). Further effort will have to be done by the community to overcome this problem and provide datasets and metrics which *do* consider these factors.

TABLE 1 | Comparison for the hereby proposed methods.

Method	Angle accuracy (%)	Distance accuracy (%)
Analytic solution	99.7	73.4
Temporal coincidence	100	37.5
Complex weights and delays	100	100*
Temporal difference encoders	100	N.A.
Synaptic delay plasticity	100	N.A.
Structural plasticity	100	N.A.

If all the approaches are able to distinguish between the arrival angles, the distance is still an open issue here. The analytic solution is able to (almost) correctly extract the distance, under the assumption of a known geometry. Some thoughts about this problem are detailed in the Discussion section (*). The 100% accuracy for the distance in the Complex weight and delay method is under the assumption of using an extra linear classifier to process the output data.

5. DISCUSSION

In this paper we demonstrated, through a simple task, different approaches for tackling spatio-temporal pattern classification with SNNs. The problem of separating spatio-temporal patterns into prototypical features or discrete classes by learning, clustering or any other form of transformation resides at the core of both event-driven computing and event-based neuromorphic processing (Chicca et al., 2014;

Indiveri and Sandamirskaya, 2019). This work is not intended to demonstrate high-precision computing, but rather to open new perspectives on learning these spatio-temporal patterns and on performing event-based tactile sensory processing. The presented algorithms were chosen to provide a qualitative overview given certain constraints of available information on how to extract task-relevant information from the timing of incoming events. Despite their different complexities, all of these approaches extract the required information solely from the precise timing of the incoming events. While a major drawback of all of these approaches is the need for temporal precision on the sensory side, our experiments expand on how information can be extracted from the timing of an incoming event in neurally-inspired processing paradigms.

As discussed in section 2.1, in sand, compressional waves attenuates rapidly with distance ($G(d) = \frac{1}{d}$) meaning that the gradient of attenuation across the scorpion's outspread legs can be used to estimate distance to the stimuli. However, on the surface used in this work, attenuation is lower. Therefore, it seems unlikely that our system would sense a difference in amplitude between the sensors. Nevertheless, *mean* amplitude across the sensors could be used to estimate distance, although it would be unable to disambiguate between a distant stimuli with a large amplitude and a nearby stimuli with a smaller amplitude.

Although it is true that the case of multiple sources is not addressed in this work, we would highlight that the precision of the sensors is of a few microseconds, meaning that the vibration should have to be generated at two sources exactly at the same time, which is unlikely. We can, however, speculate that in the case of multiple sources, the methods with excitatory synapses only—such as complex weights—should promote the activation of the neurons corresponding to the two sources, while those with lateral inhibition—such as synaptic delay plasticity—would resolve a conflict one way or the other, giving one active source at a time.

Due to the nature of the stimuli, all of the approaches presented in this paper require simulations with high temporal resolution. While small models requiring high temporal resolution—such as the TDE-based approach discussed in section 4.4—can be simulated in real-time using simple CPU-based simulations, for larger models many current approaches are not capable of providing high temporal resolution and real-time simulation speed.

The majority of digital neuromorphic systems (Furber et al., 2014; Merolla et al., 2014; Davies et al., 2018; Frenkel et al., 2018) use a time-driven approach for simulating neurons with simulation time steps of around 1 ms. While some systems can operate at a higher temporal resolution, this typically requires increasing the clock speed, leading to increased power consumption. This programmability of the SpiNNaker platform (Furber et al., 2014) means that, although this platform was *designed* to operate on a 1 ms simulation time step, it has been recently demonstrated that a 0.1 ms time step is achievable through careful programming (Knight and Furber, 2016; Rhodes et al., 2020). Furthermore, when even higher temporal resolution is required, truly event-driven models capable of learning temporal patterns with sub-millisecond precision have also been demonstrated on SpiNNaker (Lagorce et al., 2015). On the other hand, in terms of efficient processing with high temporal precision, mixed-signal analog/digital neuromorphic systems such as ROLLS (Qiao et al., 2015) or DYNAP-SE (Moradi et al., 2017) have a distinct advantage, as their neuronal dynamics arise from the physical characteristics of their analog circuits so time *represents itself*. As such, analog systems have been successfully used for a variety of complex spatio-temporal signal processing tasks, even exhibiting cognitive abilities Neftci et al. (2013), or including spike-based plasticity mechanism applied to learning auditory features from a silicon cochlea (Sheik et al., 2012), sequence learning (Kreiser et al., 2018a; Milde, 2019) and simultaneous localization and mapping using a silicon retina (Kreiser et al., 2018b).

We anticipate that this work will be extended to qualitatively and quantitatively assess solutions to the problem of spatio-temporal pattern learning, which exploit the fact that time represent itself in neural computation, and thus uses the precise timing of events to learn in a purely event-driven manner. The Neuromorphic Engineering community is facing and needs to overcome this canonical problem, to establish itself as a viable alternative to conventional clock-based sensing and processing systems.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

GH and MM developed the neuromorphic tactile sensor. GH developed the analytical solution to the localization problem. PA developed the temporal coincidence detection and complex weights and delays based algorithms. JK developed the Temporal Difference Encoder based algorithm. OO developed the synaptic delay plasticity algorithm. MM developed the structural plasticity algorithm. GH, MM, PA, OO, JK, and AS wrote the paper. AS, RB, and GI contributed to the paper writing process and supervised the research.

FUNDING

This work was partially supported by the EPSRC (Brains on Board project, grant number EP/P006094/1), the Bundesministerium für Bildung und Forschung and the Max Planck Society through the Max Planck School of Cognition, the NEUROTECH CSA project through the Neurotech fellowship, the European Union's Horizon 2020 research and innovation programme under grant agreement no 732642 and the Swiss National Science Foundation (Sinergia project #CRSII5-18O316).

ACKNOWLEDGMENTS

We authors would like to thank the organizers of the CapoCaccia Neuromorphic Engineering Workshop 2019, where intense discussion and work on the approaches proposed in this paper began. Additionally, we would like to thank Wolfgang Stuerzl for kindly providing us with a library of beautiful photographs of sand scorpions, Martin Reichert for originally taking them.

REFERENCES

Afshar, S., George, L., Tapson, J., van Schaik, A., and Hamilton, T. J. (2014). Racing to learn: statistical inference and learning in a single spiking neuron with adaptive kernels. *Front. Neurosci.* 8:377. doi: 10.3389/fnins.2014.00377

Afshar, S., Hamilton, T. J., Tapson, J., van Schaik, A., and Cohen, G. (2019a). Investigation of event-based surfaces for high-speed detection, unsupervised feature extraction, and object recognition. *Front. Neurosci.* 12:1047. doi: 10.3389/fnins.2018.01047

Afshar, S., Xu, Y., Tapson, J., van Schaik, A., and Cohen, G. (2019b). Event-based feature extraction using adaptive selection thresholds. *arXiv preprint arXiv:1907.07853*. doi: 10.3390/s20061600

Astrom, K. J., and Bernhardsson, B. M. (2002). "Comparison of Riemann and Lebesgue sampling for first order stochastic systems," in *Proceedings of the 41st IEEE Conference on Decision and Control*, Vol. 2 (Las Vegas, NV: IEEE), 2011–2016. doi: 10.1109/CDC.2002.1184824

Baldi, P., and Atiya, A. F. (1994). How delays affect neural dynamics and learning. *IEEE Trans. Neural Netw.* 5, 612–621. doi: 10.1109/72.298231

Baudot, P., Levy, M., Marre, O., Monier, C., Pananceau, M., and Frégnac, Y. (2013). Animation of natural scene by virtual eye-movements evokes high precision and low noise in v1 neurons; handwritten. *Front. Neural Circ.* 7:206. doi: 10.3389/fncir.2013.00206

Bekkers, J. M. (2011). Changes in dendritic axial resistance alter synaptic integration in cerebellar Purkinje cells. *Biophys. J.* 100, 1198–1206. doi: 10.1016/j.bpj.2011.01.042

Brownell, P., and Farley, R. D. (1979a). Detection of vibrations in sand by tarsal sense organs of the nocturnal scorpion, Paruroctonus mesaensis. *J. Comp. Physiol. A* 131, 23–30. doi: 10.1007/BF00613080

Brownell, P., and Farley, R. D. (1979b). Orientation to vibrations in sand by the nocturnal scorpion Paruroctonus mesaensis: mechanism of target localization. *J. Comp. Physiol. A* 131, 31–38. doi: 10.1007/BF00613081

Brownell, P. H. (1977). Compressional and surface waves in sand: Used by desert scorpions to locate prey. *Science* 197, 479–482. doi: 10.1126/science.197.4302.479

Buonomano, D. V., and Maass, W. (2009). State-dependent computations: spatiotemporal processing in cortical networks. *Nat. Rev. Neurosci.* 10, 113–125. doi: 10.1038/nrn2558

Buonomano, D. V., and Merzenich, M. M. (1995). Temporal information transformed into a spatial code by a neural network with realistic properties. *Science* 267, 1028–1030. doi: 10.1126/science.7863330

Carr, C. E., and Konishi, M. (1990). A circuit for detection of interaural time differences in the brain-stem of the barn owl. *J. Neurosci.* 10, 3227–3246. doi: 10.1523/JNEUROSCI.10-10-03227.1990

Chauhan, T., Masquelier, T., Montlibert, A., and Cottereau, B. R. (2018). Emergence of binocular disparity selectivity through Hebbian learning. *J. Neurosci.* 38, 9563–9578. doi: 10.1523/JNEUROSCI.1259-18.2018

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

Cohen, G. K., Orchard, G., Leng, S.-H., Tapson, J., Benosman, R. B., and van Schaik, A. (2016). Skimming digits: neuromorphic classification of spike-encoded images. *Front. Neurosci.* 10:184. doi: 10.3389/fnins.2016.00184

Corradi, F., and Indiveri, G. (2015). A neuromorphic event-based neural recording system for smart brain-machine-interfaces. *IEEE Trans. Biomed. Circ. Syst.* 9, 699–709. doi: 10.1109/TBCAS.2015.2479256

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* 30, 82–99. doi: 10.1109/MM.2018.112130359

Dayan, P., and Abbott, L. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, MA: MIT Press.

Dean, A. (1981). The variability of discharge of simple cells in the cat striate cortex. *Exp. Brain Res.* 44, 437–440. doi: 10.1007/BF00238837

Deneve, S., and Machens, C. K. (2016). Efficient codes and balanced networks. *Nat. Neurosci.* 19:375. doi: 10.1038/nn.4243

Eurich, C. W., Pawelzik, K., Ernst, U., Cowan, J. D., and Milton, J. G. (1999). Dynamics of self-organized delay adaptation. *Phys. Rev. Lett.* 82, 1594–1597. doi: 10.1103/PhysRevLett.82.1594

Eurich, C. W., Pawelzik, K., Ernst, U., Thiel, A., Cowan, J. D., and Milton, J. G. (2000). Delay adaptation in the nervous system. *Neurocomputing* 32, 741–748. doi: 10.1016/S0925-2312(00)00239-3

Fields, R. D. (2015). A new mechanism of nervous system plasticity: activity-dependent myelination. *Nat. Rev. Neurosci.* 16, 756–767. doi: 10.1038/nrn4023

Frady, E. P., and Sommer, F. T. (2019). Robust computation with rhythmic spike patterns. *Proc. Natl. Acad. Sci. U.S.A.* 116, 18050–18059. doi: 10.1073/pnas.1902653116

Frenkel, C., Lefebvre, M., Legat, J.-D., and Bol, D. (2018). A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm CMOS. *IEEE Trans. Biomed. Circ. Syst.* 13, 145–158. doi: 10.1109/TBCAS.2018.2880425

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

George, R. M. (2018). *Structural plasticity in neuromorphic systems* (Ph.D. thesis). University of Zurich, Zurich, Switzerland. doi: 10.1109/BIOCAS.2017.8325074

Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature* 383:76. doi: 10.1038/383076a0

Goel, A., and Buonomano, D. V. (2016). Temporal interval learning in cortical cultures is encoded in intrinsic network dynamics. *Neuron* 91, 320–327. doi: 10.1016/j.neuron.2016.05.042

Goodhill, G. J., and Barrow, H. G. (1994). The role of weight normalization in competitive learning. *Neural Comput.* 6, 255–269. doi: 10.1162/neco.1994.6.2.255

Gütig, R., and Sompolinsky, H. (2006). The tempotron: a neuron that learns spike timing-based decisions. *Nat. Neurosci.* 9, 420–428. doi: 10.1038/nn1643

Hipp, J., Arabzadeh, E., Zorzin, E., Conradt, J., Kayser, C., Diamond, M. E., et al. (2006). Texture signals in whisker vibrations. *J. Neurophysiol.* 95, 1792–1799. doi: 10.1152/jn.01104.2005

Hirose, A. (1992). Continuous complex-valued back-propagation learning. *Electron. Lett.* 28, 1854–1855. doi: 10.1049/el:19921186

Hu, J.-S., and Yang, C.-H. (2010). Estimation of sound source number and directions under a multisource reverberant environment. *EURASIP J. Adv. Signal Process.* 2010:870756. doi: 10.1155/2010/870756

Hubel, D., and Wiesel, T. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* 160, 106–54. doi: 10.1113/jphysiol.1962.sp006837

Hussain, S., Basu, A., Wang, M., and Hamilton, T. J. (2012). "Deltron: neuromorphic architectures for delay based learning," in *2012 IEEE Asia Pacific Conference on Circuits and Systems* (Kaohsiung: IEEE), 304–307. doi: 10.1109/APCCAS.2012.6419032

Indiveri, G., and Sandamirskaya, Y. (2019). The importance of space and time for signal processing in neuromorphic agents. *IEEE Signal Process. Mag.* 36, 16–28. doi: 10.1109/MSP.2019.2928376

Izhikevich, E. (2006). Polychronization: computation with spikes. *Neural Comput.* 18, 245–282. doi: 10.1162/089976606775093882

Kappel, D., Nessler, B., and Maass, W. (2014). STDP installs in winner-take-all circuits an online approximation to hidden markov model learning. *PLoS Comput. Biol.* 10:e1003511. doi: 10.1371/journal.pcbi.1003511

Knight, J., and Furber, S. (2016). Synapse-centric mapping of cortical models to the SpiNNaker neuromorphic architecture. *Front. Neurosci.* 10:420. doi: 10.3389/fnins.2016.00420

Knoblauch, A., Körner, E., Körner, U., and Sommer, F. T. (2014). Structural synaptic plasticity has high memory capacity and can explain graded amnesia, catastrophic forgetting, and the spacing effect. *PLoS ONE* 9:e96485. doi: 10.1371/journal.pone.0096485

Koudelka, S., Voas, M. G., Almeida, R. G., Baraban, M., Soetaert, J., Meyer, M. P., et al. (2016). Individual neuronal subtypes exhibit diversity in CNS myelination mediated by synaptic vesicle release. *Curr. Biol.* 26, 1447–1455. doi: 10.1016/j.cub.2016.03.070

Krammer, J., and Koch, C. (1997). Pulse-based analog VLSI velocity sensors. *IEEE Trans. Circ. Syst. II Anal. Digital Signal Process*. 44, 86–101. doi: 10.1109/82.554431

Kreiser, R., Aathmani, D., Qiao, N., Indiveri, G., and Sandamirskaya, Y. (2018a). Organizing sequential memory in a neuromorphic device using dynamic neural fields. *Front. Neurosci.* 12:717. doi: 10.3389/fnins.2018.00717

Kreiser, R., Renner, A., Sandamirskaya, Y., and Pienroj, P. (2018b). "Pose estimation and map formation with spiking neural networks: toward neuromorphic slam," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Madrid: IEEE), 2159–2166. doi: 10.1109/IROS.2018.8594228

Lagorce, X., Orchard, G., Galluppi, F., Shi, B. E., and Benosman, R. B. (2017). Hots: a hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Lagorce, X., Stromatias, E., Galluppi, F., Plana, L. A., Liu, S.-C., Furber, S. B., et al. (2015). Breaking the millisecond barrier on SpiNNaker: implementing asynchronous event-based plastic models with microsecond resolution. *Front. Neurosci.* 9:206. doi: 10.3389/fnins.2015.00206

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, W. W., Kukreja, S. L., and Thakor, N. V. (2017). Discrimination of dynamic tactile contact by temporally precise event sensing in spiking neuromorphic networks. *Front. Neurosci.* 11:5. doi: 10.3389/fnins.2017.00005

Maass, W. (2001). On the relevance of time in neural computation and learning. *Theor. Comput. Sci.* 261, 157–178. doi: 10.1016/S0304-3975(00)00137-7

Mahajan, A., and Walworth, M. (2001). 3D position sensing using the differences in the time-of-flights from a wave source to various receivers. *IEEE Trans. Robot. Autom.* 17, 91–94. doi: 10.1109/70.917087

Mainen, Z., and Sejnowski, T. (1995). Reliability of spike timing in neocortical neurons. *Science* 268, 1503–1506. doi: 10.1126/science.7770778

Markram, H., Lübke, 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

Masquelier, T. (2012). Relative spike time coding and STDP-based orientation selectivity in the early visual system in natural continuous and saccadic vision: a computational model. *J. Comput. Neurosci.* 32, 425–441. doi: 10.1007/s10827-011-0361-9

Matsubara, T. (2017). Spike timing-dependent conduction delay learning model classifying spatio-temporal spike patterns. *Front. Comput. Neurosci.* 11:104. doi: 10.3389/fncom.2017.00104

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

Milde, M. B. (2019). *Spike-based computational primitives for vision-based scene understanding* (Ph.D. thesis), University of Zurich.

Milde, M. B., Bertrand, O. J. N., Ramachandran, H., Egelhaaf, M., and Chicca, E. (2018). Spiking elementary motion detector in neuromorphic systems. *Neural Comput.* 30, 2384–2417. doi: 10.1162/neco_a_01112

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. Circ. Syst.* 12, 106–122. doi: 10.1109/TBCAS.2017.2759700

Morrison, A., Diesmann, M., and Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biol. Cybernet.* 98, 459–478. doi: 10.1007/s00422-008-0233-1

Neftci, E., Binas, J., Rutishauser, U., Chicca, E., Indiveri, G., and Douglas, R. (2013). Synthesizing cognition in neuromorphic electronic systems. *Proc. Natl. Acad. Sci. U.S.A.* 110, E3468–E3476. doi: 10.1073/pnas.1212083110

Nowotny, T. (2014). Two challenges of correct validation in pattern recognition. *Front. Robot. AI* 1:5. doi: 10.3389/frobt.2014.00005

O'Keefe, J., and Recce, M. L. (1993). Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus* 3, 317–330. doi: 10.1002/hipo.450030307

Paugam-Moisy, H., Martinez, R., and Bengio, S. (2008). Delay learning and polychronization for reservoir computing. *Neurocomputing* 71, 1143–1158. doi: 10.1016/j.neucom.2007.12.027

Qiao, N., Bartolozzi, C., and Indiveri, G. (2016). "Automatic gain control of ultra-low leakage synaptic scaling homeostatic plasticity circuits," in *Biomedical Circuits and Systems Conference (BioCAS)* (Shanghai: IEEE), 156–159. doi: 10.1109/BioCAS.2016.7833755

Qiao, N., Bartolozzi, C., and Indiveri, G. (2017). An ultralow leakage synaptic scaling homeostatic plasticity circuit with configurable time scales up to 100 ks. *IEEE Trans. Biomed. Circ. Syst.* 11, 1271–1277. doi: 10.1109/TBCAS.2017.2754383

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.* 9:141. doi: 10.3389/fnins.2015.00141

Rall, W. (1967). Distinguishing theoretical synaptic potentials computed for different soma-dendritic distributions of synaptic input. *J. Neurophysiol.* 30, 1138–1168. doi: 10.1152/jn.1967.30.5.1138

Reichert, D. P., and Serre, T. (2013). Neuronal synchrony in complex-valued deep networks. *arXiv [Preprint] arXiv:*1312.6115.

Rhodes, O., Peres, L., Rowley, A. G. D., Gait, A., Plana, L. A., Brenninkmeijer, C., et al. (2020). Real-time cortical simulation on neuromorphic hardware. *Philos. Trans. R. Soc. A* 378:20190160. doi: 10.1098/rsta.2019.0160

Roy, S., and Basu, A. (2016). An online unsupervised structural plasticity algorithm for spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 28, 900–910. doi: 10.1109/TNNLS.2016.2582517

Roy, S., San, P. P., Hussain, S., Wei, L. W., and Basu, A. (2016). Learning spike time codes through morphological learning with binary synapses. *IEEE Trans. Neural Netw. Learn. Syst.* 27, 1572–1577. doi: 10.1109/TNNLS.2015.2447011

Shadlen, M. N., and Newsome, W. T. (1998). The variable discharge of cortical neurons: implications for connectivity, computation, and information coding. *J. Neurosci.* 18, 3870–3896. doi: 10.1523/JNEUROSCI.18-10-03870.1998

Sheik, S., Coath, M., Indiveri, G., Denham, S. L., Wennekers, T., and Chicca, E. (2012). Emergent auditory feature tuning in a real-time neuromorphic VLSI system. *Front. Neurosci.* 6:17. doi: 10.3389/fnins.2012.00017

Shrestha, S. B., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems* (Montreal, QC), 1412–1421.

Softky, W. R., and Koch, C. (1993). The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs. *J. Neurosci.* 13, 334–350. doi: 10.1523/JNEUROSCI.13-01-00334.1993

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3, 919–26. doi: 10.1038/78829

State, L. (2019). *Training delays in spiking neural networks* (Master's thesis). Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany. Available from the Preprint repository of the Max Planck Institute for Mathematics in the Sciences, Preprint 96/2019.

Stürzl, W., Kempter, R., and Van Hemmen, J. L. (2000). Theory of arachnid prey localization. *Phys. Rev. Lett.* 84, 5668–5671. doi: 10.1103/PhysRevLett.84.5668

Swadlow, H. A., and Waxman, S. G. (2012). Axonal conduction delays. *Scholarpedia* 7:1451. doi: 10.4249/scholarpedia.1451

Taherkhani, A., Belatreche, A., Li, Y., and Maguire, L. P. (2015). Dl-resume: a delay learning-based remote supervised method for spiking neurons. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 3137–3149. doi: 10.1109/TNNLS.2015.2404938

Takagi, H. (2000). Roles of ion channels in EPSP integration at neuronal dendrites. *Neurosci. Res.* 37, 167–171. doi: 10.1016/S0168-0102(00)00120-6

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

Thorpe, S., Delorme, A., Rullen, R. V., et al. (2001). Spike-based strategies for rapid processing. *Neural Netw.* 14, 715–725. doi: 10.1016/S0893-6080(01)00083-1

Turrigiano, G., and Nelson, S. (2004). Homeostatic plasticity in the developing nervous system. *Nat. Rev. Neurosci.* 5, 97–107. doi: 10.1038/nrn1327

Wang, R., Cohen, G., Stiefel, K., Hamilton, T., Tapson, J., and van Schaik, A. (2013). An FPGA implementation of a polychronous spiking neural network with delay adaptation. *Front. Neurosci*. 7:14. doi: 10.3389/fnins.2013.00014

Wang, R. M., Hamilton, T. J., Tapson, J. C., and van Schaik, A. (2015). A neuromorphic implementation of multiple spike-timing synaptic plasticity rules for large-scale neural networks. *Front. Neurosci*. 9:180. doi: 10.3389/fnins.2015.00180

Wehr, M., and Zador, A. M. (2003). Balanced inhibition underlies tuning and sharpens spike timing in auditory cortex. *Nature* 426:442. doi: 10.1038/nature02116

Wolfe, J., Hill, D. N., Pahlavan, S., Drew, P. J., Kleinfeld, D., and Feldman, D. E. (2008). Texture coding in the rat whisker system: slip-stick versus differential resonance. *PLoS Biology* 6:e215. doi: 10.1371/journal.pbio.0060215

Yavuz, E., Turner, J., and Nowotny, T. (2016). GeNN: a code generation framework for accelerated brain simulations. *Sci. Rep*. 6:18854. doi: 10.1038/sr5ep18854

Zito, K., Parnas, D., Fetter, R. D., Isacoff, E. Y., and Goodman, C. S. (1999). Watching a synapse grow: noninvasive confocal imaging of synapticgrowth in drosophila. *Neuron* 22, 719–729. doi: 10.1016/S0896-6273(00)80731-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 Neuroscience

ORIGINAL RESEARCH
published: 23 June 2020
doi: 10.3389/fnins.2020.00551

An On-chip Spiking Neural Network for Estimation of the Head Pose of the iCub Robot

Raphaela Kreiser[1†], *Alpha Renner*[1†], *Vanessa R. C. Leite*[1†], *Baris Serhan*[2], *Chiara Bartolozzi*[3], *Arren Glover*[3*] *and Yulia Sandamirskaya*[1*]

[1] Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich, Switzerland, [2] Lincoln Centre for Autonomous Systems, University of Lincoln, Lincoln, United Kingdom, [3] Istituto Italiano di Tecnologia, Genoa, Italy

In this work, we present a neuromorphic architecture for head pose estimation and scene representation for the humanoid iCub robot. The spiking neuronal network is fully realized in Intel's neuromorphic research chip, Loihi, and precisely integrates the issued motor commands to estimate the iCub's head pose in a neuronal path-integration process. The neuromorphic vision system of the iCub is used to correct for drift in the pose estimation. Positions of objects in front of the robot are memorized using on-chip synaptic plasticity. We present real-time robotic experiments using 2 degrees of freedom (DoF) of the robot's head and show precise path integration, visual reset, and object position learning on-chip. We discuss the requirements for integrating the robotic system and neuromorphic hardware with current technologies.

Keywords: pose estimation, event-based vision, neuromorphic SLAM, on-chip learning, scene memory, iCub robot, visual reset, spiking neural networks

OPEN ACCESS

Edited by:
Alejandro Linares-Barranco, Universidad de Sevilla, Spain

Reviewed by:
Federico Corradi, Imec, Netherlands
Jim Harkin, Ulster University, United Kingdom

***Correspondence:**
Arren Glover
arren.glover@iit.it
Yulia Sandamirskaya
yulia.sandamirskaya@intel.com

†These authors have contributed equally to this work and share first authorship

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *24 December 2019*
Accepted: *04 May 2020*
Published: *23 June 2020*

Citation:
Kreiser R, Renner A, Leite VRC, Serhan B, Bartolozzi C, Glover A and Sandamirskaya Y (2020) An On-chip Spiking Neural Network for Estimation of the Head Pose of the iCub Robot. Front. Neurosci. 14:551. doi: 10.3389/fnins.2020.00551

1. INTRODUCTION

Neuromorphic hardware implements the non-Von Neumann brain-inspired computing architecture based on known properties of biological neural networks. This computing architecture features event-based asynchronous processing and fine-grained parallelism of a network of spiking neurons (Indiveri et al., 2009; Schemmel et al., 2010; Furber et al., 2012; Merolla et al., 2014; Galluppi et al., 2015; Qiao et al., 2015; Davies et al., 2018; Moradi et al., 2018). Neuromorphic hardware not only supports parallel processing; it also enables feedback loops, recurrence, and online adaptation—the key properties of biological brains that lead to flexible and robust behavior. Biological neural systems evolved to solve tasks that are highly relevant to robotics: perception, movement control, action planning, or decision making under uncertainty. Thus, robotics is a promising application domain for neuromorphic hardware (Krichmar and Wagatsuma, 2011). Autonomous robots require that computing be performed with low latency and low power consumption, and these are the key characteristics of neuromorphic devices. In this work, we contribute to the emerging field of neuromorphic robotics by presenting a number of design patterns—spiking neural network models—to solve one of the key robotic tasks, state estimation.

To be used efficiently, neuromorphic hardware requires a radical rethinking of the computing paradigm. In neuromorphic hardware, we cannot run functions, create conditional loops, or have if-then-else statements in the same way as in conventional software. To use the brain-inspired computing substrate—neurons and synapses—efficiently, we need to abandon the notion of addition and multiplication as elementary computing operations. Even the mere representation

of values as binary bit-strings becomes obsolete in a neuronal computing framework. Instead, neuromorphic systems represent the measured physical variables and perform computation using events (spikes) spreading in a neuronal network, as brains do. Thus, in our work, we aim to develop the neuronal computing elements that are required to solve different tasks, seeking to derive principles and structures that can be reused in different domains. Moreover, we show how an interface can be established between the sensors and motors of a robot and neuromorphic representations.

Neuronal network-based algorithms currently deliver the most impressive results in computer vision and machine learning and are increasingly being deployed in robotics (Chen et al., 2015; Mnih et al., 2015). Training spiking neuronal networks (SNNs) using methods developed for deep learning (i.e., error backpropagation) is challenging and currently leads to reduced performance compared to conventional, full-precision DNNs (Shrestha and Orchard, 2018; Neftci et al., 2019). On the other hand, the neuromorphic hardware supports online learning, i.e., the adaptation of synaptic weights after deployment of the system. When designing our SNN models, we do not rely on tabula-rasa data-driven learning. Instead, we leverage the knowledge of neuronal circuits that solve similar tasks in animals, reserving learning only to parts that depend on the environment with which the robot interacts. This learning can happen in a "shallow" network.

Findings in neuroscience have inspired a number of neuronal architectures for addressing the problem of simultaneous localization and mapping (SLAM) (Arleo and Gerstner, 2000; Cuperlier et al., 2007; Barrera and Weitzenfeld, 2008; Weikersdorfer et al., 2013; Milford and Schulz, 2014; Jauffret et al., 2015). SLAM is one of the core problems in mobile robotics (Stachniss et al., 2016) but can be generalized to any robotic system that requires state estimation of the robot relative to its environment. In this work, we present a spiking neural network (SNN) implemented on Intel's neuromorphic research chip, Loihi, for pose estimation of the robot's head. The pose is estimated in the SNN based on the "efferent copy" of the motor commands. The estimate is corrected by a visual cue when the robot sees an object multiple times during exploration of an environment. The initial pose, under which the object was seen the first time, is learned in plastic synapses on chip and is used for the visual reset. Estimating the pose by integrating the motor commands is referred to as dead reckoning in robotics and as path integration in biology.

In robotics, the head-pose estimation amounts to the camera pose estimation problem (e.g., Scaramuzza and Fraundorfer, 2011). The camera pose is estimated using on-board sensors to measure an incremental change in pose, e.g., the visual system itself, a built-in inertial measurement unit (IMU), laser range finder, time of flight camera (Engelhard et al., 2011) or sonar (Thrun et al., 2007). Fusion of information from multiple sensors is performed to provide a more robust estimate of the pose change. Since integration of movement is prone to error accumulation, such systems need frequent recalibration. The global positioning system (GPS) or external cameras, e.g., Vicon system, help to avoid this problem, but in many cases measuring the ground-truth pose directly is not possible, and the problem becomes one of simultaneous localization and mapping (SLAM). The reference relative to which the pose is measured is itself estimated concurrently with the estimate of the pose (Stachniss et al., 2016).

Animals can also navigate in large environments by combining a set of "on-board" sensors, i.e., the vestibular and vision system (Burak and Fiete, 2009; Seelig and Jayaraman, 2015; Green et al., 2017; Fisher et al., 2019). They combine motion commands and internal sensing in their neuronal systems to provide a motion estimate. Even simple animals, such as insects, show complex navigation behaviors. A brain region called the central complex (CX) appears to be their navigation center (Pfeiffer and Homberg, 2014; Turner-Evans and Jayaraman, 2016; Heinze, 2017). Visual landmarks (Seelig and Jayaraman, 2013), rotational optic flow, and non-visual angular velocity cues (Green et al., 2017; Turner-Evans et al., 2017) were shown to mediate direction coding in CX neurons, suggesting that allothetic and idiothetic cues are continuously integrated to generate a robust representation of body orientation (Honkanen et al., 2019). The orientation appears to be encoded in the activity bump of neurons arranged in a ring that corresponds to the 360° of possible directions. These computational principles were uncovered in brains with a size of <100K neurons and were shown to fit small-scale neuromorphic platforms (Dalgaty et al., 2018).

Orientation-selective Head Direction (HD) cells have also been discovered in rodents. Several models propose attractor networks to account for their selective firing behavior (Skaggs et al., 1995; Redish et al., 1996). Such attractor networks might self-organize to respond best to the observed sensory information (Stringer et al., 2002). These models have been mapped onto brain anatomy, explaining which brain regions might be involved in the encoding of angular velocity signals, the current head direction estimate, and the update mechanism (Goodridge and Touretzky, 2000). The detailed mapping of the HD neuronal circuits gave rise to a Spiking Neural Network (SNN) model in which persistent activity is realized through cross-inhibition rather than through recurrent excitation, as previously assumed (Song and Wang, 2005). The function of the HD network is to act as a neural integrator that is supervised by visual signals (Hahnloser, 2003) and supposedly is calibrated through angular velocity signals (Stratton et al., 2010). The vestibular information appears to be critical for generating the directional signal, and landmark information is important for updating it (Taube, 2007).

Inspired by the biological findings regarding navigation systems of insects and mammals, several computing architectures have been developed to estimate the position under uncertainty and re-calibrate it using familiar landmarks (Skaggs et al., 1995; Samu et al., 2009; Arena et al., 2013; Erdem et al., 2015; Seelig and Jayaraman, 2015; Heinze et al., 2018). An early successful attempt of a bio-inspired SLAM was the RatSLAM model—a biologically inspired SLAM system able to map indoor and outdoor environments (Milford et al., 2004). Recently, the original RatSLAM model was extended to function in 3D environments (Yu et al., 2019). Loop closure detection

was realized based on visual template matching (Gu and Yan, 2019), and multi-sensor fusion was shown to provide more accurate odometry and precise cognitive mapping (Zhang et al., 2019). Neural networks of grid cells have been shown to perform long-range navigation through path integration in the 2-dimensional plane (Edvardsen, 2017), and a model that was established through the Neural Engineering Framework confirms the attractor map implementation of path integration and proposes that the head direction signal can be used to modulate allocentric velocity input (Conklin and Eliasmith, 2005).

These computational approaches are complemented by approaches toward neuromorphic SLAM, which realized neuronal models in neuromorphic hardware. In this line of research, the formation of a 1D-map was demonstrated on a neuromorphic chip that could perform Bayesian inference using path integration and visual estimate (Tang et al., 2019). A model of the bat navigational system was realized in a neuromorphic VLSI device (Massoud and Horiuchi, 2012). This architecture includes a head direction ring attractor network (Massoud and Horiuchi, 2011a) and online correction through learned landmarks that are identified using sonar sensory signals (Massoud and Horiuchi, 2011b). Similarly, our previous work on neuromorphic SLAM, implemented on a miniature autonomous vehicle, incorporates a 1D head direction ring, 2D map formation, and a loop closure detection mechanism (Kreiser et al., 2018b,c, 2019a) based on vision. A neuromorphic system that can generate angular velocity and linear acceleration using IMU signals can be used as input to an HD network and was implemented on a VLSI chip to model the vestibular system (Corradi et al., 2014). More recently an SNN model was proposed for performing angular velocity regression on event-based visual data (Gehrig et al., 2020) that could potentially be used as input to an HD network when implemented in neuromorphic hardware.

Up until now, current approaches to neuromorphic implementations have been proofs of concept and either have not been deployed in a real-world scenario using a robotic agent or do not address the issue of scaling and performance under disturbances. In this work, we build on previous implementations for orientation estimation and use the biologically inspired head-direction network (Seelig and Jayaraman, 2015; Green et al., 2017; Fisher et al., 2019) to build an SNN model that estimates the pose of the robot's head through path integration using feed-forward commands and visual landmark detection. Compared to previous work on neuronal path integration, this work scales up the system to a higher resolution of pose representation, applies it to a 2D system of the robot's head, and quantitatively assesses the path integration performance.

We realize this model directly and fully in neuromorphic hardware—Intel's research chip, Loihi (Davies et al., 2018). We explore the model's function with a humanoid robot, the iCub (Metta et al., 2008), in the system designed to enable closed-loop experiments, i.e., the network controlling the robot's movement. The network tracks the movement in two degrees of freedom of the robot's neck. In our experiments, the iCub explores a wall with an object (a dotted pattern) on it by moving its head. Here, we do not use proprioceptive sensors (motor encoders or IMU) to estimate the robot's pose in an SNN; we use only the issued motor commands. This is done because we would like to estimate the precision of path integration in an SNN without mixing it with sensor errors in pose measurement. Moreover, sensors directly measuring the state of a joint are often not available in more complex motor systems or are costly (e.g., force sensors of compliant actuators). Such sensors can always be used to improve state estimation, similar to how vision is used in our model.

We use an event-based camera and simple visual preprocessing to estimate the position of an object in the field of view. More sophisticated event-based feature extraction could be used instead (Alzugaray and Chli, 2018; Gallego et al., 2019), but the visual processing was not our focus. When the object falls in the center of the visual field for the first time, the network stores the current pose of the robot's head, estimated in the network. Each time the object is seen in the center again, the stored pose is activated and used to correct the current pose estimate. The stored pose can also be used as long-term memory for object location and can trigger a goal-directed movement toward the memorized object, even if it is not in view.

The paper proceeds with a description of the hardware setup and the hardware and algorithmic interfaces between the robot and the neuromorphic chip. We then explain the SNN model and show results for pose estimation through path integration on-chip and vision-driven object-directed pose learning. We evaluate network performance in terms of the precision of state estimation and discuss how the SNN parameters influence it. Finally, we conclude with a discussion and the positioning of this work in state-of-the-art neuromorphic robotics.

2. HARDWARE SYSTEMS

2.1. The iCub Humanoid Robot

Our goal (beyond this paper) is to perform closed-loop experiments between the SNN and iCub. Therefore, we built an online interface between the humanoid robot iCub (Metta et al., 2008) and the neuromorphic device Kapoho Bay, which contains Intel's neuromorphic research chip, Loihi (Davies et al., 2018). An overview of the system is shown in **Figure 1**. We used YARP (Metta et al., 2006)—a middleware that allows seamless communication between different software components across the network—for modular processing and transparency between different computers and devices.

The neuromorphic iCub (Bartolozzi et al., 2011) has two event-based cameras, specifically, the Asynchronous Time-based Image Sensor (ATIS) (Posch et al., 2008), as part of its biologically-inspired vision system. The camera pixels produce asynchronous events as output. Each pixel emits an event when the level of sensed brightness changes by a certain amount. We used an event-driven visual tracking algorithm that produced "spikes" (event addresses) representing the target object position. This output was sent to the SNN on Loihi. Specifically, the visual

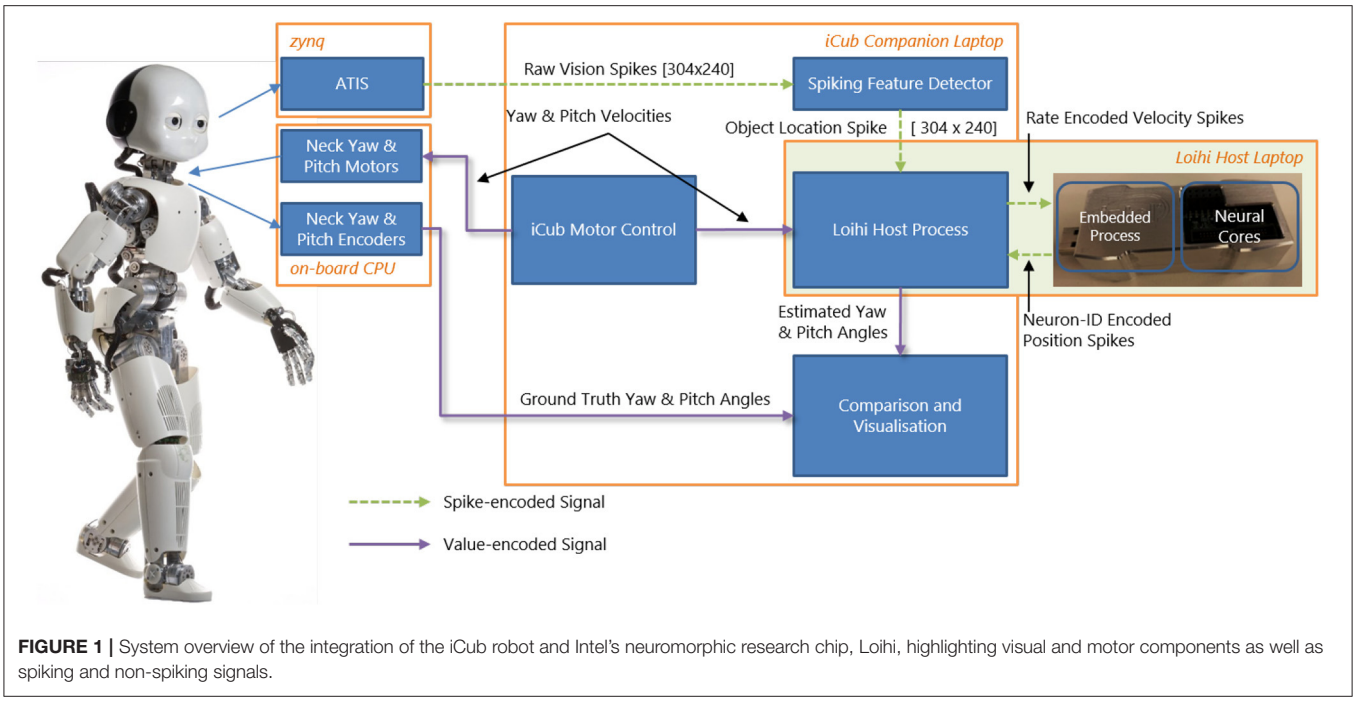


FIGURE 1 | System overview of the integration of the iCub robot and Intel's neuromorphic research chip, Loihi, highlighting visual and motor components as well as spiking and non-spiking signals.

input network in our model received input whenever the target object was in the field of view (as described in section 3.2).

The iCub's motors are controlled by sending velocity commands from a motor control loop. In our experiments, we moved the iCub head within two degrees of freedom, setting velocities for its yaw and pitch joints. We used movements at various speeds between the joint limits of the robot. The motor control module produced the following behaviors used in our experiments: a constant velocity along a single axis and a random-walk motion along both axes.

The iCub has encoders that count motor rotations for the six degrees of freedom of the head and neck. The encoders are usually calibrated by hand by initializing the robot in the 0° positions and measuring the encoder offsets to these positions. These offsets need to be updated whenever there is a mechanical change or after time due to wear. In this work, we did not use the position of the head read by the encoders to control the robot. Instead, we only controlled the motors' velocities along each axis directly, without sensory feedback on the motor position. Thus, the controller did not rely on the external calibration of the encoders. The head pose was estimated in the spiking head-direction network on the neuromorphic chip (section 4). The encoders were used solely to obtain the ground truth for experimental analysis and therefore provided no input to the algorithm.

The overall experimental system consisted of two laptops and the iCub robot connected to the same local network. We used the iCub middleware YARP (Metta et al., 2006) to connect different modules. For clarity, we briefly describe the exact computer configuration used. An iCub-companion laptop was used to run the motor control, which communicated with the iCub's on-board PC to move the robot. The motor commands (velocities) were at the same time sent over the network to the Loihi host laptop. The Kapoho Bay Loihi device was directly connected to this laptop by USB. The iCub companion laptop also read the raw camera events and ran the event-driven object detection algorithm. The object location spikes were sent to the Loihi host laptop. The output of the head-direction network was sent from the Loihi host laptop to the iCub companion laptop for visualization and recording; the encoder values were sent from the iCub on-board PC to the same module. During experiments, all signals were recorded except for the direct USB communication with the Loihi and the direct motor control with the iCub (the velocities sent to the Loihi host laptop were recorded instead).

Currently, to get these two cutting-edge, complex technologies to work together, the systems interface also has to be complex. One important contribution we make is to highlight this fact, with the aim of understanding how, in the future, we can develop a fully neuromorphic-integrated robot with fully spiking communications. We believe the system as we present it is still the required first step to doing so.

2.2. The Loihi Neuromorphic Research Chip

Intel Neuromorphic Computing Lab designed the neuromorphic research chip, Loihi, in which spiking neural network models can be simulated in real time efficiently (Davies et al., 2018). The chip consists of a mesh of 128 neuromorphic cores and three embedded ×86 processor cores. For this work, we used Kapoho Bay, the USB form factor version, which contains two Loihi chips. The chips are configured using a Python API provided by the Intel Neuromorphic Computing Lab (NxSDK 0.9) that allows us to define the spiking neural network on the level of groups

of neurons and synapses. Loihi implements the leaky-integrate-and-fire neuron model as described by Davies et al. (2018) and allows flexible on-chip learning.

2.3. Hardware Interface Between iCub and Loihi Using YARP

Figure 1 gives an overview of the interfaces between the iCub robot and the Loihi neuromorphic research chip. From the robot, a copy of the movement commands (section 3.1) and the visual information (section 3.2) are sent via YARP to the Loihi host computer. On the latter, a program is running that sends the values of the motor commands and the visual spikes to the embedded process running on the ×86 processor on the Kapoho Bay device. The embedded process sends spikes received from the host to the neural cores and reads the spikes from the previous time step to send them out to the host process.

At the beginning of each trial, the embedded process waits for an initialization input from the host computer to confirm that the robot or data player is sending events. While the experiment is running, the host process is synchronized to the embedded process and the neural cores so that they all advance their algorithmic time steps at the same time. On average, one algorithmic time step takes about 1.6 ms. Most of this time is taken up by the output being sent from the embedded processor to the host computer to monitor the spikes. Section 3.1 discusses the time step duration issues. The output spikes of the head-direction network—from the "goal head direction" layer—are designed to be sent through a YARP port, to eventually control the robot's gaze to one of the stored object locations in a closed-loop experiment; however, this is planned for future work.

3. ALGORITHMIC INTERFACES BETWEEN ROBOT AND NEUROMORPHIC CHIP

In this section, we describe how we generate input to the SNN on-chip model from motor commands and camera events and how we read out SNN activity.

3.1. Input Spike Generation Based on Velocity Commands

When the robot moves its head, the velocity commands are sent to both the robot and the host computer of the Loihi chip. On the host computer, a small C++ program receives the velocity commands that are interpreted as the neuron's input current I_{in} (in $^\circ/s$) after being multiplied by the measured timestep duration on Loihi. Four of Loihi's integrate-and-fire neurons are dedicated to integrating the velocity input for yaw (left and right) and pitch (up and down) movements. A change in velocity, therefore, leads to an immediate change in input current and, with that, changes the neuron's membrane potential $V(t)$, Equation (1).

$$\begin{aligned} &\Delta V(t) = I_{in} \cdot \Delta t - V_{thr} \cdot \Theta(V(t) - V_{thr}), \quad \text{where} \\ &\Theta(x) = 0, \quad \text{if } x \leq 0; \\ &\Theta(x) = 1, \quad \text{if } x > 0. \end{aligned} \tag{1}$$

Here, at every timestep, t, the current speed command I_{in} (in $^\circ/s$) is multiplied by the measured duration of the timestep and added to the neuron's membrane potential $V(t)$. Note that timesteps may take a variable amount of time in the system depending on spiking rates and other computational overhead. When $V(t)$ of the velocity neurons surpasses a threshold value V_{thr}, the neuron emits a spike, and the magnitude of V_{thr} is subtracted from the membrane potential to reset the neuron. The *firing rate* of the velocity neuron is thus proportional to the velocity command sent by the motor controller of the robot, and the proportionality coefficient can be controlled by the threshold parameter V_{thr}. The emitted spikes then stimulate the shift layer of the SNN model, as explained in section 4.

The value of $V(t)$ is clipped at a maximum value of $V(t) = 2V_{thr}$. Furthermore, we added a refractory period that prevents the input neuron from firing more often than every third timestep, which is the time that the head-direction network needs to fully integrate an input spike in our "every spike matters" setting.

The threshold V_{thr} of the simulated velocity input neurons determines the quantization step and the path integration rate of the network. For instance, if we set $V_{thr} = 0.5^\circ$, the velocity neuron will produce a spike whenever the robot has moved its head by 0.5°. This spike shifts the current estimate of the head angle in the head-direction network's activity by one neuron within $n = 3$ time steps, so we need 200 neurons to represent an angle of 100°. Assuming an average timestep duration of 1.6 ms, we can calculate that if we set the threshold to $V_{thr} = 0.5^\circ$, the SNN activity can faithfully follow an angular velocity of approximately $\omega = \frac{V_{thr}}{n\Delta t} \approx 100\,^\circ\,\mathrm{s}^{-1}$. This sets the maximal speed at which the activity can be shifted in our SNN model.

The timestep can be further shortened (and the maximal velocity increased) by optimizing the I/O from the chip. The duration of timesteps fluctuates as SNN simulation unfolds in real time. **Figure 2** shows the distribution of the measured timestep duration in our experiments.

Note that all other neurons besides the velocity input neurons receive spikes from connected neurons as input instead of a direct change in current. The input to these neurons is the sum of filtered spikes from connected neurons, leading to a synaptic response current $I_{in}(t)$ (Davies et al., 2018). By default, after each spike, the membrane potential $V(t)$ is reset to zero instead of subtracting the threshold. Although this neuron model is often used as an input integrator in computational neuroscience, it might lead to "loss" of input current at large inputs and consequently to an error at the value-to-spikes interface. We thus introduced the "soft-reset" in the input layer to achieve maximum accuracy of pose representation in the network. At timepoints with a reliable visual reset or when other external sensing can be used to correct path integration, this input-integration error can be neglected.

3.2. Spiking Object Detector

The fundamental purpose of the vision system is to give a consistent signal about head pose that is not affected by integration drift. The signal is not explicitly known *a-priori*, i.e., we don't know where an object will be, but given any pose, the visual signal will not change over the course of the experiment.

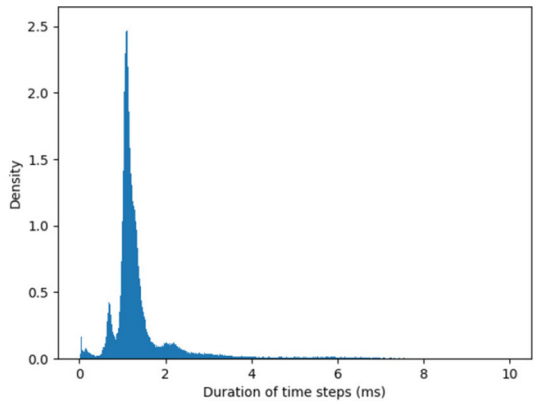


FIGURE 2 | Histogram of the algorithmic time step duration as recorded by YARP in our experiments. The average timestep is 1.6 ms, but in rare cases, time steps can be as long as 40 ms. Note, these values hold for the specific version of the Loihi API used.

However, the relationship between pose and visual signal still needs to be learned during the experiment.

Our eventual goal would be to use an event-driven object detection system (e.g., Liang et al., 2019) as an SNN inside the Loihi chip. However, in this paper, we are focusing on the path integration and visual reset in the SNN rather than a complex visual system. One other current integration issue is the data bandwidth to the Kapoho Bay, which limits the amount of visual data that can be sent to the chip. Therefore, for this paper, we used a simplified spiking object detector outside the SNN (in software) based on a visual tracking algorithm to encode the position of a single object in the visual field of view. In our current setup, we consider only a single object in the visual field of view. However, more complex recognition systems can be extended to multiple objects.

The spiking object detector receives the raw events from all pixels of the ATIS neuromorphic camera (Posch et al., 2008) and outputs spikes associated with the object position, O_{xy}. The output array of the detection, therefore, has the same dimensions as the sensor itself, 304 × 240 pixels. The starting position of the object of interest O^0_{xy} was marked in an initialization phase in which the user sets the correct position of the object. Following initialization, tracking was achieved by setting a region of interest of size R_{size} around the initialized point. When N_{events} camera events were received in the region of interest, the mean position of the events was calculated, and the output neuron produced a spike at the position of the object in the visual field, O^t_{xy}. The new region of interest was defined around the updated mean-firing position, O^t_{xy}, and the events were again accumulated within the region of interest in order to produce the subsequent spike.

The output of the object detector is event-based: its firing rate depends on the rate of camera events within the region of interest. A single spike is output at the moment in time that the object position moves by 1 pixel. The resolution of the temporal precision of the output is under 1 ms.

The detector's output spike is sent to the spike-generator interface on the Loihi host computer and sent to the neuronal cores, to the visual input network. The visual input network receives the detector spikes according to their position in the visual field with a rectangular 2 × 2 pixels receptive field. The central neuron of this array activates the visual reset neuron.

3.3. Reading Out the Head Direction From the Network

At every timestep, a data package containing the indices of the currently firing neurons ("address event representation") is sent by the Loihi embedded process to the Loihi host computer. Since the total processing time is dominated by sending the output packages to the host computer, we only record the neuronal populations required for the system's performance evaluation. In our place-code representation, the spike's index ("address") directly corresponds to the represented variable value, e.g., the yaw or pitch.

4. THE HEAD-DIRECTION SNN

4.1. Network Overview

The path integration network consists of two identical SNNs for yaw and pitch estimation (**Figure 3**). Each of these SNNs, similar to networks used in Kreiser et al. (2018a,c), consists of six layers of $N = 200$ neurons each:

- the current head direction layer (CHD),
- the shift left layer (SL),
- the shift right layer (SR),
- the integrated head direction layer (IHD),
- the reset head direction layer (RHD), and
- the goal head direction layer (GHD).

The input to each of the two networks comes from two velocity populations [one for clockwise ("right") and one for counter-clockwise ("left") movement] and several visual-landmark (visual-reset) populations. We can have as many of these populations as there are landmarks or objects known to the robot. As Loihi is a digital, deterministic neuromorphic system that does not require redundancy to cope with mismatch and noise in neuronal dynamics, each of the velocity input and visual landmark populations consists of a single neuron in our implementation.

4.2. Functional Description of the Network

In the *Current head direction (CHD) layer* of the yaw and pitch path integration SNNs, the current pose (yaw or pitch, respectively) of the robot's head is encoded in the position of the active neuron: each neuron corresponds to a specific value of yaw or pitch. We thus use one-hot encoding. At the start of every trial, the neuron that codes for the initial position (the central neuron) is activated. An active neuron in the CHD layer inhibits all but one neuron in all Shift layers: only neurons with the same index as the active CHD neuron can be activated.

The *Shift layers* are responsible for shifting the position of the active neuron to the left (Shift Left, SL) or the right (Shift Right, SR). An entire shift layer is activated by the respective velocity

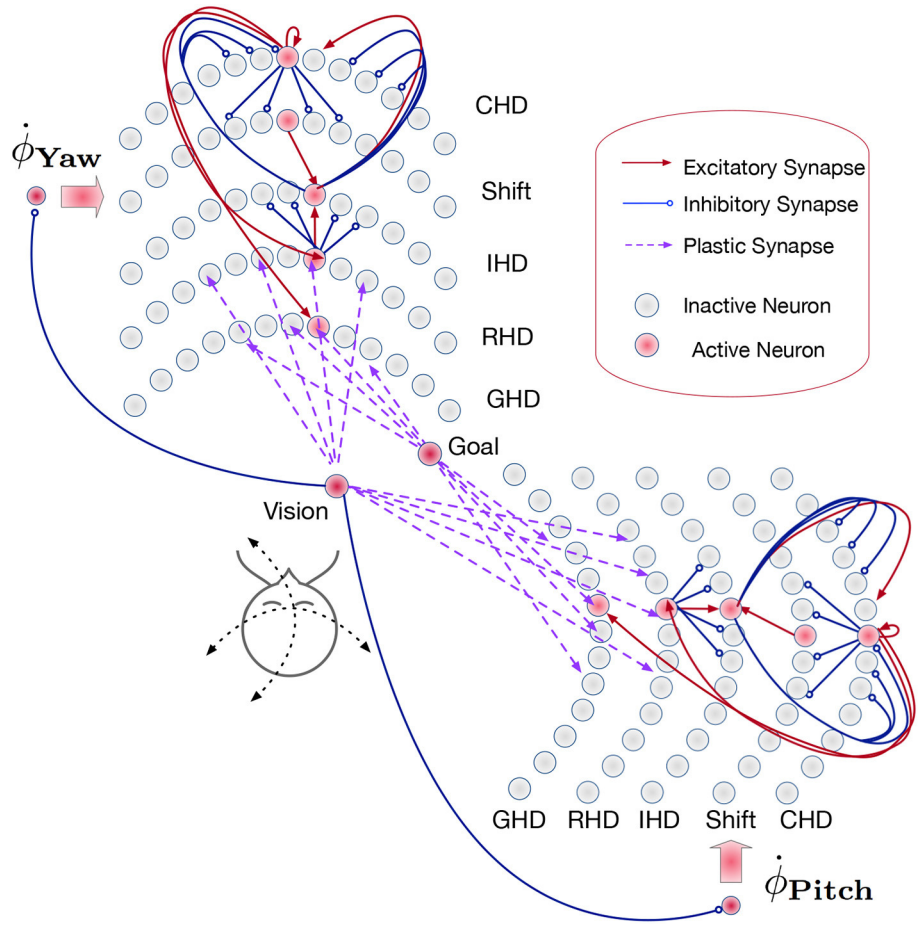


FIGURE 3 | Overview of the path integration and map learning network: Two identical SNNs with five functional layers each estimate the yaw and pitch of the iCub's head pose, integrating the respective motor velocities. When the robot's gaze is directed at a visual landmark, the yaw and pitch angles are stored in plastic synapses connecting the Vision and Goal neurons to the Reset Head Direction and Goal Head Direction layers, respectively. Excitatory synapses (red lines) between layers connect neurons in a one-to-one manner. The velocity input neurons are connected to shift layers in a one-to-all manner. Plastic connections (purple lines) are one-to-all, and inhibitory connections (blue lines) are all-to-all-but-one. Only exemplary connections are shown, in order to avoid clutter. See section 4.3 for details.

neuron through a "boosting" one-to-all connectivity pattern. The SL layer is activated by a counter-clockwise movement command and the SR layer by the clockwise movement command. During visual reset, the shift layers are inhibited by the visual landmark population. Both shift layers project their activation to the IHD layer with a one-to-one-shifted connectivity pattern. The IHD layer integrates this "shifted" CHD activation with the visually-driven reset input.

The *Reset head direction (RHD) layer* is active when it receives input from the visual-landmark population, which arrives through plastic connections that are learned when the landmark is seen in the center of the visual field for the first time. The plastic weights store the pose (head direction angle: yaw or pitch) that the robot had when it was looking at the landmark for the first time. When the landmark is revisited, the strong potentiated plastic synapses drive an activity bump in the RHD layer. Weak input from CHD is not sufficient to induce activity in the RHD on its own. If the RHD layer is active, it resets the activity in the IHD layer through a set of strong weights: an active RHD neuron excites the corresponding neuron in the IHD layer and inhibits all other neurons (the "reset" connectivity pattern).

The *Goal head direction (GHD) layer* behaves exactly the same as the RHD layer but is only a readout population with no outgoing connections to the other parts of the network. It is used in a scenario of goal-directed behavior to look at the learned object. It receives the same subthreshold activation from the IHD layer and additionally receives input through plastic synapses from a goal population that is activated by the visual landmark input. As in the RHD layer, the plastic weights leading to the GHD layer act as a memory that associates a specific landmark with a pose.

Finally, *Integrated head direction (IHD) neurons* project in a one-to-one manner to the CHD layer, also with inhibition to all other CHD neurons ("reset" pattern), thus either shifting the activity location if no visual landmark is detected or resetting this activity to an updated location if a visual landmark dictates such an update.

4.3. Connectivity in the Head-Direction SNN

To achieve the described behavior, the layers of the model are connected as shown in **Figure 3**. Note that all weight values in the description below and the parameter tables are given in multiples of the neuron threshold ($V_{thr} = 100$ here).

In the CHD layer, every neuron excites itself with a weight of $w_{CHD\_CHD} = 1.2$ so that the activity of the network is self-sustained. I.e., the current pose is stored until it is set to a different position by the IHD.

The CHD layer is connected to the shift layers with all-to-all-but-one inhibition ("negative preshape" connectivity pattern) so that only the corresponding neuron can be activated by the one-to-all excitatory input from the velocity populations ("boost"). The shift layers have shifted one-to-one synapses to the IHD.

RHD and GHD receive a subthreshold ($w_{CHD\_RHD} = w_{CHD\_GHD} = 0.2$) one-to-one input from the CHD layer that is used to learn the initial pose of the landmark ("preshape" connectivity pattern).

Plastic one-to-all connections from the visual-landmark population add input to the RHD neurons and drive the neuron with a pre-shaping CHD input above the threshold. The plastic connections between this neuron and the visual landmark neuron are updated then and store the pose of the iCub's head that corresponds to having the landmark in the center of the visual field. After learning, plastic connections form a one-to-one connectivity pattern: a single synapse from the visual neuron to the correct RHD neuron is potentiated (high); all other synapses are depressed (low).

A goal neuron is connected via one-to-all plastic synapses to the GHD layer. The goal neuron can be driven externally to remember the pose-landmark association without resetting the current pose estimate through the IHD layer. The goal neuron receives excitatory one-to-one connections from the visual landmark neuron to learn the pose-landmark associations.

The RHD layer has one-to-one excitatory and all-to-all-but-one inhibitory connectivity (the "reset" pattern) to the IHD layer to override input from the shift layers when the visual reset is active. The IHD layer is connected to the CHD layer with all-to-all-but-one inhibition ($w_{IHD\_CHD\_inh} = -1$) to delete the previous state and one-to-one excitation ($w_{IHD\_CHD\_exc} = 1.24$) to "copy" the current state.

The learning rule of the plastic synapses between the visual-landmark neurons and the RHD/GHD layer is specified as follows:

$$\Delta w = y_0 \cdot x_1 - \lambda \cdot x_0, \quad \text{if } w < w_{\max}. \tag{2}$$

Here, Δw is the weight update at a given timestep. $x_0 \in \{0, 1\}$ and $y_0 \in \{0, 1\}$ are variables that become 1 if there is a pre- or post-synaptic spike, respectively. x_1 is a variable that stores an eligibility trace of the pre-synaptic neuron activity, and it decays over n time steps ($n = 2$ here, since the post-synaptic spike should arrive in the next time step); $w_{\max} = 256$ is a maximal weight value at which weights saturate.

According to the learning rule (Equation 2), a synapse potentiates if an RHD neuron (post-synaptic) fires after the visual-landmark neuron (pre-synaptic) fired. The closer in time the visual-landmark neuron fires to the RHD neuron, the higher the pre-synaptic trace x_1, leading to a more significant weight update. Synapses are depressed (decrease) by a constant factor of λ if the post-synaptic (RHD) neuron did not fire but the pre-synaptic neuron (visual) did fire. The weights are initialized to a subthreshold value ($w = 0.8$) so that, together with the input from the CHD, their summed input activates the RHD at the currently estimated pose. This leads to one-shot learning of the pose, while all other synapses that connect to non-active RHD neurons are depressed to 0. Plastic synapses between the goal and GHD neurons are learned in an online fashion throughout the whole experiment: learning was not artificially stopped at any time.

TABLE 1 | Values of synaptic weights between layers in the head-direction SNN on Loihi and parameters of neurons.

Parameter	Value	Parameter	Value	Parameter	Value
w_{CHD_CHD}	1.2	w_{CHD_Shift}	−0.5	$w_{Velocity\_Shift}$	1.0
w_{Shift_IHD}	1.0	$w_{IHD\_CHD\_exc}$	1.2	$w_{IHD\_CHD\_inh}$	−1.0
w_{CHD_RHD}	0.2	$w_{RHD\_IHD\_exc}$	1.0	$w_{RHD\_IHD\_inh}$	−0.7
$w_{vision\_RHD\_initial}$	0.8	V_{thr}	100	τ_V, τ_i	1

All weights are given as multiples of V_{thr} (i.e., they are multiplied by 100 before being set on the chip). Both time constants of neurons (τ_V) and synaptic temporal filters (τ_i) are set to 1 timestep, which means that each neuron's membrane potential is reset after every timestep, i.e., neurons do not keep a state*. Memory in the network is maintained using the recurrent connectivity.*

Table 1 lists all neuronal and synaptic weight parameters used in the head-direction SNN and their values.

5. EXPERIMENTS AND RESULTS

We describe experiments in which the proposed SNN model estimates the head pose of the iCub robot. Three evaluations were performed:

- We assessed the accuracy of the integration component of the head-direction SNN without visual input.
- We assessed the improvement of the network with visual learning and reset.
- We investigated the representation of the object location in the SNN and how it relates to the map creation.

5.1. Experimental Setup and Dataset

Data for repeatable experiments were produced using the *neuromorphic iCub* robot (Bartolozzi et al., 2011). We evaluated the entire system using online, live experiments connecting the SNN and robot. However, the results presented were produced on recorded data to ensure reproducibility. The datasets are available permanently and can be downloaded here[1]. There are five datasets with random head movements and a simpler one with a squared movement of the head. Each dataset contains ATIS features, visual tracker output, motor commands, and

[1]https://services.ini.uzh.ch/permlink.php/puGu3hai

robot encoder values. All datasets start with a calibration phase (0–25 s) where the robot performs independent yaw and pitch movements.

The yaw and pitch motors of the iCub head were controlled to maintain the desired velocity during operation. By using only velocity commands, the head-control module has no information about the position of the joints, and in the main part of the architecture, the only module that estimated the pose was the head-direction SNN. We used the encoder values in auxiliary functions to enforce joint limits (to avoid robot damage) and to center the head between trials, which was required for repeatable experiments. Encoder values were also used as a ground truth against which we compared the activity of the head-direction network. All parameter values used in experiments are listed in **Table 2**.

The iCub robot performed the following behaviors:

- **Home:** The head is moved to the center of the workspace, controlling the velocity, such that new trials begin with identical pose.
- **Nodding:** The robot nods its head upward and downward between the joint limits (j^0_{min} and j^0_{max}). No horizontal motion.
- **Head-shaking:** The robot shakes its head side-to-side between the joint limits (j^2_{min} and j^2_{max}). No vertical motion.
- **Random:** The robot chooses random velocities at which to move both vertically and horizontally. The velocity is chosen as a uniform distribution between 0 to v^0 and 0 to v^2 for the vertical and horizontal motion, respectively. If a joint limit is reached, the velocity of the respective joint is reversed. New velocities are chosen after $r_{timeout}$ seconds.

In addition to direction, we changed the speed of the robot's movements: v^0_1 and v^2_1 are the base velocities used, and during experiments, the speed was increased such that v_2 is double and v_4 is four times the base speed, applied to both joints simultaneously (see **Table 2**).

Five datasets were recorded with the robot beginning in the home position and then proceeding with the following strategy: nodding, home, head-shaking, home, random with speed v_1 for ∼30 s, random with speed v_2 for ∼30 s, random with speed v_4 for ∼30 s, and finally, home. The data were recorded from one of the ATIS cameras on the robot after a pre-processing stage to eliminate the saving of uninformative events (a noise filter). The motor-control module output the velocity of the head when the commanded velocity changed; the data were saved along with the iCub head encoder values. All data were timestamped to synchronize during playback correctly and for further processing.

The data were saved and processed offline to enable a repeatable analysis of the visual integration; however, the entire pipeline was also tested with the robot in the control loop. Therefore the system is capable of estimating the robot's head pose and memorizing object-directed poses in real time during the robot operation.

The robot was positioned to look at a dot pattern, which was chosen because it produced a strong signal in the visual stream (**Figures 4A,B**). The background of the scene was predominantly a blank wall to avoid the interference that would be introduced by a cluttered scene (as visual processing was not the focus of this experiment) but also included desks and windows. The dot-pattern was placed in different positions relative to the robot for each of the five datasets. The position of the dot pattern in the visual array of the ATIS sensor was extracted from the visual stream by visual tracking (**Figure 4C**), section 3.2. Both the position of the dot-pattern and the commanded velocities of the robot's head motors were sent to the Loihi host process to be converted to spikes compatible with the SNN on the Loihi neural core. The SNN produced the estimate of the iCub's head position, which was recorded and compared to the encoder values and path integration in software.

TABLE 2 | Parameters of iCub movements in the experiments.

Parameter	Value	Parameter	Value
j^0_{min}	−20 deg.	j^2_{min}	−35 deg.
j^0_{max}	10 deg.	j^2_{max}	35 deg.
v^0_1	7 deg./s	v^2_1	14 deg./s
v^0_2	14 deg./s	v^2_2	28 deg./s
v^0_4	28 deg./s	v^2_4	56 deg./s
$r_{timeout}$	3 s		
R_{size}	50 pix.	N_{events}	1000

5.2. Integration-Only Pose Estimation

The head-direction SNN was initially evaluated on its ability to integrate the velocity commands to estimate the position, without any correction from the visual system. The robot began each trial in the center of the workspace. The head-direction network was initialized with an active neuron in the center of the CHD layer. With the correct input threshold (calculated as described in section 3.1) applied to the integration dynamics of the network, we achieved a close correspondence between the ground-truth pose calculated in software and yaw and pitch angles estimated by the SNN, as can be qualitatively seen in the time course of, e.g., experiment 1 shown in **Figure 5A**. Here, the yaw and pitch angles estimated in the SNN on-chip (blue line) and in software (orange line) overlap perfectly. **Table 3** lists the RMSEs of dataset 5 using different thresholds V_{thr}. We also show the value measured by the motor encoders (green line) for completeness. Trajectories in the 2D joint (yaw-pitch) space for all five datasets are shown in **Figures 5B–F**. Note that in the last example, the actual movement, measured by the motor encoders, deviates more strongly from the motor commands, which will be noticeable later in the learned map.

Quantitatively, the RMSE between the estimated pose and pose measured by the encoders was 1.93° and 2.43° for the yaw and pitch angles, respectively, for our 2-min long experiments. Errors compared to the pose measured with motor encoders appear due to the inertia of the robot's movements. The encoders capture the actual position of the motors, and the joint motors are affected by inertia and other higher-order dynamics; i.e., the head cannot instantly change velocity. The head-direction network, to the contrary, immediately integrates the changing velocity—the motor commands are integrated precisely.

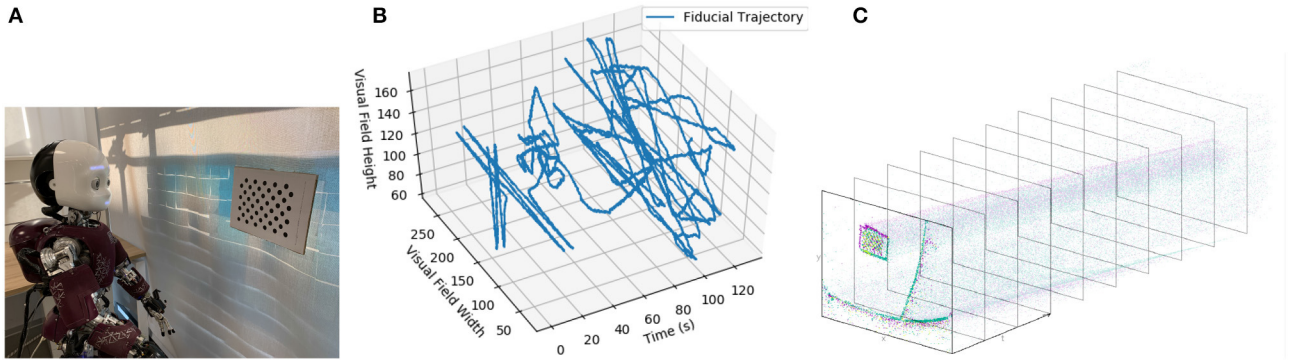


FIGURE 4 | (A) The iCub robot and visual fiducial (dot pattern). **(B)** The visual trajectory of the fiducial over dataset 1. **(C)** An example of the ATIS camera output.

Further, we compared the estimated pose from the CHD layer to a software-based integration of the command signals performed with precise floating-point computation. To compute errors, we sampled the software pose estimation with a fixed interval of 2 ms and linearly interpolated to the spike times of the SNN-based pose estimation. The error was found to be 0.31 and 0.58° for the pitch and yaw angles, respectively. **Table 3** lists the RMSEs of dataset 5, comparing the SNN estimation with command-based path integration in software for different settings of V_{thr}. A $V_{thr} = 2$ corresponds to a network with $N = 25$ neurons in the CHD layer, $V_{thr} = 1$: $N = 50$, $V_{thr} = 0.5$: $N = 100$, and $V_{thr} = 0.25$: $N = 200$.

These encouraging results demonstrate the strong potential of the network as a head-direction estimator.

5.3. Visual Reset of Imprecise Pose Tracking

A correctly parametrized head-direction SNN is able to integrate the motor commands with high accuracy. However, when running for longer periods and with other (external) disturbances, it cannot be guaranteed that the estimate will always remain accurate. To simulate a disturbance within the shorter time frame of our recorded datasets, we artificially introduced a bias into the network. We show that the visual input and the visual reset layer of the network allow the pose estimation to be corrected.

To corrupt the path integration, we multiplied the velocity signal, I_{in}, in Equation (1) by a factor of 1.1 for the clockwise direction. We applied this bias after ∼45 s (30,000 time steps) of the experiment. The performance of the resulting biased network can be seen in **Figure 6A**. Here, the yaw and pitch estimated in the CHD layer of the SNN (blue line) diverge from the software-integrated commands (green line) after the 45-s mark.

The visual reset component of the network allowed the estimated head direction to be corrected. In **Figure 6B**, the same biased network is used, but the active neuron in the CHD layer is reset when the target object is again seen in the center of the visual field. At the points of visual reset, the pose "jumps" to the pose learned when the robot looked at the object for the first time, thereby correcting the pose estimation. The RMSE, when compared to encoder information (see **Table 4**), is 6.05° in pitch and 11.10° in yaw for the corrupted network without visual reset and 4.47° in pitch and 8.98° in yaw when the detected landmark corrects the network during visual reset. The visual correction could potentially improve overall performance, removing the discrepancy between the motor commands and actual movements. However, our visual preprocessing itself was not precise enough to achieve improvement here.

The visual reset component of the network is potentially more than just a correction tool. As we have shown previously (Kreiser et al., 2019a,b), this "loop closure" event can also be used to calibrate the gain of path integration, such that manual parametrization of the velocity input layer becomes unnecessary.

5.4. Representing the Visual Scene in the Network (Map Formation)

Although our simple visual pre-processing did not allow us to use multiple objects in the visual scene, the network can learn poses that correspond to looking at multiple objects. To demonstrate this, we concatenated five datasets with different object positions (two positions were the same). Each target was considered a unique object, and a new pose was learned for each object without forgetting the other ones. To achieve this, we introduced multiple visual landmark neurons. Each landmark neuron was activated by the object detected in the central part of the field of view. Here, we let different landmark neurons be activated in each of the five datasets. This manual neuron selection is a placeholder for the output of a fully-fledged object recognition system (e.g., Liang et al., 2019).

The network was successfully able to store multiple different objects with the plasticity mechanism described in section 4.3. We visualize the learned object-directed poses by activating each of five visual landmark neurons and reading out activity in the goal head direction (GHD) layer. The resulting 2D motor poses are shown in **Figure 8B** (colored crosses), compared to the ground truth of the encoder values read out when

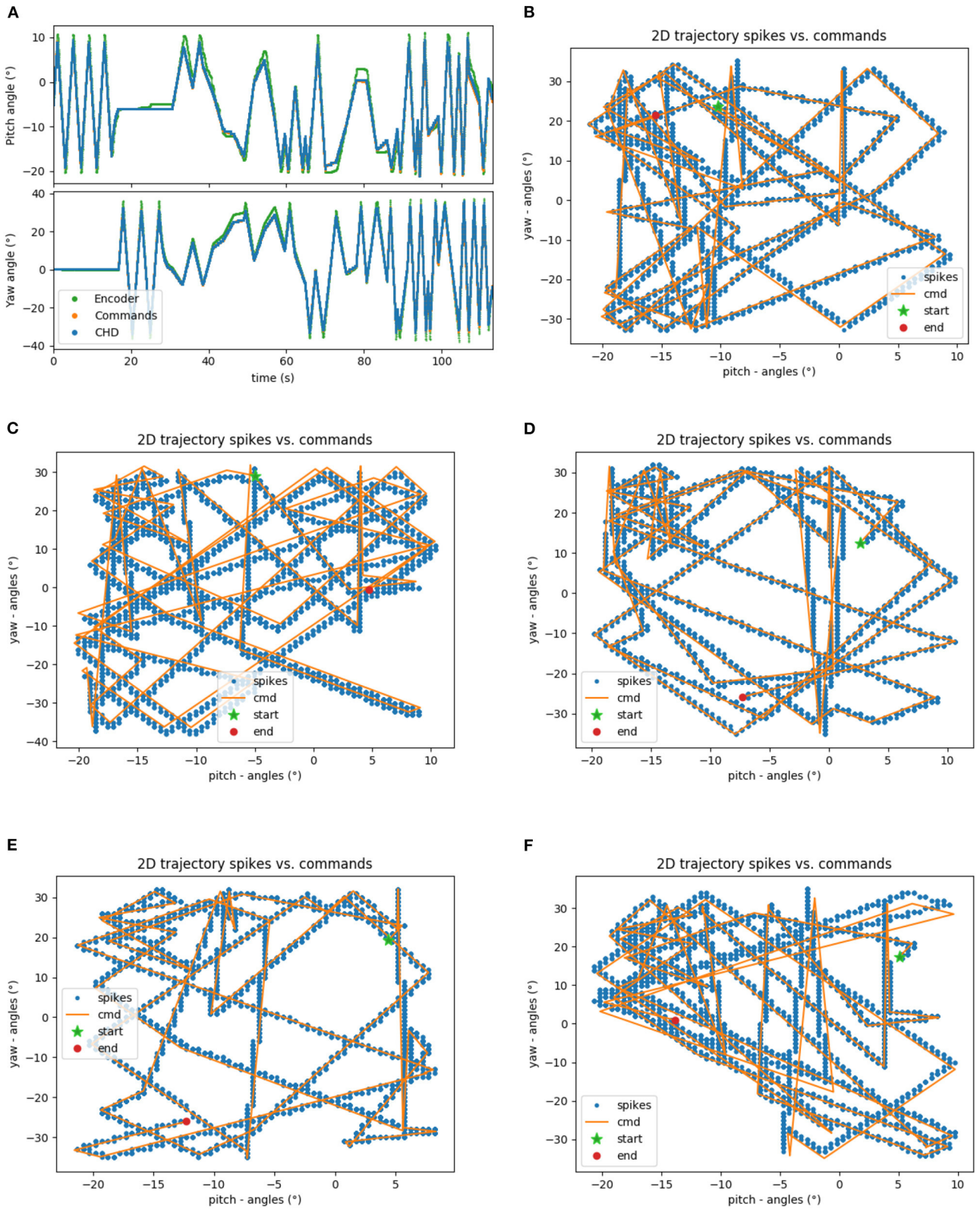


FIGURE 5 | Results of robotic experiments on path integration in head-direction SNN on chip. **(A)** The estimated yaw and pitch angles over time for dataset 5. A match between the SNN-estimated pose and integrated motor commands can be observed, with small deviations from actual movement as measured by the motor encoder. **(B)** The same trajectory in 2D motor space. **(C–F)** Trajectories for the datasets 1, 2, 3, and 4, respectively.

the robot was centering the target object in its field of view (colored squares).

Figure 7 shows the time course of the whole experiment with five concatenated datasets. Visual resets occurred throughout the entire 10 min of dataset, periodically correcting the drift that accumulated over time. This demonstrates that visual reset is helpful even if the path integration is precise when the pose is tracked for a long time.

At the end of the experiment, the goal neurons that represent the five different landmarks were activated one by one, and the associated pose was recalled through the activity of neurons in the GHD layer.

Figure 8A shows the five target locations in the camera's field of view, extracted from the events of the object tracker during the calibration phase when the robot's head is moved up and down and left to right. The location of each object was extracted from the intersection of event-traces during the yaw and pitch movements. **Figure 8B** shows the respective five positions in the robot's motor space: recorded from the encoders at the time when the landmark was in the center of the visual field (squares) and learned by the SNN (crosses). Each marker represents the pose that the robot needs to take to "look" at the respective object, i.e., center an object in its visual field. Note that there is a close match between the estimated pose and the ground truth (measured movement).

The learned landmark-centering poses can be used to direct the robot's gaze to the memorized object locations. E.g., the vector-integration to end-point (VITE) neuronal motor-control model generates movement based on the currently estimated pose and the stored goal pose (Grossberg, 1988). Alternatively, one can use a saccadic eye-movement-generating neuronal architecture (Bell et al., 2014; Sandamirskaya and Storck, 2014, 2015) to initiate the gaze to the memorized pose.

Note that during all of our experiments with recorded data, the data were fed to the SNN on-chip in real time, at a speed at which the real robot would provide the same data. Thus, no re-parametrization of the network was needed to run closed-loop experiments with the robot, and learning can proceed alongside the behavior in real time.

6. DISCUSSION

In this work, we applied elements of neuromorphic SLAM—neuronal path integration, visual reset, and map learning—in the new setting of a humanoid robot observing a visual scene. The main results of this work can be summarized as follows:

- We have shown that even a small population of spiking neurons can perform precise path integration of motor commands to obtain an estimation of the current pose of

TABLE 3 | Root mean squared errors (RMSE) in degrees for different thresholds of velocity input neurons, V_{thr}, for the yaw and pitch estimation.

RMSE (in °)	$V_{thr} = 2$	$V_{thr} = 1$	$V_{thr} = 0.5$	$V_{thr} = 0.25$
Pitch	0.86	0.95	0.31	0.23
Yaw	1.54	2.04	0.58	0.58

RMSEs are calculated based on time-aligned differences in angles estimated from the CHD layer of the SNN and calculated by path integration in software.

TABLE 4 | Root mean square errors (RMSEs) of pose estimation by the biased head-direction network with and without visual reset.

RMSE, °	With visual reset	Without visual reset
Pitch	4.47	6.05
Yaw	8.98	11.10

RMSEs are calculated between SNN output and motor commands integrated in software.

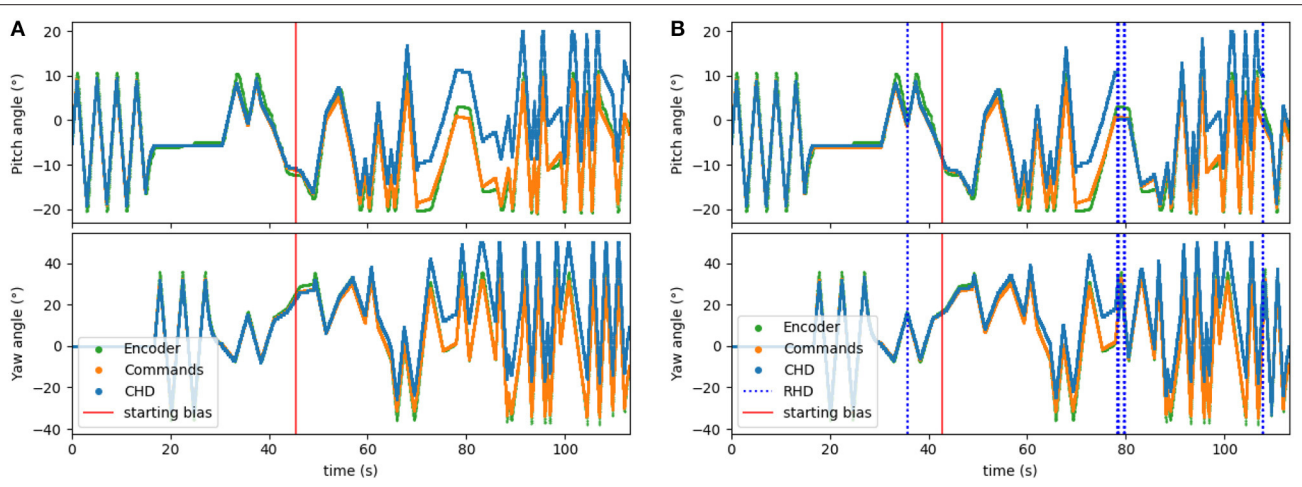


FIGURE 6 | Testing the visual reset. After ~45 s (30,000 time-steps), the clockwise velocity signal is scaled by a factor of 1.1 as a simulated disturbance in the neuronal estimation of head direction. **(A)** Pose estimation without visual reset diverges from the ground truth. **(B)** Pose estimation with visual reset when the object is revisited. The dashed blue lines indicate the presence of the visual stimulus in the center of the visual field. The first blue line (around 36 s) indicates when the object was learned.

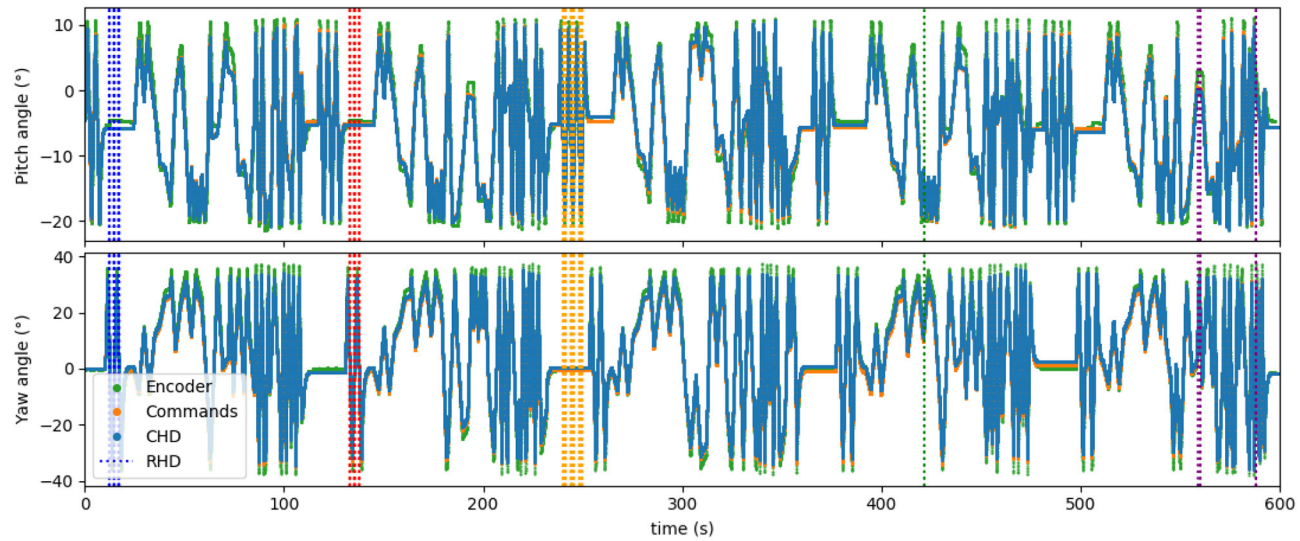


FIGURE 7 | Five datasets concatenated in time, leading to about 10 min of recorded data. Visual learning and recall are indicated by the dashed vertical lines, with each color indicating a "new" object (object in a different location). Five different poses are stored in the network.

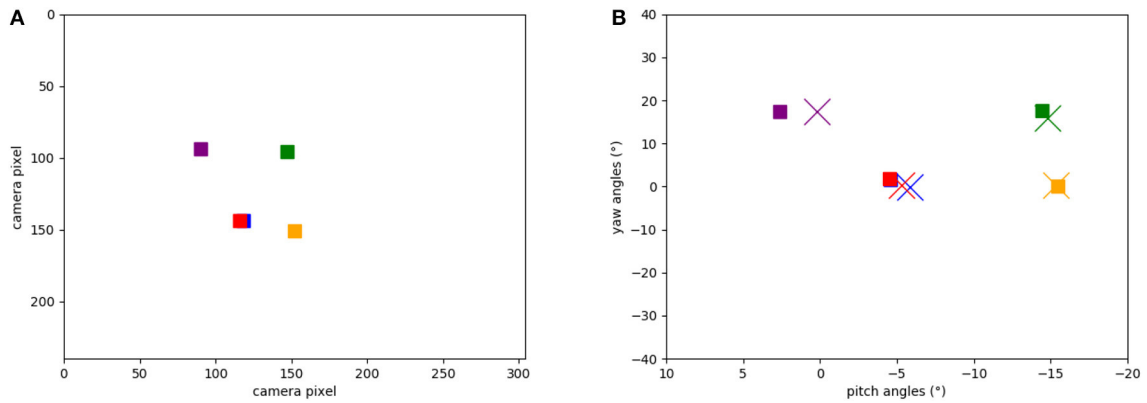


FIGURE 8 | Positions of the objects in our experiments: **(A)** Positions of the objects calculated from ATIS events (in camera pixels when the robot looks straight). **(B)** Learned motor poses in the head-direction network (crosses) and object-directed poses calculated from the encoders (squares). Note that red and blue squares are on top of each other.

the robot's head. The error, compared to path integration in software, accumulated over 120 s of the experiment, was at the resolution of value representation, $< 1°$, for the network with 100 neurons representing 100°.

- We have shown how error that is accumulated due to imperfections of the robot (motor commands do not perfectly correspond to executed movements) can be corrected with external sensing, i.e., vision.
- We have demonstrated online learning of the reference pose in a closed behavioral loop, i.e., with the weight adaptation occurring in parallel to the robot's movements and path integration. Plastic weights are updated in timesteps, in which the learning conditions are fulfilled: the respective pre- and post-synaptic spikes co-occur in the same timestep. These updates can lead to one-shot learning (as shown here). The network can also be configured to require several co-activations of pre- and post-synaptic neurons for the updated weight to have a noticeable effect after the learning increment. We have shown how multiple objects can be stored in the network by adding one "label" neuron per object and a set of N plastic synapses, where N is the size of the head direction network layers ($N = 100$ here).
- We have introduced a number of structural motifs that solve computational tasks involved in path integration and map formation, that is, setting, resetting, and shifting connections and boosting and pre-shaping, as well as input and output interfaces between the non-spiking periphery and the neuromorphic chip.

When presenting the SNN model, we paid particular attention to the computing modules that realize important computational primitives that can be reused as building blocks in other tasks and on other neuromorphic hardware. Thus, we hope to contribute to building-up a neuromorphic "instruction set" that will allow us to design neuronal models, in particular for robotic tasks. Such neuronal models can be built using known biological neural circuits, creating a complementary approach to data-driven learning, which can be too costly in learning time and data-preparation effort for some applications. In comparison to previous work on neuromorphic head direction estimation, we evaluate system performance in a real-world task to estimate the 2D head pose of a humanoid iCub robot, particularly emphasizing the required interfaces between different hardware components. We have shown that disturbances can be mitigated by using information of different sensory modalities, and we evaluated how the path integration error relates to scaling of the network.

The main contributions of this work that we would like to emphasize are:

- We use "place code" to represent values (e.g., the angles of the head's pose): we represent values by the identity of the most active neuron (or localized region) in a neuronal population (layer). In particular, taking advantage of Loihi's precise nature, we use "one-hot" and "single-spike" encoding here, making every spike matter in our network.
- We show an example of combining rate code and place code to represent values in an SNN architecture, and we show how non-spiking sensory input can drive a spiking network.
- We use recurrent self-excitatory connections to create self-sustained activation in a neuron or neuronal population: an active neuron continues spiking to represent the current estimate of the pose in the SNN, even in the absence of input. This models the working memory of biological neural systems.
- We propose connectivity patterns between neuronal populations that solve different computational tasks:
  - *mapping* activity from one population to another one in a one-to-one or shifted manner;
  - *resetting* activity by inhibiting the currently active neuron(s) and activating another/others;
  - *boosting* the whole population through one-to-all connections;
  - providing subthreshold localized input (*preshape*) to create a potentiality for activation, i.e., when boosted, such a preshape can lead to fully-fledged activation.
- We demonstrate the integration of different modalities: input from one modality (motor command) is *integrated* into the network to produce the pose, and input from another modality (vision) is mapped onto the network through *plastic*—learned—synapses.
- Finally, we demonstrate one-shot online learning of the object-centering pose and how it can be used to generate object-directed gaze. To use the learned pose, we introduced an additional layer that can read out the learned pose without triggering a reset. Thus, we explicitly distinguish the "remembered" and the "currently perceived" object-centering pose representation, modeling different "directions of fit" from the theory of intentionality (Searle and Willis, 1983).

These elements form the basic algorithmic building blocks for pose estimation and SLAM-like systems in neuromorphic technologies. In this work, we realize the SNN to estimate the pose of a robot's head within two degrees of freedom (yaw-pitch). The error of the head-direction network compared to the integrated velocity commands in software remains below "one neuron" (i.e., an angle corresponding to V_{thr} from Equation 1). Plastic synaptic connections between the yaw-pitch motor space and visually-activated object-neurons in our SNN are learned to store the positions of objects autonomously during operation, i.e., showing online learning. These connections can be used to produce goal-directed head-movements toward stored poses, "looking back" at objects. The stored associations are also used to correct the pose, as the path integration process may be subject to drift (as shown in **Figure 6**).

This work also highlights the interfaces that we developed between the iCub robot and the Loihi chip. System integration is an important challenge in robotics in general and in neuromorphic robotics in particular. Our solution is still in a prototype stage but already achieves real-time performance (processing loop of $<$10 ms). Tighter integration of the hardware system will further improve the system's latency. When combined with a more powerful object recognition system, our pose estimation and learning SNN can be used as a component of an interactive scene representation system for robotic and augmented reality applications.

DATA AVAILABILITY STATEMENT

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

AUTHOR CONTRIBUTIONS

RK, AR, and VL have worked on the SNN on chip, run the simulated experiments, and worked on SNN-related plots and results. BS helped with the initial experiments and writing. CB supervised the work related to iCub robot and event-based vision and helped with writing. AG planned and performed the experiments with the robot, developed the software infrastructure required to connect the robot and neuromorphic hardware, worked on the plots and text related to robot experiments. YS conceptualized the study, suggested the experiments, supervised all work, and put it in the context of SoA.

FUNDING

This work has started as a project at CapoCaccia 2019 Cognitive Neuromorphic Engineering Workshop and

was funded by the SNSF grant Ambizione ELMA (Grant No. PZOOP2_168183), EU ERC grant NeuroAgents (Grant No. 724295), Forschungskredit of the University of Zurich (Grant No. FK-76204-03-01), and EU MSCA (Grant No. 676063).

ACKNOWLEDGMENTS

We would like to thank Intel Neuromorphic Computing Lab for their continual support with the neuromorphic research chip Loihi.

REFERENCES

Alzugaray, I., and Chli, M. (2018). Asynchronous corner detection and tracking for event cameras in real-time. *IEEE Robot. Autom. Lett.* 3, 3177–3184. doi: 10.1109/LRA.2018.2849882

Arena, P., Maceo, S., Patane, L., and Strauss, R. (2013). "A spiking network for spatial memory formation: towards a fly-inspired ellipsoid body model," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX: IEEE), 1–6. doi: 10.1109/IJCNN.2013.6706882

Arleo, A., Gerstner, W. (2000). Spatial cognition and neuro-mimetic navigation: a model of hippocampal place cell activity. *Biol Cybern.* 83, 287–299. doi: 10.1007/s004220000171

Barrera, A., and Weitzenfeld, A. (2008). Biologically-inspired robot spatial cognition based on rat neurophysiological studies. *Auton. Robots* 25, 147–169. doi: 10.1007/s10514-007-9074-3

Bartolozzi, C., Rea, F., Clercq, C., Fasnacht, D. B., Indiveri, G., Hofstätter, M., et al. (2011). "Embedded neuromorphic vision for humanoid robots," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (Colorado Springs, CO). doi: 10.1109/CVPRW.2011.5981834

Bell, C., Storck, T., and Sandamirskaya, Y. (2014). "Learning to look: a dynamic neural fields architecture for gaze shift generation," in *ICANN* (Hamburg), 699–706. doi: 10.1007/978-3-319-11179-7_88

Burak, Y., and Fiete, I. R. (2009). Accurate path integration in continuous attractor network models of grid cells. *PLoS Comput. Biol.* 5:e1000291. doi: 10.1371/journal.pcbi.1000291

Chen, C., Seff, A., Kornhauser, A., and Xiao, J. (2015). "DeepDriving: learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)* (Santiago). doi: 10.1109/ICCV.2015.312

Conklin, J., and Eliasmith, C. (2005). A controlled attractor network model of path integration in the rat. *J. Comput. Neurosci.* 18, 183–203. doi: 10.1007/s10827-005-6558-z

Corradi, F., Zambrano, D., Raglianti, M., Passetti, G., Laschi, C., and Indiveri, G. (2014). Towards a neuromorphic vestibular system. *IEEE Trans. Biomed. Circuits Syst.* 8, 669–680. doi: 10.1109/TBCAS.2014.2358493

Cuperlier, N., Quoy, M., and Gaussier, P. (2007). Neurobiologically inspired mobile robot navigation and planning. *Front. Neurorobot.* 1:3. doi: 10.3389/neuro.12.003.2007

Dalgaty, T., Vianello, E., De Salvo, B., and Casas, J. (2018). Insect-inspired neuromorphic computing. *Curr. Opin. Insect Sci.* 30, 59–66. doi: 10.1016/j.cois.2018.09.006

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, 82–89. doi: 10.1109/MM.2018.112130359

Edvardsen, V. (2017). "Long-range navigation by path integration and decoding of grid cells in a neural network," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK: IEEE), 4348–4355.

Engelhard, N., Endres, F., Hess, J., Sturm, J., and Burgard, W. (2011). "Real-time 3D visual slam with a hand-held RGB-D camera," in *Proceedings of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum* (Vasteras), Vol. 180, 1–15.

Erdem, U. M., Milford, M. J., and Hasselmo, M. E. (2015). A hierarchical model of goal directed navigation selects trajectories in a visual environment. *Neurobiol. Learn. Mem.* 117, 109–121. doi: 10.1016/j.nlm.2014.07.003

Fisher, Y. E., Lu, J., D'Alessandro, I., and Wilson, R. I. (2019). Sensorimotor experience remaps visual input to a heading-direction network. *Nature* 576, 121–125. doi: 10.1038/s41586-019-1772-4

Furber, S. B., Lester, D. R., Plana, L. A., Garside, J. D., Painkras, E., Temple, S., et al. (2012). Overview of the SpiNNaker system architecture. *IEEE Trans. Comput.* 62, 2454–2467. doi: 10.1109/TC.2012.142

Gallego, G., Delbruck, T., Orchard, G., Bartolozzi, C., Taba, B., Censi, A., et al. (2019). Event-based vision: a survey. *arxiv [Preprint] arXiv* 1904.08405.

Galluppi, F., Lagorce, X., Stromatias, E., Pfeiffer, M., Plana, L. A., Furber, S. B., et al. (2015). A framework for plasticity implementation on the spinnaker neural architecture. *Front. Neurosci.* 8:429. doi: 10.3389/fnins.2014.00429

Gehrig, M., Shrestha, S. B., Mouritzen, D., and Scaramuzza, D. (2020). Event-based angular velocity regression with spiking networks. *arXiv* 2003.02790.

Goodridge, J. P., and Touretzky, D. S. (2000). Modeling attractor deformation in the rodent head-direction system. *J. Neurophysiol.* 83, 3402–3410. doi: 10.1152/jn.2000.83.6.3402

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. doi: 10.1038/nature22343

Grossberg, S. (1988). Nonlinear neural networks: principles, mechanisms, and architectures. *Neural Netw.* 1, 17–61.

Gu, T., and Yan, R. (2019). "An improved loop closure detection for RatSLAM," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)* (Beijing: IEEE), 884–888.

Hahnloser, R. H. (2003). Emergence of neural integration in the head-direction system by visual supervision. *Neuroscience* 120, 877–891. doi: 10.1016/S0306-4522(03)00201-X

Heinze, S. (2017). Unraveling the neural basis of insect navigation. *Curr. Opin. Insect Sci.* 24, 58–67. doi: 10.1016/j.cois.2017.09.001

Heinze, S., Narendra, A., and Cheung, A. (2018). Principles of insect path integration. *Curr. Biol.* 28, R1043–R1058. doi: 10.1016/j.cub.2018.04.058

Honkanen, A., Adden, A., Da Silva Freitas, J., and Heinze, S. (2019). The insect central complex and the neural basis of navigational strategies. *J. Exp. Biol.* 222:jeb188854. doi: 10.1242/jeb.188854

Indiveri, G., Chicca, E., and Douglas, R. J. (2009). Artificial cognitive systems: from VLSI networks of spiking neurons to neuromorphic cognition. *Cogn. Comput.* 1, 119–127. doi: 10.1007/s12559-008-9003-6

Jauffret, A., Cuperlier, N., and Gaussier, P. (2015). From grid cells and visual place cells to multimodal place cell: a new robotic architecture. *Front. Neurorobot.* 9:1. doi: 10.3389/fnbot.2015.00001

Kreiser, R., Aathmani, D., Qiao, N., Indiveri, G., and Sandamirskaya, Y. (2018a). Organizing sequential memory in a neuromorphic device using dynamic neural fields. *Front. Neurosci.* 12:717. doi: 10.3389/fnins.2018.00717

Kreiser, R., Cartiglia, M., Martel, J. N., Conradt, J., and Sandamirskaya, Y. (2018b). "A neuromorphic approach to path integration: a head-direction spiking neural network with vision-driven reset," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)* (Florence). doi: 10.1109/ISCAS.2018.8351509

Kreiser, R., Pienroj, P., Renner, A., and Sandamirskaya, Y. (2018c). "Pose estimation and map formation with spiking neural networks: towards neuromorphic SLAM," in *IEEE/RSJ International Conference on Intelligent Robots and Systems* (Madrid). doi: 10.1109/IROS.2018.8594228

Kreiser, R., Renner, A., and Sandamirskaya, Y. (2019a). "Error-driven learning for self-calibration in a neuromorphic path integration system," in *Robust AI for Neurorbotics* (Edinburgh).

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

Krichmar, J. L., and Wagatsuma, H. (2011). *Neuromorphic and Brain-Based Robots*, Vol. 233. Cambridge: Cambridge University Press.

Liang, D., Kreiser, R., Nielsen, C., Qiao, N., Sandamirskaya, Y., and Indiveri, G. (2019). Neural state machines for robust learning and control of

neuromorphic agents. *IEEE J. Emerg. Select. Top. Circuits Syst*. 9, 679–689. doi: 10.1109/JETCAS.2019.2951442

Massoud, T. M., and Horiuchi, T. K. (2011a). A neuromorphic VLSI head direction cell system. *IEEE Trans. Circuits Syst. I Reg. Pap*. 58, 150–163. doi: 10.1109/TCSI.2010.2055310

Massoud, T. M., and Horiuchi, T. K. (2011b). "Online correction of orientation estimates using spatial memory in a neuromorphic head direction system," in *Proceedings–IEEE International Symposium on Circuits and Systems* (Rio de Janeiro). doi: 10.1109/ISCAS.2011.5938094

Massoud, T. M., and Horiuchi, T. K. (2012). "A neuromorphic VLSI grid cell system," in *ISCAS 2012–2012 IEEE International Symposium on Circuits and Systems*. (Seoul). doi: 10.1109/ISCAS.2012.6271787

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

Metta, G., Fitzpatrick, P., and Natale, L. (2006). YARP: yet another robot platform. *Int. J. Adv. Robot. Syst*. 3, 43–48. doi: 10.5772/5761

Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). "The iCub humanoid robot: An open platform for research in embodied cognition," in *Performance Metrics for Intelligent Systems (PerMIS) Workshop* (Gaithersburg, MD), 50–56. doi: 10.1145/1774674.1774683

Milford, M., and Schulz, R. (2014). Principles of goal-directed spatial robot navigation in biomimetic models. *Philos. Trans. R. Soc. B Biol. Sci*. 369, 1–13. doi: 10.1098/rstb.2013.0484

Milford, M. J., Wyeth, G. F., and Prasser, D. (2004). "RatSLAM: a hippocampal model for simultaneous localization and mapping," in *Proceeding of the 2004 IEEE international Conference on Robotics & Automation* (New Orleans, LA), 403–408. doi: 10.1109/ROBOT.2004.1307183

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., a., Veness, J., et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518, 529–533. doi: 10.1038/nature14236

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

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *IEEE Signal Process. Mag.* 36, 51–63. doi: 10.1109/MSP.2019.2931595

Pfeiffer, K., and Homberg, U. (2014). Organization and functional roles of the central complex in the insect brain. *Annu. Rev. Entomol.* 59, 165–184. doi: 10.1146/annurev-ento-011613-162031

Posch, C., Matolin, D., and Wohlgenannt, R. (2008). "An asynchronous time-based image sensor," in *2008 IEEE International Symposium on Circuits and Systems* (Seattle, WA), 2130–2133. doi: 10.1109/ISCAS.2008.4541871

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*. 9:141. doi: 10.3389/fnins.2015.00141

Redish, A., Elga, A., and Touretzky, D. (1996). A coupled attractor model of the rodent head direction system. *Netw. Comput. Neural Syst.* 7, 671–685. doi: 10.1088/0954-898x/7/4/004

Samu, D., Erős, P., Ujfalussy, B., and Kiss, T. (2009). Robust path integration in the entorhinal grid cell system with hippocampal feed-back. *Biol. Cybern*. 101, 19–34. doi: 10.1007/s00422-009-0311-z

Sandamirskaya, Y., and Storck, T. (2014). "Neural-dynamic architecture for looking: shift from visual to motor target representation for memory saccade," in *ICDL-EPIROB* (Genoa). doi: 10.1109/DEVLRN.2014.6982951

Sandamirskaya, Y., and Storck, T. (2015). "Chapter: learning to look and looking to remember: a neural-dynamic embodied model for generation of saccadic gaze shifts and memory formation," in *Artificial Neural Network, Vol. 4*, eds P. Koprinkova-Hristova, V. Mladenov, and N. K. Kasabov (Springer), 175–200. doi: 10.1007/978-3-319-09903-3_9

Scaramuzza, D., and Fraundorfer, F. (2011). Visual odometry. *IEEE Robot. Autom. Mag.* 18, 80–92. doi: 10.1109/MRA.2011.943233

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., and Millner, S. (2010). "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *ISCAS 2010–2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems* (Paris), 1947–1950. doi: 10.1109/ISCAS.2010.5536970

Searle, J. R., and Willis, S. (1983). *Intentionality: An Essay in the Philosophy of Mind*. Cambridge: Cambridge University Press.

Seelig, J. D., and Jayaraman, V. (2013). Feature detection and orientation tuning in the Drosophila central complex. *Nature* 503, 262–266. doi: 10.1038/nature12601

Seelig, J. D., and Jayaraman, V. (2015). Neural dynamics for landmark orientation and angular path integration. *Nature* 521, 186–191. doi: 10.1038/nature14446

Shrestha, S. B., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems* (Montreal, QC), 1412–1421.

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," in *Advances in Neural Information Processing Systems* (Denver, CO), 173–180.

Song, P., and Wang, X. J. (2005). Angular path integration by moving "hill of activity": a spiking neuron model without recurrent excitation of the head-direction system. *J. Neurosci.* 25, 1002–1014. doi: 10.1523/JNEUROSCI.4172-04.2005

Stachniss, C., Leonard, J. J., and Thrun, S. (2016). "Simultaneous localization and mapping," in *Springer Handbook of Robotics,* eds B. Siciliano and O. Khatib (Springer), 1153–1176. doi: 10.1007/978-3-319-32552-1_46

Stratton, P., Wyeth, G., and Wiles, J. (2010). Calibration of the head direction network: a role for symmetric angular head velocity cells. *J. Comput. Neurosci.* 28, 527–538. doi: 10.1007/s10827-010-0234-7

Stringer, S. M., Trappenberg, T. P., Rolls, E. T., and De Araujo, I. E. (2002). Self-organizing continuous attractor networks and path integration: one-dimensional models of head direction cells. *Netw. Comput. Neural Syst.*13, 217–242. doi: 10.1088/0954-898X/13/2/304

Tang, G., Shah, A., and Michmizos, K. P. (2019). Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. *arXiv* 1903.02504. doi: 10.1109/IROS40897.2019.8967864

Taube, J. S. (2007). The head direction signal: origins and sensory-motor integration. *Annu. Rev. Neurosci.* 30, 181–207. doi: 10.1146/annurev.neuro.29.051605.112854

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., et al. (2007). "Stanley: the robot that won the DARPA grand challenge," in *Springer Tracts in Advanced Robotics*, eds M. Buehler, K. Iagnemma, and S. Singh (Springer), 661–692. doi: 10.1007/978-3-540-73429-1_1

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:e23496. doi: 10.7554/eLife.23496

Turner-Evans, D. B., and Jayaraman, V. (2016). The insect central complex. *Curr. Biol.* 26, R445–R460. doi: 10.1016/j.cub.2016.04.006

Weikersdorfer, D., Hoffmann, R., and Conradt, J. (2013). "Simultaneous localization and mapping for event-based vision systems," in *International Conference on Computer Vision Systems* (Thessaloniki: Springer), 133–142. doi: 10.1007/978-3-642-39402-7_14

Yu, F., Shang, J., Hu, Y., and Milford, M. (2019). NeuroSLAM: a brain-inspired SLAM system for 3D environments. *Biol. Cybern.* 113, 515–545. doi: 10.1007/s00422-019-00806-9

Zhang, H., Tang, H., and Yan, R. (2019). "Multi-sensor fusion for a brain-inspired SLAM system," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)* (Beijing: IEEE), 619–623.

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 Neuroscience

ORIGINAL RESEARCH
published: 30 June 2020
doi: 10.3389/fnins.2020.00662

Optimizing the Energy Consumption of Spiking Neural Networks for Neuromorphic Applications

Martino Sorbaro[1,2], *Qian Liu*[1], *Massimo Bortone*[1] *and Sadique Sheik*[1]*

[1] SynSense (formerly aiCTX), Zurich, Switzerland, [2] Institute of Neuroinformatics, University of Zürich and ETH Zürich, Zurich, Switzerland

In the last few years, spiking neural networks (SNNs) have been demonstrated to perform on par with regular convolutional neural networks. Several works have proposed methods to convert a pre-trained CNN to a Spiking CNN without a significant sacrifice of performance. We demonstrate first that quantization-aware training of CNNs leads to better accuracy in SNNs. One of the benefits of converting CNNs to spiking CNNs is to leverage the sparse computation of SNNs and consequently perform equivalent computation at a lower energy consumption. Here we propose an optimization strategy to train efficient spiking networks with lower energy consumption, while maintaining similar accuracy levels. We demonstrate results on the MNIST-DVS and CIFAR-10 datasets.

Keywords: neuromorphic computing, spiking networks, loss function, synaptic operations, energy consumption, convolutional networks, CIFAR10, MNIST-DVS

OPEN ACCESS

Edited by:
Chiara Bartolozzi,
Italian Institute of Technology (IIT), Italy

Reviewed by:
Jonathan Binas,
Montreal Institute for Learning Algorithm (MILA), Canada
Roshan Gopalakrishnan,
*Institute for Infocomm Research (A*STAR), Singapore*

***Correspondence:**
Sadique Sheik
sadique.sheik@synsense.ai

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *02 December 2019*
Accepted: *28 May 2020*
Published: *30 June 2020*

Citation:
Sorbaro M, Liu Q, Bortone M and Sheik S (2020) Optimizing the Energy Consumption of Spiking Neural Networks for Neuromorphic Applications. Front. Neurosci. 14:662. doi: 10.3389/fnins.2020.00662

1. INTRODUCTION

Since the early 2010s, computer vision has been dominated by the introduction of convolutional neural networks (CNNs), which have yielded unprecedented success in previously challenging tasks such as image recognition, image segmentation or object detection, among others. Considering the theory of neural networks was mostly developed decades earlier, one of the main driving factors behind this evolution was the widespread availability of high-performance computing devices and general purpose Graphic Processing Units (GPU). In parallel with the increase in computational requirements (Strubell et al., 2019), the last decades have seen a considerable development of portable, miniaturized, battery-powered devices, which pose constraints on the maximum power consumption.

Attempts at reducing the power consumption of traditional deep learning models have been made. Typically, these involve optimizing the network architecture, in order to find more compact networks (with fewer layers, or fewer neurons per layer) that perform equally well as larger networks. One approach is energy-aware pruning, where connections are removed according to a criterion based on energy consumption, and accuracy is restored by fine-tuning of the remaining weights (Molchanov et al., 2016; Yang et al., 2017). Other work looks for more efficient network structures through a full-fledged architecture search (Cai et al., 2018). The latter work was one of the winners of the Google "Visual Wake Words Challenge" at CVPR 2019, which sought models with memory usage under 250 kB, model size under 250 kB and per-inference multiply-add count (MAC) under 60 millions.

Using spiking neural networks (SNNs) on neuromorphic hardware is an entirely different, and much more radical, approach to the energy consumption problem. In SNNs, like in biological

neural networks, neurons communicate with each other through isolated, discrete electrical signals (spikes), as opposed to continuous signals, and work in continuous instead of discrete time. Neuromorphic hardware (Indiveri et al., 2011; Esser et al., 2016; Furber, 2016; Thakur et al., 2018) is specifically designed to run such networks with very low power overhead, with electronic circuits that faithfully reproduce the dynamics of the model in real time, rather than simulating it on traditional von Neumann computers. Some of these architectures (including Intel's Loihi, IBM's TrueNorth, and SynSense's DynapCNN) support convolution operations, which are necessary for modern computer vision techniques, by an appropriate weight sharing mechanism.

The challenge of using SNNs for machine learning tasks, however, is in their training. Mimicking the learning process used in the brain's spiking networks is not yet feasible, because neither the learning rules, nor the precise fitness functions being optimized are sufficiently well-understood, although this is currently a very active area of research (Marblestone et al., 2016; Richards et al., 2019). Supervised learning routines for spiking networks have been developed (Bohte et al., 2002; Mostafa, 2017; Nicola and Clopath, 2017; Shrestha and Orchard, 2018; Neftci et al., 2019), but are slow and challenging to use. For applications which have little or no dependence on temporal aspects, it is more efficient to train an analog network (i.e., a traditional, non-spiking one) with the same structure, and transfer the learned parameters onto the SNN, which can then operate through rate coding. In particular, the conversion of pre-trained CNNs to SNNs has been shown to be a scalable and reliable process, without much loss in performance (Diehl et al., 2015; Rueckauer et al., 2017; Sengupta et al., 2019). But this approach is still challenging, because the naive use of analog CNN weights does not take into account the specific characteristics and requirements of SNNs. In particular, SNNs are more sensitive than analog networks to the magnitude of the input. Naive weight transfer can, therefore, lead to a silent SNN, or, conversely, to one with unnecessarily high firing rates, which have a high energy cost.

Here, we propose a hybrid training strategy which maintains the efficiency of training analog CNNs, while accounting for the fact that the network is being trained for eventual use in SNNs. Furthermore, we include the energy cost of the network's computations directly in the loss function during training, in order to minimize it automatically and dynamically. We demonstrate that networks trained with this strategy perform better per Joule of energy utilized. While we demonstrate the benefit of optimizing based on energy consumption, we believe this strategy is extendable to any approach that uses back-propagation to train the network, be it through a spiking network or a non-spiking network.

In the following sections, we will detail the training techniques we devised and applied for these purposes. We will test our networks on two standard problems. The first is the MNIST-DVS dataset of Dynamic Vision Sensor recordings. DVSs are event-based sensors, and, as such, the analysis of their recordings is an ideal application of spike-based neural networks. The second is the standard CIFAR-10 object recognition benchmark, which provides a reasonable comparison on computation cost to non-spiking networks. For each of these tasks, we will demonstrate the energy-accuracy trade-off of the networks trained with our methods. We show that significant amounts of energy can be saved with a small loss in performance, and conclude that ours is a viable strategy for training neuromorphic systems with a limited power budget.

2. MATERIALS AND METHODS

In most state-of-the art neuromorphic architectures with time multiplexed units like Merolla et al. (2014), Davies et al. (2018), and Furber et al. (2014), the various states need to be fetched from memory and rewritten. Such operations happen every time a neuron receives a synaptic event. Whenever one of these operations is performed, the neuromorphic hardware consumes a certain amount of energy. For instance in Indiveri and Sandamirskaya (2019) the authors show that this energy consumption is usually of the order of 10^{-11} J. While there are several other processes that consume power on a neuromorphic device, the bulk of the active power on these devices is used by the synaptic operations. Reducing their number is therefore the most natural way to keep energy usage low.

In this paper we explore strategies to lower synaptic operations and evaluate their effect on the network's computational performance. We suggest to train, or fine-tune, networks with an additional loss term which explicitly enforces lower activations in the trained network—and consequently lower firing rates of the corresponding spiking network. This is analogous to the L_1 term used by Esser et al. (2016) and Neil et al. (2016), but applied on synaptic operations directly rather than firing rates, and set up so that a target SynOp count value can be set. Additionally, we introduce quantization of the activations on each layer, which mimics the discretization effect of spiking networks, so that the network activity remains at reasonable levels even when the regularization term is strong. The following sections illustrate the technical details and introduce the datasets and networks we use for evaluation.

2.1. Training Strategies

2.1.1. Parameter Scaling

By scaling the weights, biases and/or thresholds of neurons in different layers, we can influence the number of spikes generated in each layer, thereby allowing us to tune the synaptic activity of the model. This is easy to do, even with pre-trained weights. For a scale-invariant network, such as any network whose only non-linearities are ReLUs, this method attains perfect results, because a linear rescaling of the weights causes a linear rescaling of the output, which gives identical results for classification tasks where we select the class that receives the highest activation.

We use this method as a baseline comparison for our results. We chose to rescale the weights of the first convolutional layer of our network by a variable factor ρ:

$$w_0' = \rho w_0,$$

which is equivalent to a rescaling of the input signal by the same factor. Note that an increase/decrease in the first layer's output firing rate causes a correspondent increase/decrease in the activation of all the subsequent layers, and thereby reduces the global energy consumption of the whole network.

For baseline comparisons, we also apply the "robust" weight scaling suggested by Rueckauer et al. (2017). This consists of a per-layer scaling of weights, in such a way that the maximum level of activation is constant along the network. For robustness, the 99th percentile of activations is taken as a measure of output magnitude in each layer, estimated from forward passes over 25,600 samples of the training set. In this way, the activity of the network is balanced over its layers, in the sense that no layer unnecessarily amplifies or reduces the activity level compared to its input.

The scale-invariance property of ReLU functions does not hold for the corresponding spiking network, and small activation values could cause discretization errors, or even yield a completely silent spiking network from a perfectly functional analog network.

2.1.2. Synaptic Operation Optimization

We measure the activity of the network, for each layer group, in correspondence with the ReLU operations, which effectively correspond to the spikes from an equivalent SNN (**Supplementary Figure 2**). We denote the activity of neuron i in layer ℓ as a_i^ℓ. We define the *fan-out* of each group of layers, f_{out}^ℓ, as the number of units of layer $\ell + 1$ that receive the signal emitted by a single neuron in layer ℓ. This measure is essential in estimating the number of synaptic operations (SynOps) s^ℓ elicited by each layer:

$$s^\ell = f_{\text{out}}^\ell \sum_i a_i^\ell \tag{1}$$

We directly add this number to the loss we want to minimize, optionally specifying a target value S_0 for the desired number of SynOps:

$$\mathcal{L} = \mathcal{C}(a^{\text{output}}, t) + \alpha \left(S_0 - \sum_\ell s^\ell \right)^2 \tag{2}$$

where $\mathcal{C}$ is the cross-entropy loss, t is the target label, and α is a constant. We will refer to this additional term as *SynOp loss*. In this work, we will always choose $\alpha = 1/S_0^2$, in order to keep the SynOp loss term normalized independently of S_0. Although setting $S_0 = 0$ and tweaking the value of α instead is also a valid choice, we found it easier to set a direct target for the power budget, which leads to more predictable results.

Additionally, we performed some experiments where an L_1 penalty on activations was used, without fanout-based weighting. Against our expectations, we did not find a significant difference in power consumption between the models trained with or without per-layer weighting (**Supplementary Figure 1**). However, we use the fanout-based penalty throughout this paper, since this addresses the power consumption more directly, and we cannot rule out that this difference may be more significant in larger networks.

2.1.3. Quantization-Aware Training and Surrogate Gradient

Optimizing for energy consumption with the SynOp loss mentioned above has unintended consequences. During training, the optimizer tries to achieve smaller activations, but cannot account for the fact that, when the activations are too small, discretization errors become more prominent. Throughout this paper, by discretization error we mean the discrepancy that occurs when a real number needs to be represented in a discrete way—namely, the value of each neuron's activation, which is continuous, needs to be translated in a finite number of spikes, leading to inevitable approximations. To solve this issue, we introduce a form of quantization during training. The quantization of activations mimics, in the context of an analog network, a form of discretization analogous to what happens in a spiking network. Therefore, the network can be already aware of the discretization error at training time, and automatically adjust its parameters in order to properly account for it. To this end, we turn all ReLU activation functions into "quantized" (i.e., step-wise) ReLUs, which additionally truncate the inputs to integers, as follows:

$$\text{QReLU}(x) = \begin{cases} 0 & x \leq 0 \\ \lfloor x \rfloor & x > 0 \end{cases} \tag{3}$$

where $\lfloor \cdot \rfloor$ indicates the floor operation. This choice introduces a further problem: this function is discontinuous, and its derivative is uniformly zero wherever it is defined. To avoid the zeroing of gradients during the backward pass, we use a *surrogate gradient* method (Neftci et al., 2019), whereby the gradient of QReLU is approximated with the gradient of a normal ReLU during the backward pass:

$$\nabla_x \, \text{QReLU}(x) \approx \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{4}$$

This is not the only way to approximate the gradient of a step-wise function in a meaningful way, and closer approximations are certainly possible; however, we found that this linear approximation works sufficiently well for our purposes.

In this work, we apply QReLUs in combination with the SynOp loss term illustrated in the previous section, but quantization on activations could be independently used for a more accurate training of spiking networks. We note that quantization-aware training in different forms has been used before (Hubara et al., 2017; Guo, 2018), but its typical purpose is to sharply decrease the memory consumption of CNNs, by storing both activations and weights as lower-precision numbers (e.g., as int8 instead of the typical float32). PyTorch recently started providing support utilities for this purpose[1].

2.2. Spiking Network Simulations With Sinabs

After training, we tested our trained weights on spiking network simulations. Unlike tests done on analog networks, these are

[1]https://pytorch.org/docs/stable/quantization.html

time-dependent simulations, which fully account for the time dynamics of the input spike trains, and closely mimic the behavior of a neuromorphic hardware implementation, like DynapCNN (Liu et al., 2019). Our simulations are written using the Sinabs Python library[2], which uses non-leaky integrate-and-fire neurons with a linear response function. The sub-threshold neuron dynamics of the non-leaky integrate and fire neurons are described as follows:

$$\dot{v} = R \cdot \left(I_{syn}(t) + I_{bias}\right) \quad (5)$$

$$I_{syn}(t) = WS(t) \quad (6)$$

where v is the membrane potential of the neuron, R is a constant, I_{syn} is the synaptic input current, I_{bias} is a constant input current term, W is the synaptic weight matrix and $S(t)$ is a vector of input spike trains. For the results presented in this paper, we assume $R = 1$ without any loss of generality. Upon reaching a spiking threshold vth the neuron's membrane potential is reduced by a value v th (not reset to zero).

As a result of the above, between times t and $t + \delta t$, for a total input current $I(t) = I_{syn}(t) + I_{bias}$, the neurons generate a number of spikes $n(t, t + \delta t)$ given by the following equation:

$$n(t, t + \delta t) = \left\lfloor \frac{R \cdot I(t) \cdot \delta t}{v_{th}} \right\rfloor. \quad (7)$$

In order to simulate the equivalent SNN model on Sinabs, the CNN's pre-trained weights are directly transferred to the equivalent SNN.

2.3. Digit Recognition on DVS Recordings

2.3.1. Task and Dataset

As a benchmark to assess the performance of the above training methods, we used an image recognition task on real data recorded by a Dynamic Vision Sensor (DVS). Given a spike train generated by the DVS, our spiking networks identify the class to which the object belongs—corresponding to the fastest-firing neuron in the output layer. For this task, we used the MNIST-DVS dataset at scale 16 (Serrano-Gotarredona and Linares-Barranco, 2015; Liu et al., 2016), a collection of DVS recordings where digits from the classic MNIST dataset (LeCun et al., 1998) are shown to the DVS camera as they move on a screen.

During the training phase, we presented the (analog) neural network with images formed of accumulated DVS events, i.e., DVS spike trains divided into chunks and collapsed along the time dimension. The value of each pixel (0-255 in the image encoding we chose) was determined simply by the number of events on that pixel. The DVS recordings were split into frames not based on time length, but according to event count: the accumulation of each frame was stopped when the total number of events per frame reached a value of 3,000 (**Figures 1A,B**). This ensured all frames had comparable pixel values without the need for normalization, and all contained similar amounts of information regardless of the type of activity presented. The information regarding event polarity was discarded, resulting in a 1-channel input frame (analogous to gray-scale image).

[2]https://sinabs.ai

During testing on the spiking network simulation, the corresponding spike trains were presented to the network with 1 ms time resolution (**Figure 1C**), to simulate the real-time event transmission between the DVS and a neuromorphic chip. This value was chosen to enable reasonable simulation times, but could be lowered if needed. **Figure 1C**, to simulate the real-time event transmission between the DVS and a neuromorphic chip. The network state was reset between the presentation of a data chunk and the next. The polarity of events was ignored. Of the original 10,000 recordings (1,000 per digit from zero to nine), we set 20% aside as test set.

2.3.2. Network Architecture

In order to solve the task mentioned above, we used a simple convolutional neural network, with three 2D convolutional layers (3×3 filters), each followed by an average pooling layer (2×2 filters) and a rectified linear unit. The choice of average pooling is due to the difficulties of implementing max pooling in spiking networks (Rueckauer et al., 2017). The last layer is a linear (fully connected) layer, which outputs the class predictions (**Figure 1D**). We used a cross-entropy loss function to evaluate the model predictions and optimized the network weights using the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3}. Bias parameters were deactivated everywhere in the network. A 50% dropout was used just before the fully connected layer at training time. The network was implemented using PyTorch (Paszke et al., 2017).

The whole procedure can be summarized as follows:

1. The dataset is prepared by dividing the original DVS recordings in sections of 3,000 spikes each, ignoring event polarity. From these the following is saved:
   (a) The spike train itself, used for testing
   (b) An image, corresponding to the time-collapsed spike train, with pixel values equal to the number of spikes at that location, used for training.
2. A neural network is trained, applying quantization in correspondence with every ReLU. The loss used for training is binary cross-entropy with the addition of the synoploss term (Equation 2).
3. The trained weights are transferred to a spiking network simulation, implemented in Sinabs. The network dynamics is simulated with 1 ms time resolution. The network prediction is defined as the neuron that spikes the most over the 3,000-spike input. Synaptic operations are counted as the sum of spikes emitted by each layer, weighted on the fan-out of that layer.

For reproducibility, the python code implementing these methods is available at gitlab.com/aiCTX/synoploss.

2.4. Object Recognition on CIFAR-10

2.4.1. Task and Dataset

In order to validate the approach on a dataset with higher complexity than MNIST, we also benchmarked our work on CIFAR-10 (Krizhevsky et al., 2009), a visual object classification task. The input images were augmented with random crop and horizontal flip, and then normalized to $[-1, 1]$. A 20% dropout

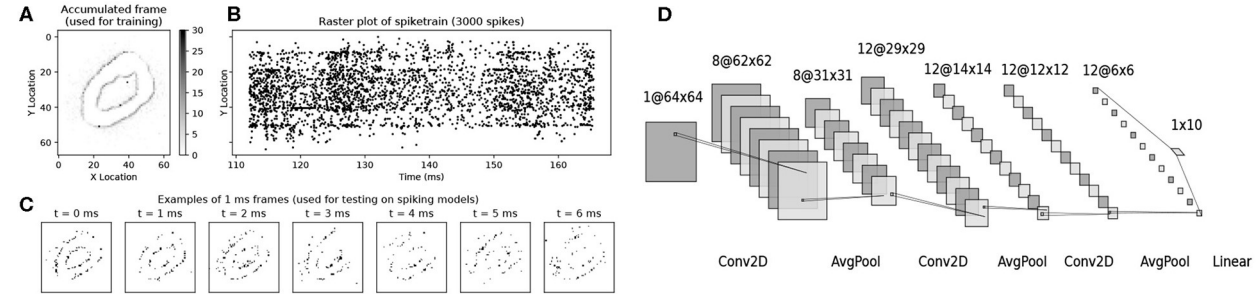


FIGURE 1 | Illustration of the MNIST-DVS dataset, as used in this work, and of the network model we used for the task. **(A)** A single accumulated frame of 3,000 spikes, as used for training. **(B)** the corresponding 3,000 spikes, in location and time. **(C)** Example single-millisecond frames, as sequentially shown to the Sinabs spiking network during the tests presented in **Figure 2**. **(D)** The convolutional network model we used for this task. All convolutional layers and the linear layer are followed by ReLUs. Dropout is used before the linear layer at training time.

rate was applied to the input layer to further augment the input data.

For the experimental results on this dataset, we directly injected the image pixel analog value to the first layer of convolutions as input current in each simulation time step for N_{dt} time steps. The magnitude of the current was scaled down by the same value N_{dt}, in order to have an accumulated current, over the whole simulation, equal to the analog input value. The Sinabs simulations were run for $N_{dt} = 10$ time steps, obtaining SynOps and accuracy values. The network state was reset between the presentation of an image and the next.

2.4.2. Network Architecture and Training Procedure

In order to solve the task mentioned above, we used an All-ConvNet (Springenberg et al., 2014), a 9-layer convolutional network, without bias terms, which has 1.9M parameters in total. The ReLU layers in the model, including the last output layer, were replaced with QReLUs. All the convolutional layers in this network are followed by a dropout layer with a rate of 10%, which not only prevents over-fitting, but also compensates the SNN's discrete representations of analog values. As illustrated in Springenberg et al. (2014), training lasts 350 epochs, and the learning rate is initialized at 2.5×10^{-4} and scaled down by a factor of 10 at epochs [200, 250, 300]. We use the Adam optimizer with weight decay of 10^{-3}. Note that the model was trained without ReLU on the last output layer, since it is harder to train the classification layer when the outputs are only positive, while the classification accuracy was tested with ReLU on the output layer, in order to have an equivalent network to the spiking model.

The entire experiment is as follows:

1. Train an ANN network, *anet*, get its MAC and test the accuracy with original CIFAR10 dataset.
2. Scale up the weights of the first layer of *anet* by ρ, and transfer the weights to the SNN equivalence, *snet*.
   (a) Test the accuracy and SynOps of *snet* with $N_{dt} = 10$, the input current is 1/10 of a pixel value.
   (b) Increase ρ, and repeat 2(a) till the accuracy reached about the ANN accuracy.
3. Select a ρ from Step 2, where the *snet* have SynOps > MAC, and start quantization-aware training with the SynOp loss.
   (a) Set the target SynOp to half of the current SynOps and train.
   (b) Test the accuracy and get the SynOps, then repeat 3(a) until the accuracy is too low to be meaningful, and thus a full accuracy/SynOps curve is obtained.

2.4.3. SynOps Optimization

Before training the network with QReLU activations, the network was first trained with ReLU to get an initial set of parameters. The network with QReLU was then initialized with the scaled parameters (scaling up by ρ on the first layer). The scaling factor ρ was chosen to initialize the network in a state where enough information is propagated through layers so that the network performs reasonably well. Consequently, the weights of the last weighted layer were scaled by $1/\rho$, in order to adapt the classification loss back to its original range.

During testing, we measured the ANN and SNN performance in terms of their accuracy and SynOps, and found a mismatch of SynOps between training and testing. There are two main reasons: (1) The output of a dropout layer (with a dropout rate p) is always scaled down by $1 - p$ to compensate the dropped out activations, however the mismatch could be large after a sequence of dropout layers. (2) Due to discrete spike events operated in the network where the order (not only the count) of the spikes matters, the mismatch occurs between the spike count-based analog activation and the actual spiking ones.

To compensate for this mismatch, for all the trained models we tested the performance with both 1.5× and 2× scaled-up first layer weights. Lastly, we optimized the QReLU-based model with the objective of minimizing the classification error given a target SynOps. We trained 30 models with lower and lower target SynOps, and each model was initialized with the trained weights of the previous one.

3. RESULTS

3.1. The SynOp Loss Term Leads to a Reduction in Network Activity on DVS Data

In **Figure 2**, we show the results of four methods to reduce the activity of the network, in a way that yields energy savings. First, as a baseline, we trained a traditional CNN using a cross-entropy loss function, and rescaled down the weights of its first layer.

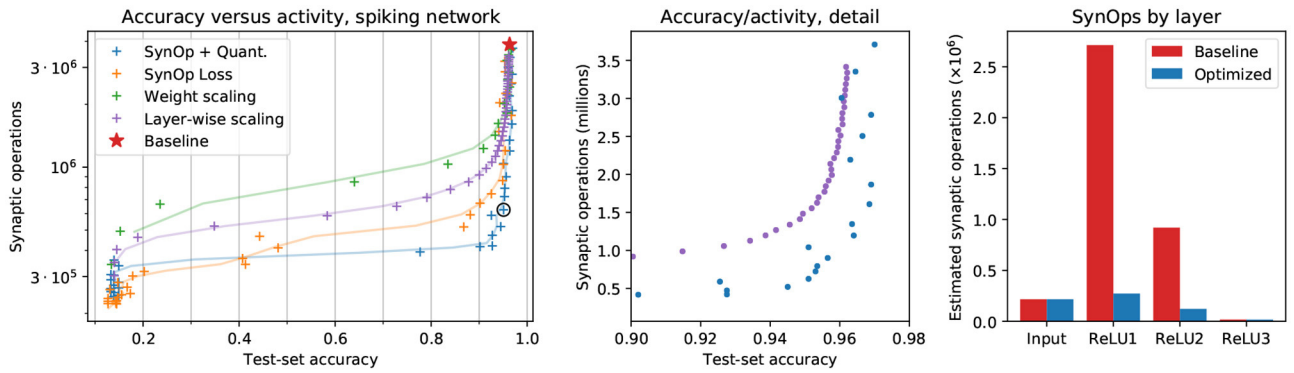


FIGURE 2 | Results on the MNIST-DVS dataset. **Left**: SynOps-accuracy curves computed on a spiking network simulation with Sinabs. Each point represents a different model, trained for a different value of S_0 or rescaled by a different value of ρ. The red star represents the original model, standard CNN weights transferred to the SNN without changes. The solid lines are smoothed versions of the curves described by the data points, provided as a guide to the eye. **Center**: a zoomed-in version of the left panel, showing direct comparison between layer-wise gain scaling (Rueckauer et al., 2017) and our method. **Right**: SynOps per layer, compared between the baseline model and a selected model trained with the SynOp loss and quantization. This is the model indicated by a black circle in the first panel. Note that the input SynOps depend only on the input data, and cannot be changed by training.

This is equivalent to rescaling the input values, and has the effect of proportionally reducing the activity in all subsequent layers of the network. The "baseline" model in **Figure 2** is the same network, with no input rescaling: weights are transferred from the CNN to the corresponding layers of the SNN without any changes or special considerations. Thresholds are set to 1 on all layers. Second, following Rueckauer et al. (2017), we rescaled the weights of each layer in such a way that the maximum activation in each layer stays constant (see section 2). The input weights are again rescaled as stated above. Third, we introduced an additional term in the loss function, the SynOp loss, which directly pushes the estimated number of SynOps to a given value. We trained CNN models, each with a different target number of synaptic operations, independently of each other. Furthermore, excessive reduction of the SynOps leads to the silencing of certain neurons, and other discretization errors, causing an immediate drop in accuracy. To account for this we jointly use the SynOp loss term and quantization-aware training.

We tested our training methods on a real-world use case of SNNs. Dynamic Vision Sensors (DVS) are used in neuromorphic engineering as very-low-power sources of visual information, and are a natural data source for SNNs simulated on neuromorphic hardware. We transferred the weights learned with the methods described above onto a spiking network simulation, and used it to identify the digits presented to the DVS in the MNIST-DVS dataset.

Our results show that adding a requirement on the number of synaptic operations to the loss yields better results in terms of accuracy compared to rescaling input weights and layer-wise activation gains (**Figure 2**, orange). Using the SynOp loss together with quantization during training outperforms the simpler methods, allowing for further reduction of the SynOps value with smaller losses in accuracy (**Figure 2**, blue).

Among the models trained in this way, we selected one with a good balance between energy consumption and accuracy, and used it for a direct comparison with the baseline (that is, weights from an ANN without quantization and no additional loss terms). The second and third panels of **Figure 2** graphically show the large decrease in the number of synaptic operations required by each layer of our model, and the very small reduction in performance. This particular model brings accuracy down from 96.3 to 95.0%, while reducing the number of synaptic operations from 3.86M to 0.63M, an 84% reduction of the SynOp-related energy consumption.

3.2. The SynOp Loss Leads to a Lower Operations Count Compared to ANNs on CIFAR10

SNNs are a natural way of working with DVS events, having advantages over ANNs in event-driven processing. However, it is also interesting to highlight the benefits of using SNNs over ANNs in conventional non-spiking computer vision tasks, e.g., CIFAR-10, where SNNs can still offer advantages in power consumption. As stated in section 2.4, we have trained the network with two approaches: (1) conventional ANN training plus weight scaling as the baseline; (2) further training with QReLU and SynOp Loss for performance optimization.

3.2.1. Weight Scaling

We first trained the analog All-ConvNet on CIFAR-10, attaining an accuracy of 91.37% and a MAC of 306M (red star in **Figure 3**). Then, we transferred the trained weights directly on the equivalent SNN and scaled the weights of its first layer to manipulate the overall activity level. This is shown by the blue crosses in **Figure 3**: as the SynOp count grows, so does the accuracy. However, the SynOps are around 10 times to the MAC of ANN when the accuracy reaches an acceptable rate of 90.7%. To improve on this result, we fine-tuned this training by adding quantization and the SynOp loss.

A faster way to measure the same quantities is by testing the analog model, with ReLU layers all replaced with QReLU, and count the activation levels instead of the Sinabs spike counts.

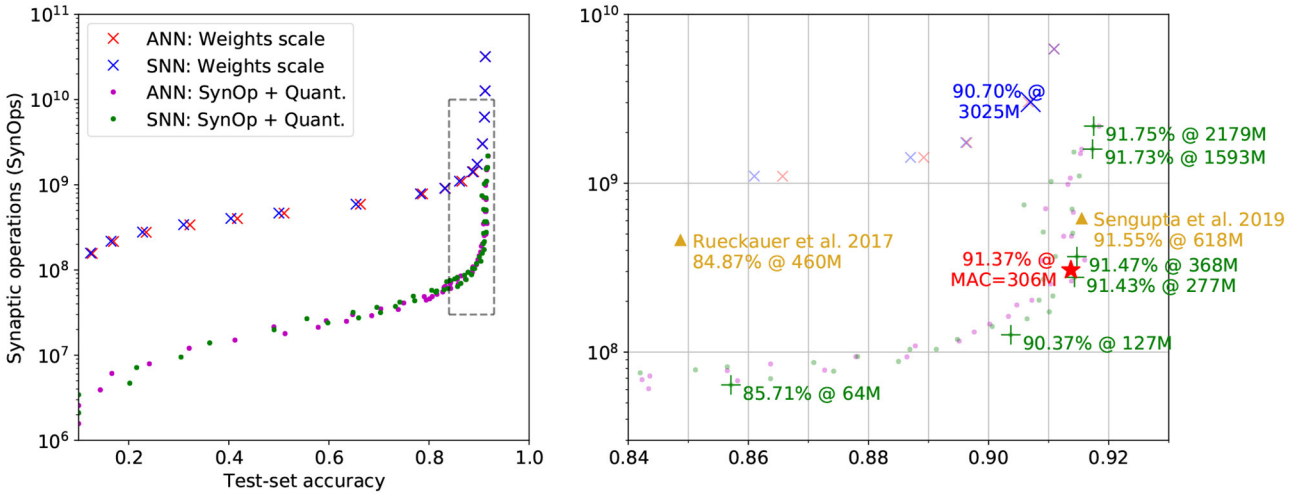


FIGURE 3 | Accuracy vs. SynOps curves on the non-spiking CIFAR-10 task. "ANN" results (red crosses and purple dots) are SynOp count estimations based on the quantized activations of an analog network. "SNN" results (blue crosses and green dots) are SynOp values obtained by Sinabs simulation. The performance of models fine-tuned with SynOp loss and quantization (dots) shows a clear advantage over weight rescaling (crosses). Right panel: a zoomed-in plot of the dashed region. A high-light blue cross represents a good performance of the arbitrary weight scaling. The good trade-off points, trained with SynOp loss and quantization, listed in **Table 1** are marked with green "+." The results from other work are marked with orange triangles. The original ANN model, for which the MAC is plotted instead of the SynOps, is marked with a red star.

Estimations based on this quantized activation layers are shown as red crosses in **Figure 3**. The performance on accuracy and SynOps of the analog network and its spiking equivalent are well aligned, showing that quantized activations are a good proxy for the firing rates of the simulated SNN, at least in this regime.

3.2.2. SynOp Loss Optimization

We further fine-tuned one of the weight-scaled models obtained above, with the addition of quantization-aware training and the SynOp loss. **Figure 3** also shows the classification accuracy and SynOps for both quantized-analog and spiking models (blue and green dots, respectively) trained with this method.

Multiple SNN test trials achieve better accuracy than the original ANN model (red star, 91.37%), thanks to the further training with QReLU. As the SynOp goes down, the accuracy stays above the original ANN model until 91.43% when SynOps are at 277M (see one of the green "+" in **Figure 3**). Note that, the SNN has outperformed ANN both on accuracy and operations count, where the number of MAC in the original ANN is 306M. As another good example of accuracy-SynOp trade-off (90.37% at 127M), our model could perform reasonably well, above 90%, by reducing 58% (Syn-MAC ratio is 0.42) of computing operations from the original ANN. Therefore, running the SNN model on neuromorphic hardware will benefit on energy efficiency not only from the lower computation cost of SynOps but also from the significant reduction on operation counts. Additionally, the plot shows how this method outperforms weight scaling in terms of operation counts by roughly a factor of 10 for all accuracy values.

As far as we know, our converted SNN model from the AllConvNet reached the state-of-the-art accuracy at 91.75% among SNN models (see detailed comparison in **Table 1** and **Figure 3**). In addition, our model is the smallest, at 1.9M parameters, while the BinaryConnect model (Rueckauer et al., 2017) is 7 times larger in size and WeightNorm, consisting of a VGG-16 (Sengupta et al., 2019), is eight-fold in size. Although achieving the best accuracy requires a SynOp of 2,179M, this can easily be reduced by 27% by giving up 0.02% in accuracy, see the two green "+" on the top-right of **Figure 3**. Comparing to the result from Sengupta et al. (2019) (orange triangle on the right of **Figure 3**), our model achieves 91.47% in accuracy at 368M SynOps, thus only loses 0.08% in accuracy but saves 41% of SynOps and energy. Thanks to the optimization of the SynOp loss, the number of SynOps is continuously pushed down while keeping an appropriate accuracy, e.g., 85.71% at a SynOp of 64M. This result not only outperforms most of the early attempts of SNN models for the CIFAR-10 task (Cao et al., 2015; Hunsberger and Eliasmith, 2015; Panda and Roy, 2016), but also brings down the SynOps to only 1/5 of the MAC and saves 86% energy compared to Rueckauer et al. (2017).

In a brief summary, (1) the energy-aware training strategy pushes down the SynOps 10 times compared to its weight scaling baseline; (2) the QReLU-trained SNN achieves the state-of-the-art accuracy in CIFAR-10 task; and (3) the trade-off performances between accuracy and energy show a significant save in computation cost/energy comparing to existing SNN models and the equivalent non-spiking CNN.

3.2.3. SynOp vs. Accuracy for Shorter Inference Times

Unlike the DVS data, which has its own time dynamics, there are no restrictions on how static images should be presented to the network in time. Therefore we measure total spike count instead of firing rates, thus calculating the total energy

TABLE 1 | Comparison with best SNN models on CIFAR-10.

SNN models	Net architecture		Best accuracy			Accuracy-SynOps trade-off		
	N. par.	MAC	Acc.	SynOps	Syn-MAC ratio	Acc.	SynOps	Syn-MAC ratio
BinaryConnect	14M	616M	90.85	N/A	N/A	84.87	460M	0.75
WeightNorm	15M	313M	91.55	**618M**	**1.98**	**91.55**	**618M**	**1.98**
Ours	**1.9M**	**306M**	**91.75**	2179M	7.12	**91.47**	368M	1.20
						91.43	277M	0.91
			91.73	1593M	5.21	90.37	127M	0.42
						85.71	**64M**	**0.21**

BinaryConnect: 8-layer ConvNet from Rueckauer et al. (2017); WeightNorm: VGG-16 model from Sengupta et al. (2019). The most efficient model and the best performance are highlighted as bold fonts. Regarding to the comparisons on accuracy-SynOps trade-off, blue colored result in our models refers to the performance close to that of Rueckauer et al. (2017); while green shows the result approximates to Sengupta et al. (2019). And the numbers in bold only highlight the winner in these two comparisons.

cost per image, independently of time. For example, **Figure 3** shows how SynOp loss optimization pushes the SynOp to a value lower than the MAC during training. This is one of the approaches in which SNNs outperform ANNs in the accuracy-operations trade-off; while the other benefit SNNs naturally bring is the temporal encoding and computation. SNNs continuously output a prediction from the moment when the input currents are injected. This prediction becomes more accurate with more time. For the experiments presented in the previous sections, we only measure the classification accuracy when the input is completely forwarded into the network, $N_{dt} = 10$: the input currents are chosen so that the total input accumulated over $N_{dt} = 10$ time steps is equivalent to that of the analog network during training. In **Figure 4**, we measure how SNN models perform in the course of the entire process, $N_{dt} = 1, 2, 3, ..., 10$. The figure shows how classification accuracy increases when more simulation time-steps are allowed, and therefore more accumulated current is injected to the network. Each gray curve represents a single trained model, and the SynOp and accuracy are tested with increasing N_{dt}. The colored dots mark the accuracy-SynOps pair at $N_{dt} = 4, 6, 8, 10$ over all trained networks. The same-colored dots approach to the expected SNN result (green dots at $N_{dt} = 10$) as N_{dt} increases.

On the other hand, understanding the relationship between inference time and accuracy is very relevant when dealing with DVS data. In general, a global reduction of spike rates in a rate-based network causes a corresponding increase in the latency, since more time is needed to accumulate enough spikes for a reliable prediction. We compared one of our networks with an equivalent model prepared through the "robust" layer-wise normalization technique from Rueckauer et al. (2017), with a few different values of input weight scale. **Figure 5** shows that the dependency of accuracy on the inference time follows a similar trajectory for all these models. We conclude that the increase in latency does not depend on the specific method used for optimization, and our network's latency is similar to that of other models with similar accuracy, despite the much lower power consumption (shown in previous sections).

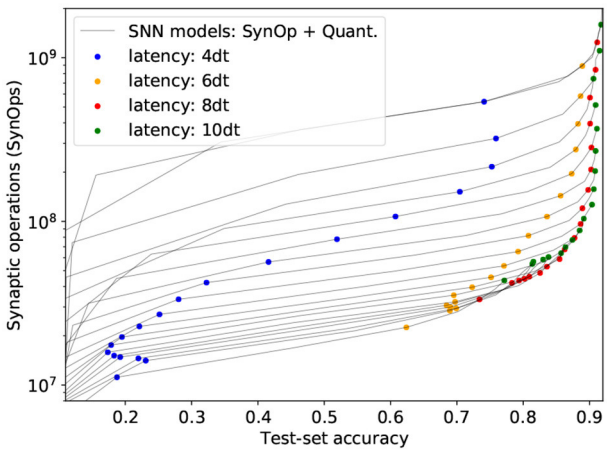


FIGURE 4 | Total activity and accuracy on the CIFAR-10 benchmark for increasing inference times (current injected for longer times, thus leading to more SynOps). Gray lines correspond to individual SNN models, tested at $N_{dt} = 1, ..., 10$. To facilitate same N_{dt} comparison, colored dots were added for all models at $N_{dt} = 4, 6, 8, 10$.

3.2.4. Effects on Weight Statistics

Common regularization techniques in ordinary neural networks often involve the inclusion of an L_1 or L_2 cost on the network weights. In rough, intuitive terms, L_1 regularization has a sparsifying effect, pushing smaller connections toward zero; L_2 regularization generally keeps the weights from growing to excessively large values. Conversely, the effect on weights of penalizing synaptic operations or reducing the network's activity, as we do with the SynOp loss term, is not immediately clear. We investigate whether imposing low synaptic operations count has a sparsifying effect on the weight structure. To this end, we examine how many synaptic connections in our models are null connections, which we define as weights w such that $|w| < 10^{-9}$ (this threshold can be changed by several orders of magnitude without impacting the conclusions). We performed this analysis on the networks trained on the CIFAR-10 dataset, as explained in the previous sections. These networks are much wider and

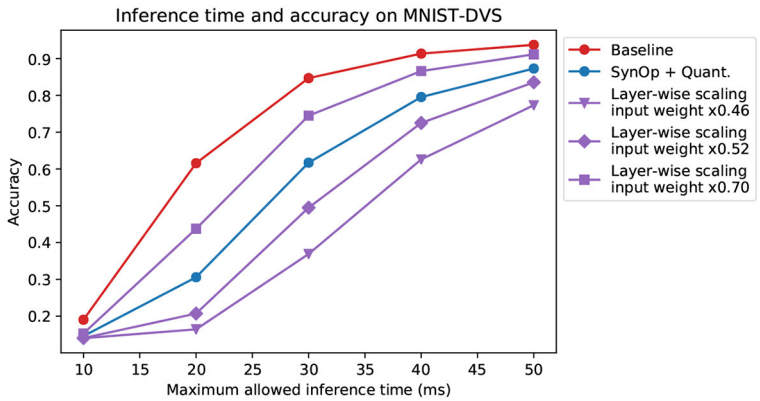


FIGURE 5 | Limited-time inference on MNIST-DVS. Here, the accuracy of the networks is measured at a limited input length of 10, 20, 30, 40, 50 ms. The accuracy of the network trained with our method (again, we chose the one indicated by the black circle in **Figure 2**) behaves, in relation to observation time, similarly to that of other networks, but with lower power consumption. The models and color choices are the same as in **Figure 2**. The different purple lines correspond to different rescaling of the input layer weights, chosen so to have a comparable accuracy with the other curves.

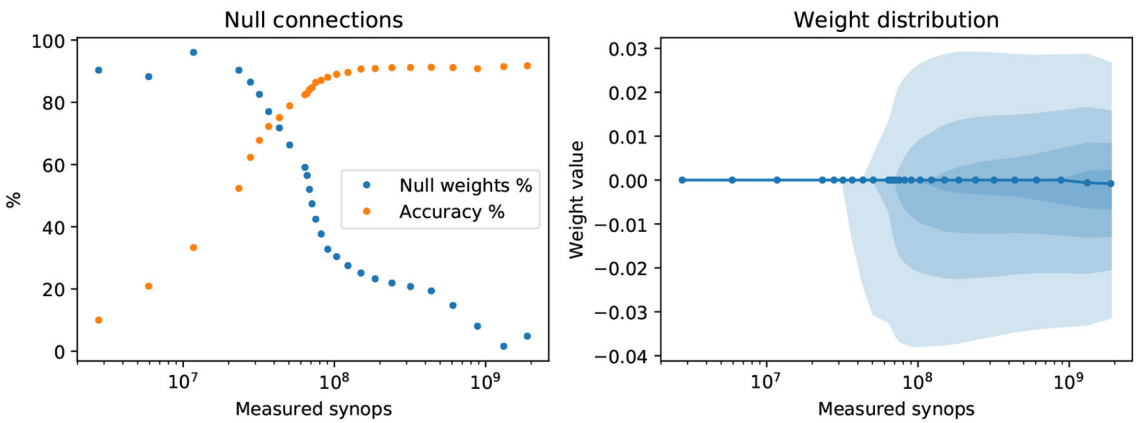


FIGURE 6 | Effect of the SynOp loss on the network's weights. **Left**: the fraction of near-zero weights ($|w| < 10^{-9}$) greatly increases in models where a stricter reduction of SynOp counts were imposed. The test-set accuracy values for each model are also shown for comparison. **Right**: the distribution of weights as a function of the SynOp count. Shaded areas indicate, from lighter to darker, the following inter-quantile ranges: 10–90%; 20–80%; 30–70%; 40–60%. The solid line is the median weight. The models used for this test are the same as those shown in **Figure 3**, trained with quantization on static CIFAR images.

deeper than the ones used for the MNIST-DVS task, and therefore can better show weight sparseness effects. **Figure 6** (left) shows how the fraction of null weights changes with the SynOp count (and thus, of the regularization strength), and compares it with the model's accuracy. When the number of synaptic operations is forced to be extremely low, the fraction of null weights reaches values above 90%. A large increase in null connections, however, is already noticeable for models above 80% accuracy, showing that the SynOp loss term does have a sparsifying effect, and that this is desirable. For the sake of completeness, in **Figure 6** (right), we also show a depiction of the distribution of weights as a function of the number of synaptic operations.

Setting synaptic weights to zero is effectively equivalent to pruning certain connections between a layer and the next. Other than L_1 regularization of the weights, more sophisticated pruning-and-retraining algorithms have been studied in the machine learning literature (LeCun et al., 1990; Hassibi and Stork, 1993). However, advanced pruning methods (such as those based on Fisher information) are usually coupled with partial retraining of the network, and are therefore more alike a form of architectural search (Crowley et al., 2018). Due to the retraining of the remaining weights, these forms of pruning are not guaranteed to reduce the activity levels if not coupled to other forms of regularization.

4. DISCUSSION AND CONCLUSION

We used two techniques which significantly improve the energy requirements of machine learning models that run on neuromorphic hardware, while maintaining similar performances.

The first improvement consisted in optimizing the energy expenditure by directly adding it to the loss function during training. This method encourages smaller activations in all

neurons, which is not in itself an issue in analog models, but can lead to discretization errors, due to the lower firing rates, once the weights are transferred to a spiking network. To solve this problem, we introduced the second improvement; quantization-aware training, whereby the network activity is quantized at each layer, i.e., only integer activations are allowed. Discretizing the network's activity would normally reduce all gradients to zero: this can be solved by substituting the true gradient with a surrogate.

Applying these two methods together, we achieved an up to 10-fold drop in the number of synaptic operations and the consequent energy consumption in the DVS-MNIST task, with only a minor (1-2%) loss in performance, when comparing to simply transferring the weights from a trained CNN to a spiking network. To demonstrate the scalability of this approach, we also show that, as the network grows bigger to solve a much more complex task of CIFAR-10 image classification, the SynOps are reduced to 42% of the MAC, while losing 1% of accuracy (90.37% at 127M). The accuracy-energy trade-off can be flexibly tuned at training time. We also showed the consequences of using this method on the distribution of network weights and the network's accuracy as a function of time.

While training based on static frames is not the optimal approach to leverage all the benefits of spike-based computation, it enables fast training with the use of state-of-the-art deep learning tools. In addition, the hybrid strategy to train SNNs based on a target power metric is unique to SNNs. Conversely, optimizing the energy requirement of an ANN/CNN requires modification of the network architecture itself, which can require large amounts of computational resources (Cai et al., 2018). In this work, we demonstrated that we can train an SNN to a target energy level without a need to alter the network hyperparameters. A potential drawback of this approach of (re)training the model as opposed to simply transferring the weights of a pre-trained model is brought to light when attempting to convert very deep networks trained over large datasets such as IMAGENET. Pre-trained deep CNNs trained over large datasets are readily available on the web and can be used to quickly instantiate a spiking CNN. The task becomes much more cumbersome to optimize for power utilization using the method described in this paper, ie. one has to retrain the network over the relevant dataset for optimal performance. However, our method can also be effectively used to fine-tune a pre-trained network, removing the need for training from scratch (**Supplementary Figure 1**). Furthermore, no large event-based datasets of the magnitude of IMAGENET exist currently, and perhaps when such datasets are generated, the corresponding models optimized for spiking CNNs will also be developed and made readily available.

The quantization and SynOp-based optimization used in this paper can potentially be applied, beyond the method illustrated here, in more general contexts such as algorithms based on back-propagation through time to reduce power usage. Such a reduction in power usage can make a large difference when the model is ran on a mobile, battery-powered, neuromorphic device, with potential for a significant impact in the industrial applications.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: http://www2.imse-cnm.csic.es/caviar/MNISTDVS.html, http://www.cs.toronto.edu/~kriz/cifar.html.

AUTHOR CONTRIBUTIONS

SS designed the research. QL and SS contributed to the methods. MS, QL, and MB contributed the code and performed the experiments. All authors wrote the paper.

FUNDING

This work was supported in part by H2020 ECSEL grant TEMPO (826655). The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

ACKNOWLEDGMENTS

The authors would like to thank Mr. Felix Bauer, Mr. Ole Richter, Dr. Dylan Muir, and Dr. Ning Qiao for their support and feedback on this work.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00662/full#supplementary-material

REFERENCES

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

Cai, H., Zhu, L., and Han, S. (2018). Proxylessnas: direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

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

Crowley, E. J., Turner, J., Storkey, A., and O'Boyle, M. (2018). A Closer Look at Structured Pruning for Neural Network Compression. *arXiv preprint arXiv:1810.04622*.

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

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S.-C., 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, IL: IEEE), 1–8. doi: 10.1109/IJCNN.2015.7280696

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast energy-efficient neuromorphic computing. *Proc. Nat. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Furber, S. (2016). Large-scale neuromorphic computing systems. *J. Neural Eng.* 13:051001. doi: 10.1088/1741-2560/13/5/051001

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

Guo, Y. (2018). A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*.

Hassibi, B., and Stork, D. G. (1993). "Second order derivatives for network pruning: optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 164–171.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* 18, 6869–6898.

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

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Front. Neurosci.* 5:73. doi: 10.3389/fnins.2011.00073

Indiveri, G., and Sandamirskaya, Y. (2019). The importance of space and time for signal processing in neuromorphic agents: the challenge of developing low-power, autonomous agents that interact with the environment. *IEEE Signal Process. Mag.* 36, 16–28. doi: 10.1109/MSP.2019.2928376

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

Krizhevsky, A., Hinton, G., et al. (2009). *Learning Multiple Layers of Features From Tiny Images*. Technical report, Citeseer.

LeCun, Y., Cortes, C., and Burges, C. J. (1998). *The MNIST database of handwritten digits, 1998*. Available online at: http://yann.lecun.com/exdb/mnist

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 598–605.

Liu, Q., Pineda-García, G., Stromatias, E., Serrano-Gotarredona, T., and Furber, S. B. (2016). Benchmarking spike-based visual recognition: a dataset and evaluation. *Front. Neurosci.* 10:496. doi: 10.3389/fnins.2016.00496

Liu, Q., Richter, O., Nielsen, C., Sheik, S., Indiveri, G., and Qiao, N. (2019). "Live demonstration: face recognition on an ultra-low power event-driven convolutional neural network ASIC," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. doi: 10.1109/CVPRW.2019.00213

Marblestone, A. H., Wayne, G., and Kording, K. P. (2016). Toward an integration of deep learning and neuroscience. *Front. Comput. Neurosci.* 10:94. doi: 10.3389/fncom.2016.00094

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

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*.

Mostafa, H. (2017). Supervised learning based on temporal coding in spiking neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 3227–3235. doi: 10.1109/TNNLS.2017.2726060

Neftci, E. O., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948*.

Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). "Learning to be efficient: algorithms for training low-latency, low-compute deep spiking neural networks," in *ACM Symposium on Applied Computing. Proceedings of the 31st Annual ACM Symposium on Applied Computing* (New York, NY: Association for Computing Machinery). doi: 10.1145/2851613.2851724

Nicola, W., and Clopath, C. (2017). Supervised learning in spiking neural networks with force training. *Nat. Commun.* 8:2208. doi: 10.1038/s41467-017-01827-3

Panda, P., and Roy, K. (2016). "Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, CA: IEEE), 299–306. doi: 10.1109/IJCNN.2016.7727212

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop* (Long Beach).

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., et al. (2019). A deep learning framework for neuroscience. *Nat. Neurosci.* 22, 1761–1770. doi: 10.1038/s41593-019-0520-2

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

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

Serrano-Gotarredona, T., and Linares-Barranco, B. (2015). Poker-DVS and MNIST-DVS. Their history, how they were made, and other details. *Front. Neurosci.* 9:481. doi: 10.3389/fnins.2015.00481

Shrestha, S. B., and Orchard, G. (2018). "Slayer: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems*, 1412–1421.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2014). Striving for simplicity: the all convolutional net. *arXiv preprint arXiv:1412.6806*.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*. doi: 10.18653/v1/P19-1355

Thakur, C. S. T., Molin, J., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., et al. (2018). Large-scale neuromorphic spiking array processors: a quest to mimic the brain. *Front. Neurosci.* 12:891. doi: 10.3389/fnins.2018.00891

Yang, T.-J., Chen, Y.-H., and Sze, V. (2017). "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Honolulu), 5687–5695. doi: 10.1109/CVPR.2017.643

Conflict of Interest: All authors were employed by SynSense AG during the course of the work published in this article.

frontiers in Neuroscience

ORIGINAL RESEARCH
published: 27 July 2020
doi: 10.3389/fnins.2020.00587

Event-Based Face Detection and Tracking Using the Dynamics of Eye Blinks

Gregor Lenz[1,2], *Sio-Hoi Ieng*[1,2] *and Ryad Benosman*[1,2,3,4]*

[1] INSERM UMRI S 968, Sorbonne Université, UPMC Univ. Paris, UMRS 968, Paris, France, [2] CNRS, UMR 7210, Institut de la Vision, Paris, France, [3] Departments of Ophthalmology/ECE/BioE, University of Pittsburgh, Pittsburgh, PA, United States, [4] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, United States

We present the first purely event-based method for face detection using the high temporal resolution properties of an event-based camera to detect the presence of a face in a scene using eye blinks. Eye blinks are a unique and stable natural dynamic temporal signature of human faces across population that can be fully captured by event-based sensors. We show that eye blinks have a unique temporal signature over time that can be easily detected by correlating the acquired local activity with a generic temporal model of eye blinks that has been generated from a wide population of users. In a second stage once a face has been located it becomes possible to apply a probabilistic framework to track its spatial location for each incoming event while using eye blinks to correct for drift and tracking errors. Results are shown for several indoor and outdoor experiments. We also release an annotated data set that can be used for future work on the topic.

Keywords: face detection, face tracking, event-based computation, neuromorphic vision, silicon retina

OPEN ACCESS

Edited by:
Chiara Bartolozzi,
Italian Institute of Technology (IIT), Italy

Reviewed by:
Guoqi Li,
Tsinghua University, China
Ander Arriandiaga,
University of the Basque Country, Spain

***Correspondence:**
Ryad Benosman
ryad.benosman@upmc.fr

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *04 September 2019*
Accepted: *12 May 2020*
Published: *27 July 2020*

Citation:
Lenz G, Ieng S-H and Benosman R (2020) Event-Based Face Detection and Tracking Using the Dynamics of Eye Blinks. Front. Neurosci. 14:587. doi: 10.3389/fnins.2020.00587

1. INTRODUCTION

This paper introduces an event-based method to detect and track faces from the output of an event-based camera. We also release a dataset of 50 recordings, consisting of a mix of indoor and outdoor conditions. 25 of those recordings have been annotated for a total of 265 blinks[1]. The method exploits the dynamic properties of human faces to detect, track and update multiple faces in an unknown scene. Although face detection and tracking are considered practically solved in classic computer vision, it is important to emphasize that current performances of conventional frame based techniques come at a high operating computational cost after days of training on large databases of static images. Event-based cameras record changes in illumination at high temporal resolutions (in the range of $1\mu s$ to 1 ms) and are therefore able to acquire the dynamics of moving targets present in a scene (Lichtsteiner et al., 2008). In this work we will rely on eye blink detection to determine the presence of a face in a scene to in a second stage initialize the position of a bayesian tracker. The permanent computation of eye blinks will allow to correct tracking drifts and reduce localization errors over time. Blinks produce a unique space-time signature that is temporally stable across populations and can be reliably used to detect the position of eyes in an unknown scene. This paper extends the sate-of-art by:

- Implementing a low-power human eye-blink detection that exploits the high temporal precision provided by event-based cameras.
- Tracking of multiple faces simultaneously at μs precision, once they have been detected.

[1]Dataset is available for download under https://www.neuromorphic-vision.com/public/downloads/data-set-face-detection.tar.gz; **Supplementary Material**.

The developed methodology is entirely event-based as every event output by the camera is processed into an incremental, non-redundant scheme rather than creating frames from events to recycle existing image-based methodology. We also show that the method is inherently robust to scale changes of faces by continuously inferring the scale from the distance of the two eyes of the tracked face from detected eye blinks. The method is compared to existing image-based face detection techniques (Viola and Jones, 2004; Liu et al., 2016; Jiang and Learned-Miller, 2017; Li and Shi, 2019). It is also tested on a range of scenarios to show its robustness in different conditions: indoors and outdoors scenes to test for the change in lighting conditions; a scenario with a face moving close and moving away to test for the change of scale, a setup of varying pose and finally a scenario where multiple faces are detected and tracked simultaneously. Comparisons with existing frame-based methods are also provided.

1.1. Event-Based Cameras

Biomimetic event-driven time-based vision sensors are a novel class of vision device that—like the biological retina—are driven by "events"happening within the visual scene. They are not like conventional vision sensors, which are driven by artificially created timing and control signals (e.g., frame clock) that have no relation whatsoever to the source of the visual information (Lichtsteiner et al., 2008).

Over the past few years, a variety of these event-based devices has been developed, including temporal contrast vision sensors that are sensitive to relative luminance change (Lichtsteiner et al., 2008), some also providing also absolute light measurement (Posch et al., 2011).

These vision sensors output visual information about the scene in the form of asynchronous address events using the Address Event Representation protocol and encode the visual information in the time dimension rather than as a voltage, charge, or current. The novel algorithm for face detection and tracking we propose in this paper is designed to take advantage of the high temporal resolution data representation provided by event-based cameras. The operating principle of these sensor is shown in **Figure 1**. An event is defined as the n-tuple: $ev = (x, y, t, p)$, where (x, y) are the pixel coordinates, t the time of occurrence and p is the polarity. Variations of event-based cameras implement additional functionality.

In this work, we are using the Asynchronous Time-based Image Sensor (ATIS) (Posch et al., 2011) as it also provides events that encode absolute luminance information. This additional information allows for direct and easier comparisons with the frame-based world.

1.2. Face Detection

State-of-the-art face detection relies on neural networks that are trained on large databases of face images, to cite the latest from a wide literature, readers should refer to Yang et al. (2017), Jiang and Learned-Miller (2017), and Sun et al. (2018). Neural Networks usually rely on intensive computation that necessitates dedicated hardware co-processors (usually GPU) to enable real-time operations (Ren et al., 2008). Currently dedicated chips such as Google's Tensor Processing Unit or Apple's Neural Engine have become an essential tool for frame-based vision. They are designed to accelerate matrix multiplications at the core of neural networks inference. However, the computation costs associated to these computations are extremely high (thousands of Watts).

Dedicated blink detection using conventional frame-based techniques operate on a single frame. To constrain the region of interest, a face detection algorithm is used beforehand (Noman and Ahad, 2018). In an event-based approach, the computational scheme can be inverted as detecting blinks is the mechanism that drives face detection.

1.3. Human Eye Blinks

Humans blink synchronously in correlation to their cognitive activities and more often than required to keep the surface of the

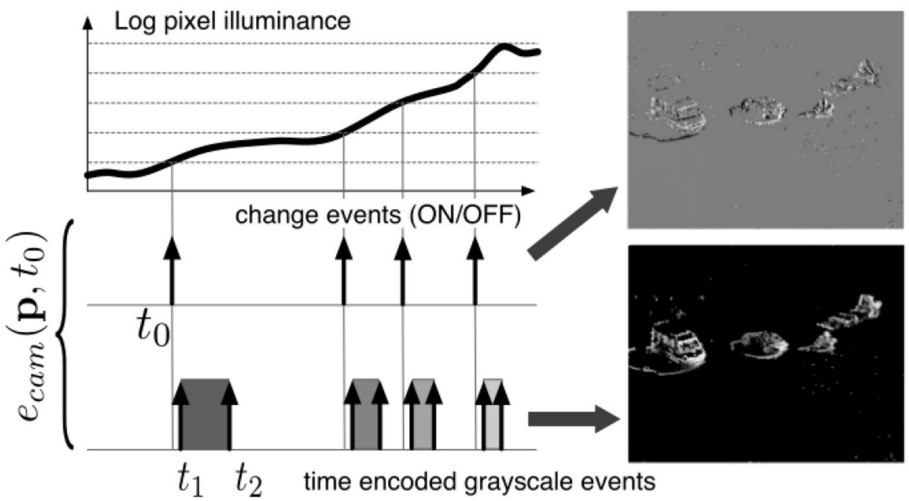


FIGURE 1 | Working principle of the event-based camera and two types of events. (1) Change event of type ON is generated at t_0 as voltage generated by incoming light crosses a voltage threshold. (2) Time $t_2 - t_1$ to receive a certain amount of light is converted into an absolute gray-level value, emitted at t_2 used for ground truth in the paper.

eye hydrated and lubricated. Neuroscience research suggests that blinks are actively involved in the release of attention (Nakano et al., 2013). Generally, the observed eye blinking rates in adults depend on the subject's activity and level of focus. Rates can range from 3 *blinks/min* when reading to up to 30 *blinks/min* during conversation (**Table 1**). Fatigue significantly influences blinking behaviors, increasing both rate and duration (Stern et al., 1994). In this paper we will not consider these boundary cases that will the be subject of further work on non-intrusive driver monitoring (Häkkänen et al., 1999; Wang et al., 2006). A typical blink duration is between 100 and 150 *ms* (Benedetto et al., 2011). It shortens with increasing physical workload, increased focus or airborne particles related to air pollution (Walker et al., 2001).

To illustrate what happens during an event-based recording of an eye blink, **Figure 2** shows different stages of the eye lid closure and opening. If the eye is in a static state, few events will be generated (**Figure 2A**). The closure of the eye lid happens within 100 ms and generates a substantial amount of ON events, followed by a slower opening of the eye (**Figures 2C,D**) and the generation of primarily OFF events. From this observation, we devise a method to build a temporal signature of a blink. This signature can then be used to signal the presence of a single eye or pair of eyes in the field of view that can then be interpreted as the presence of a face.

TABLE 1 | Mean blinking rates according to Bentivoglio et al. (1997) and Stern et al. (1994).

Activity (blinks/min)	Bentivoglio et al. (1997)	Stern et al. (1994)
Reading	4.5	3–7
At rest	17	–
Communicating	26	–
Not reading	–	15–30

2. METHODS

2.1. Temporal Signature of an Eye Blink

Eye blinks can be represented as a temporal signature. To build a canonical eye blink signature $A(t_i)$ of a blink, we convert events acquired from the sensor into temporal activity. For each incoming event $ev = (x_i, y_i, t_i, p_i)$, we update $A(t_i)$ as follows:

$$A(t_i) = \begin{cases} A_{ON}(t_i) = A_{ON}(t_u)e^{-\frac{t_i - t_u}{\tau}} + \frac{1}{scale} & \text{if } p_i\text{=ON} \\ A_{OFF}(t_i) = A_{OFF}(t_v)e^{-\frac{t_i - t_v}{\tau}} + \frac{1}{scale} & \text{if } p_i\text{=OFF} \end{cases} \quad (1)$$

where t_u and t_v are the times an ON or OFF event occurred before t_i. The respective activity function is increased by $\frac{1}{scale}$ each time t_n an event ON or OFF is registered (light increasing or decreasing). The quantity *scale* initialized to 1 acts as a corrective factor to account for a possible change in scale, as a face that is closer to the camera will inevitably trigger more events. **Figure 3** shows the two activity profiles where 5 profiles of a subject's blinks are shown, as well as much higher activities at the beginning and the end of the sequence when the subject moves as a whole. From a set of manually annotated blinks we build such an activity model function as shown in **Figure 2** where red and blue curve respectively represent the ON and OFF parts of the profile.

Our algorithm detects blinks by checking whether the combination of local ON- and OFF-activities correlates with the canonical model of a blink that had previously been "learned" from annotated material. To compute the local activity, the

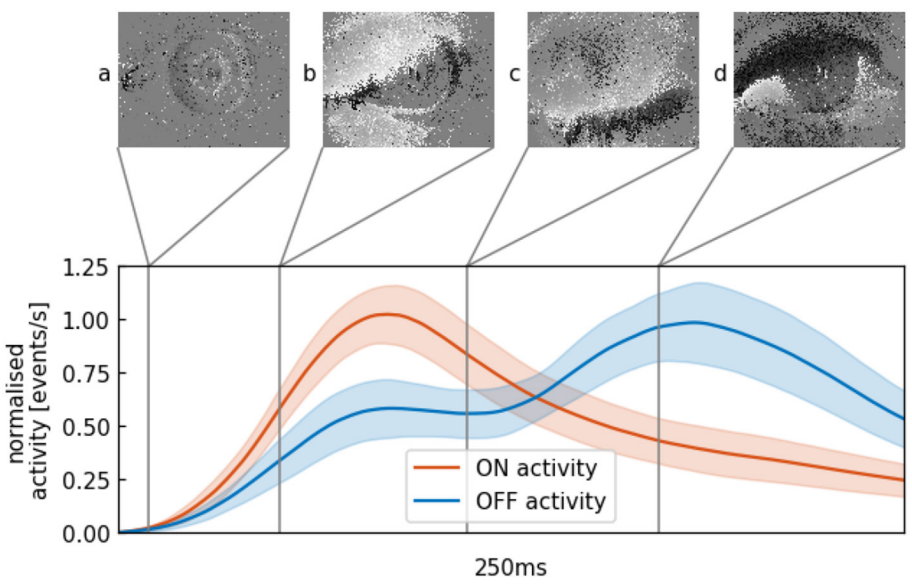


FIGURE 2 | Mean and variance of the continuous activity profile of averaged blinks in the outdoor data set with a decay constant of 50 ms. **(A)** Minimal movement of the pupil, almost no change is recorded. **(B)** Eye lid is closing within 100 ms, lots of ON-events (in white) are generated. **(C)** Eye is in a closed state and a minimum of events is generated. **(D)** Opening of the eye lid is accompanied by the generation of mainly OFF-events (in black).

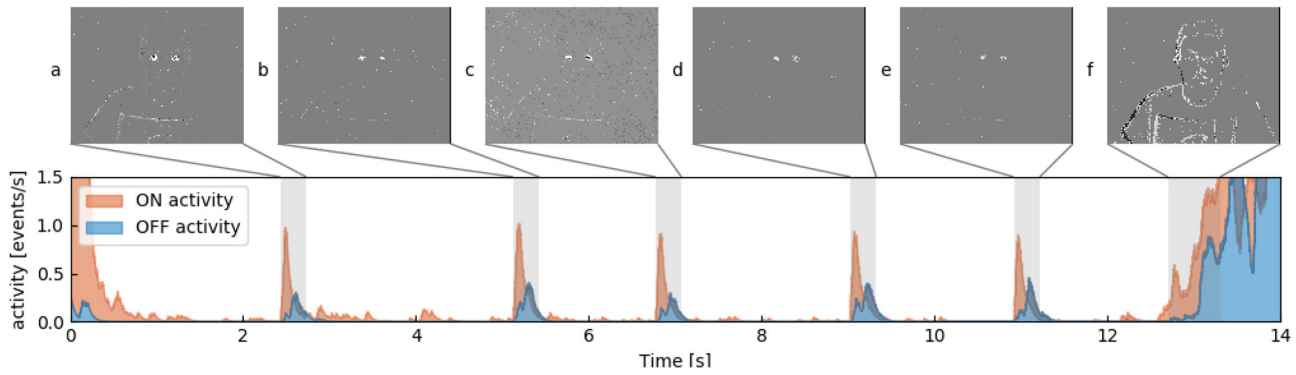


FIGURE 3 | Showing ON (red) and OFF (blue) activity for one tile which lines up with one of the subject's eyes. Multiple snapshots of accumulated events for 250 ms are shown, which corresponds to the gray areas. **(A–E)** Blinks. Subject is blinking. **(F)** Subject moves as a whole and a relatively high number of events is generated.

overall input focal plane is divided into one grid of $n \times n$ tiles, overlapped with a second similar grid made of $(n-1) \times (n-1)$ tiles. Each of these are rectangular patches, given the event-camera's resolution of 304×240 pixels. The second grid is shifted by half the tile width and height to allow for redundant coverage of the focal plane. In this work we set experimentally $n = 16$ as it corresponds to the best compromise between performance and the available computational power of the used hardware.

2.1.1. Blink Model Generation

A total of $M = 120$ blinks from six subjects are manually annotated from the acquired data to build the generic model of an eye blink shown in **Figure 2**. Each blink, extracted within a time window of 250 ms is used to compute an activity function as defined in Equation (1). The blink model is then obtained as the average of these activity functions:

$$B(t) = \begin{cases} B_{ON}(t) = \sum_{k=1}^{M} \frac{A_{ON}(t)}{M}, & \text{if } p_i\text{=ON} \\ B_{OFF}(t) = \sum_{k=1}^{M} \frac{A_{OFF}(t)}{M}, & \text{if } p_i\text{=OFF} \end{cases} \quad (2)$$

To provide some robustness and invariance to scale and time changes to the blink model, we also define N, the number of events per unit of time and normalized by the scale factor. This number provides the number of samples necessary to calculate the cross-correlation to detect blink as explained in section 2.1.2. Formally, $N = \left\lfloor \frac{\#\text{events}}{T.scale} \right\rfloor$, where $\lfloor . \rfloor$ is the floor function giving the largest integer smaller than $\frac{\#\text{events}}{T.scale}$.

Finally, we used two different models for indoor and outdoor scenes, as experiments showed that the ratio between ON and OFF events change substantially according to the lighting conditions. Although the camera is supposed to be invariant to absolute illumination, this is practically not the case due to hardware limitations of early camera generation that we used for this paper.

2.1.2. Sparse Cross-Correlation

When streaming data from the camera, the most recent activity within a $T = 250$ ms time window is taken into account in each tile to calculate the similarity score, here the cross-correlation score, for the ON and OFF activities. This cross-correlation is only computed if the number of recent events exceeds an amount N defined experimentally according to the hardware used. The cross-correlation score between the incoming stream of events and the model is given by:

$$C(t_k) = \alpha C_{ON}(t_k) + (1-\alpha) C_{OFF}(t_k), \quad (3)$$

where

$$C_p(t_k) = \sum_{i=0}^{N} A_p(t_i) B_p(t_i - t_k), \quad (4)$$

with $p \in \{ON, OFF\}$. The ON and OFF parts of the correlation score are weighted by a parameter α set experimentally that tunes the contribution of the ON vs OFF events. This is a necessary step -as explained in the previous section-, due to the camera manual parameter settings, the amount of ON and OFF events are usually not balanced. For all experiments, α is set to $\frac{2}{3}$.

It is important for implementation reasons to compute the correlation as it appears in Equation (4). While it is possible to calculate the value of the model $B(t - t_k)$ at anytime t, samples for A are only known for the set of times $\{t_i\}$, from the events. This is illustrated as an example by **Figure 4**, for an arbitrary time t_k, where triangles outline the samples of the activity for calculated events at t_i and the circles show the samples calculated at the same time t_i with the model. If $C(t_i)$ exceeds a certain threshold, we create what we call a blink candidate event for the tile in which the event that triggered the correlation occurred. Such a candidate is represented as the n-tuple $eb = (r, c, t)$, where (r, c) are the row and column coordinates of the grid tile and t is the timestamp. We do this since we correlate activity for tiles individually and only in a next step combine possible candidates to a blink.

2.1.3. Blink Detection

To detect the synchronous blinks generated by two eyes, blink candidates across grids generated by the cross-correlation are

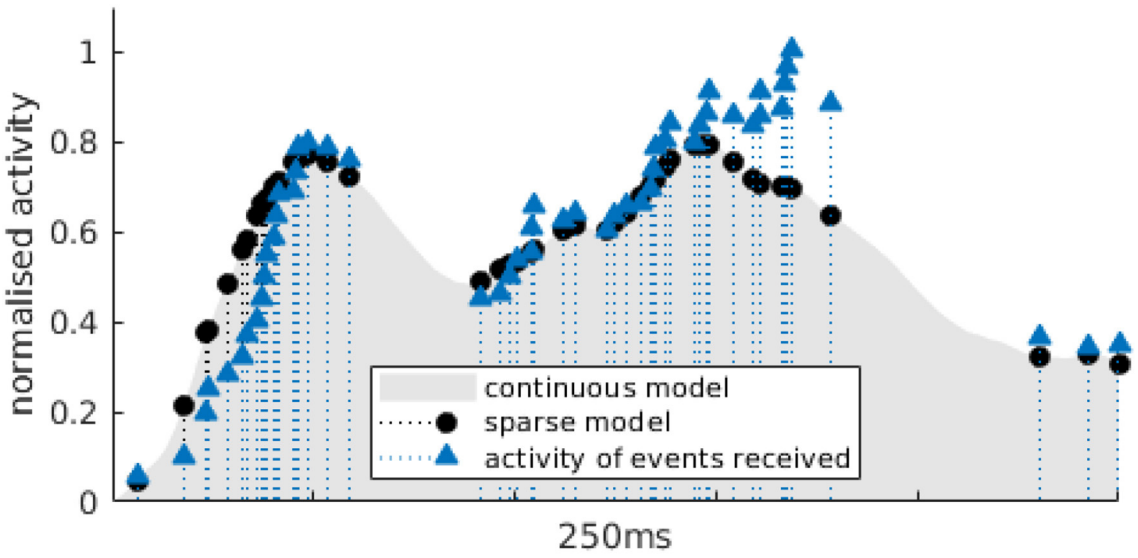


FIGURE 4 | Example of the samples used to calculate the sparse cross-correlation for the OFF activity of an actual blink. The gray area represents B_{OFF}, the activity model for OFF events (in that particular example, it is previously built for outdoor data sets). Blue triangles correspond to the activity $A(t_i)$ for which events have been received in the current time window. Black dots are the $B_{OFF}(t_i)$, the value of activity in the model at the same times-tamps as incoming events.

tested against additional constraints for verification. As a human blink has certain physiological constraints in terms of timing, we check for temporal and spatial coherence of candidates in order to find true positives. The maximum temporal difference between candidates will be denoted as ΔT_{max} and is set experimentally to 50 ms, the maximum horizontal spatial disparity ΔH_{max} is set to half the sensor width and the maximum vertical difference ΔV_{max} is set to a fifth of the sensor height. Following these constraints we will not detect blinks that happen extremely close to the camera or stem from substantially rotated faces. Algorithm 1 summarizes the set of constraints to validate the detection of a blink. The scale factor here refers to a face that has already been detected.

Algorithm 1: Blink detection

Inputs: A pair of consecutive blink candidate events $eb_u = (r_u, c_u, t_u)$ and $eb_v = (r_v, c_v, t_v)$ with $t_u > t_v$
if $(t_u - t_v < \Delta T_{max})$ *AND* $(|r_u - r_v| < \Delta V_{max} \times scale)$ *AND* $(|c_c - c_v| < \Delta H_{max} \times scale)$ **then**
  if *face is a new face* **then**
    return 2 trackers with $scale = 1$
  else
    return 2 trackers with previous *scale*
  end
end

2.2. Gaussian Tracker

Once a blink is detected with sufficient confidence, a tracker is initiated at the detected location. We use trackers such as the ones presented in Lagorce et al. (2015) that rely on bivariate normal distributions to locally model the spatial distribution of events. For each event, every tracker is assigned a score that represents the probability of the event to belong to the tracker:

$$p(u) = \frac{1}{2\pi}|\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{u}-\mu)^T \Sigma^{-1}(\mathbf{u}-\mu)} \quad (5)$$

where $u = [x, y]^T$ is the pixel location of the event, is covariance matrix Σ that defines the shape and size of the tracker. The tracker with the highest probability is updated according to the activity of pixels and also according to the estimated distance between the spatial locations of the detected eyes.

2.3. Global Algorithm

The detection and tracking blocks combined operations are summarized by following algorithm:

Algorithm 2: Event-based face detection and tracking algorithm

for *each event ev(x, y, t, p)* **do**
  if *at least one face has been detected* **then**
    update best blob tracker for *ev* as in (5)
    update *scale* of face for which tracker has moved according to tracker distance
  end
  update activity according to (1)
  correlate activity with model blink as in (3)
  run Algorithm 1 to check for a blink
end

3. EXPERIMENTS AND RESULTS

We evaluated the algorithm's performance by running cross-validation on a total of 48 recordings from 10 different subjects, comprising 248 blinks. The recordings are divided into five sets of experiments to assess the method's performances under realistic constraints encountered in natural scenarios. The event-based camera is set static. We test the following scenarios of sequences of:

- indoor and outdoor sequences showing a single subject moving in front of the camera,
- a single face moving back and forth w.r.t. the camera to test the robustness of scale change,
- several subjects discussing, facing the camera to test for multi-detection,
- a single face changing its orientation w.r.t. the camera to test for occlusion resilience.

The presented algorithm has been implemented in C++ and runs in real-time on an Intel Core i5-7200U laptop CPU. We are quantitatively assessing the proposed method's accuracy by comparing it with state of the art and gold standard face detection algorithms from frame-based computer-vision. As these approaches require frames, we are generating gray-levels from the camera when this mode is available. The Viola and Jones (2004) algorithm (VJ) provides the gold standard face detector but we also considered the Faster R-CNN (FRCNN) from Ren et al. (2015) and the Single Shot Detector (SSD) network from Liu et al. (2016) that have been trained on the Wider Face (Yang et al., 2016) data set. This allows us to compare the performances of the event-based blink detection and tracking with state-of-the-art face detectors based on deep learning. Finally, we also tested a conventional approach that combines CNN and correlation filter presented in Li and Shi (2019). It is referred to as the "Correlation Filter" (CF) for the rest of the paper. This technique, however, relies on creating frames by summing the activities of pixels within a predefined time window.

An important statement to keep in mind is that the proposed blink detection and face tracking technique requires reliable detection. We do not actually need to detect all blinks since a single detection is already sufficient to initiate the trackers. Additional blink detections are used to correct a trackers' potential drifts regularly.

3.1. Indoor and Outdoor Face Detection

The indoor data set consists of recordings in controlled lighting conditions. **Figure 5** shows tracking results. The algorithm starts tracking as soon as a single blink is detected (**Figure 5A**). Whereas tracking accuracy on the frame-based implementation is constant (25 fps), our algorithm is updated event-by-event depending on the movements in the scene. If the subject stays still, the amount of computation is drastically reduced as there is a significantly lower number of events. Head movement causes the tracker to update within μs (**Figure 5B**).

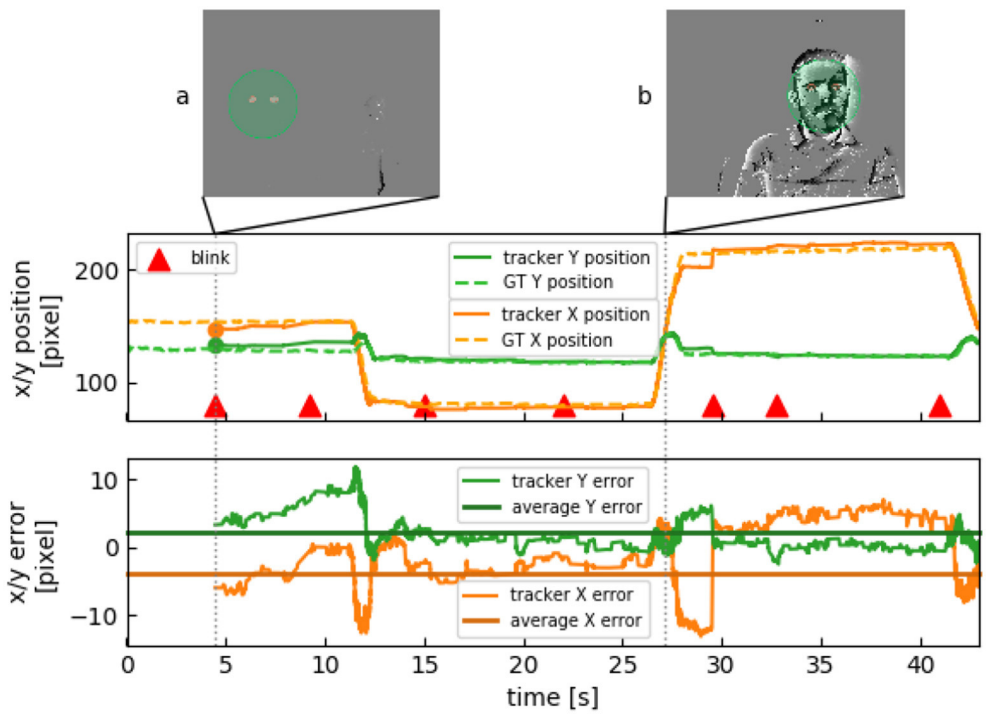


FIGURE 5 | Face tracking of one subject over 45 s. **(A)** Subject stays still and eyes are being detected. Movement in the background to the right does not disrupt detection. **(B)** When the subject moves, several events are generated.

TABLE 2 | Summary of results for detection and tracking for four sets of experiments.

	No. of recordings	Blinks detected (%)	Error VJ (%)	Error FRCNN (%)	Error SSD (%)	Error CF (%)
Indoor	21	68.4	5.92	9.42	9.21	10.51
Outdoor	21	52.3	7.6	14.57	15.08	14.88
Scale	3	62.6	4.8	10.17	10.22	17.6
Multiple	3	36.8	15	16.15	14.61	n/a
Total	48	59	7.68	11.77	11.52	12.82

Percentage of blinks detected relates to the total number of blinks in a recording. Tracking errors are Euclidean distances in pixel between the proposed and respective method's bounding boxes, normalized by the frame-based bounding box width and height in order to account for different scales.

Subjects in the outdoor experiments were asked to step from side to side in front of a camera placed in a courtyard under natural lighting conditions. They were asked to gaze into a general direction, partly engaged in a conversation with the person who recorded the video. **Table 2** shows that results are similar to indoor conditions. The slight difference is due to the non-idealities of the sensor (same camera parameters as in the indoor experiment). It is important to emphasize that Event-based cameras still lack an automatic tuning system of their parameters that hopefully will be developed for the future generations of a cameras.

3.2. Face Scale Changes

In three recordings the scale of a person's face varies by a factor of more than 5 between the smallest to the largest detected occurrence. Subjects were instructed to approach the camera within 25 cm from their initial position to then move away from the camera after 10 s to about 150 cm. **Figure 6** shows tracking data for such a recording. The first blink is detected after 3 s at around 1 m in front of the camera (**Figure 6A**). The subject then moves very close to the camera and to the left so that not even the whole face bounding box is seen anymore (**Figure 6B**). Since the eyes are still visible, this is not a problem for the tracker. However, GT had to be partly manually annotated for this part of the recording, as two of the frame-based methods failed to detect the face that was too close to the camera. The subject then moves backwards and to the right, followed by further re-detections (**Figure 6C**).

3.3. Multiple Faces Detection

We recorded three sets of three subjects sitting at a desk talking to each other. No instructions where given to the subjects. **Figure 7** shows tracking results for the recording. The three subjects stay relatively still, but will look at each other from time to time as they are engaged in a conversation or sometimes focus on a screen in front of them. Lower detection rates (see **Table 2**) are caused by an increased pose variation, however this does not result in an increase of the tracking errors due to the absence of drift.

3.4. Pose Variation Sequences

The subjects in these sequences are rotating their head from one side to the other until only one eye is visible in the scene. Experiments show that the presence of a single eye does not affect the performances of the algorithm (see **Figure 8**). These experiments have been carried out with an event-based camera that has a VGA resolution. While this camera provides better temporal accuracy and spatial resolution, it does not provide gray-level events measurements.

Although we fed frames from the change detection events (which do not contain absolute gray-level information but are binary) to the frame-based methods, none of them could detect a face. This can be expected by the fact that the used networks have been trained on gray-level images. Perhaps if we re-train the last layers of the networks with manually labeled frames from change detection events (binary), they would probably achieve similar performances. However, the frame data set creation and the training are beyond the scope of this work.

3.5. Summary

Table 2 summarizes the relative accuracy of the detection and the tracking performances of the presented method, in comparison to VJ (Viola and Jones, 2004), FRCNN (Ren et al., 2015), SSD (Liu et al., 2016), and CF (Li and Shi, 2019). We set the correlation threshold to a value that is guaranteed to prohibit false positive detections, in order to (re-)initialize trackers at correct positions. The ratio of detected blinks is highest in controlled indoor conditions and detection rates in outdoor conditions are only slightly inferior. We attribute this fact to the aforementioned hardware limitations of earlier camera generations that are sensitive to lighting conditions. A lower detection rate for multiple subjects is mostly due to occluded blinks when subjects turn to speak to each other.

The tracking errors are the deviations from the frame-based bounding box center, normalized by the bounding box's width. The normalization provides a scale invariance so that errors estimated for a large bounding box from a close-up face have the same meaning as errors for a small bounding box of a face further away.

VJ, FRCNN, and SSD re-detect faces at every frame and discard face positions in previous frames, resulting in slightly erratic tracking over time. They do however give visually convincing results when it comes to accuracy, as they can detect a face right from the start of the recording and at greater pose variation given the complex model of a neural network. CF uses a tracker that updates its position at every frame that

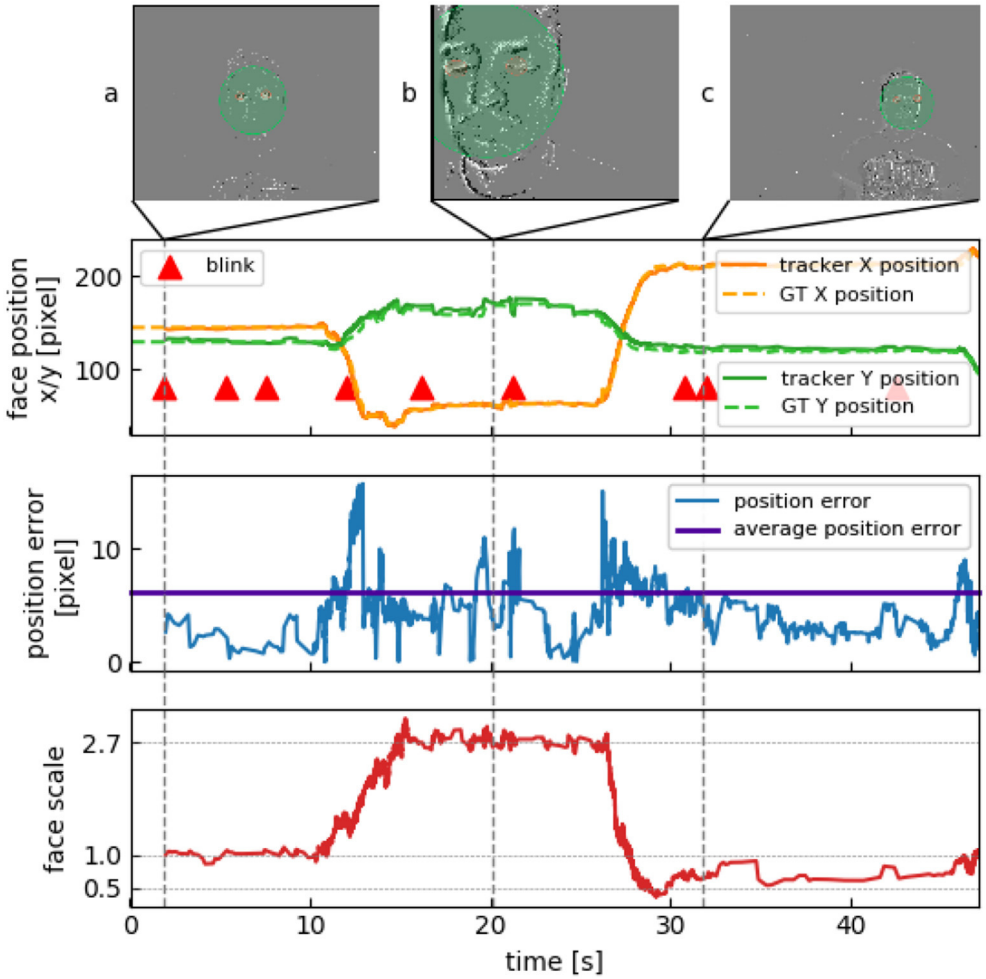


FIGURE 6 | Verifying resistance to scale. **(A)** First blink is detected at initial location. Scale value of 1 is assigned. **(B)** Subject gets within 25 cm of the camera, resulting in a three-fold scale change. **(C)** Subject veers away to about 150 cm, the face is now 35% smaller than in **(A)**.

is created from binning the change detection events, rather than working on gray-level frames. The tracker update at each frame based on the previous position ensures a certain spatial consistency and smoothness when tracking, at the temporal resolution of the frame rate. However, since a correlation filter was initially designed for classic (gray-level) images, it relies on visual information of the object to track to be present at all time, which is not necessarily the case for an event-camera.

The CF technique from Li and Shi (2019) requires the camera to move constantly in order to obtain visual information from the scene to maintain the tracking, as the algorithms uses rate-coded frames. This required us to modify their algorithm since in our data, tracked subjects can stop w.r.t. to the camera, hence they became invisible. We added a mechanism to the correlation filter that freezes the tracker's position when the object disappears. We use a maximum threshold of the peak-to-sidelobe ratio (Bolme et al., 2010), which measures the strength of a correlation peak and can therefore be used to detect occlusions or tracking failure while being able to continue the online update when the subject reappears. This results in delays in tracking whenever an object starts to move again and results in tracking penalties. CF has further limitations at tracking at high scale variance and cannot track multiple objects of the same nature at the same time.

4. CONCLUSION

We introduced a method able to perform face detection and tracking using the output of an event-based camera. We have shown that these sensors can detect eye blinks in real time. This detection can then be used to initialize a tracker and avoid drifts. The approach makes use of dynamical properties of human faces rather than relying on an approach that only uses static information of faces and therefore only their spatial structure.

The face's location is updated at μs precision once the trackers have been initialized, which corresponds to the native temporal resolution of the camera. Tracking and re-detection are robust to more than a five-fold scale, corresponding to a distance in front

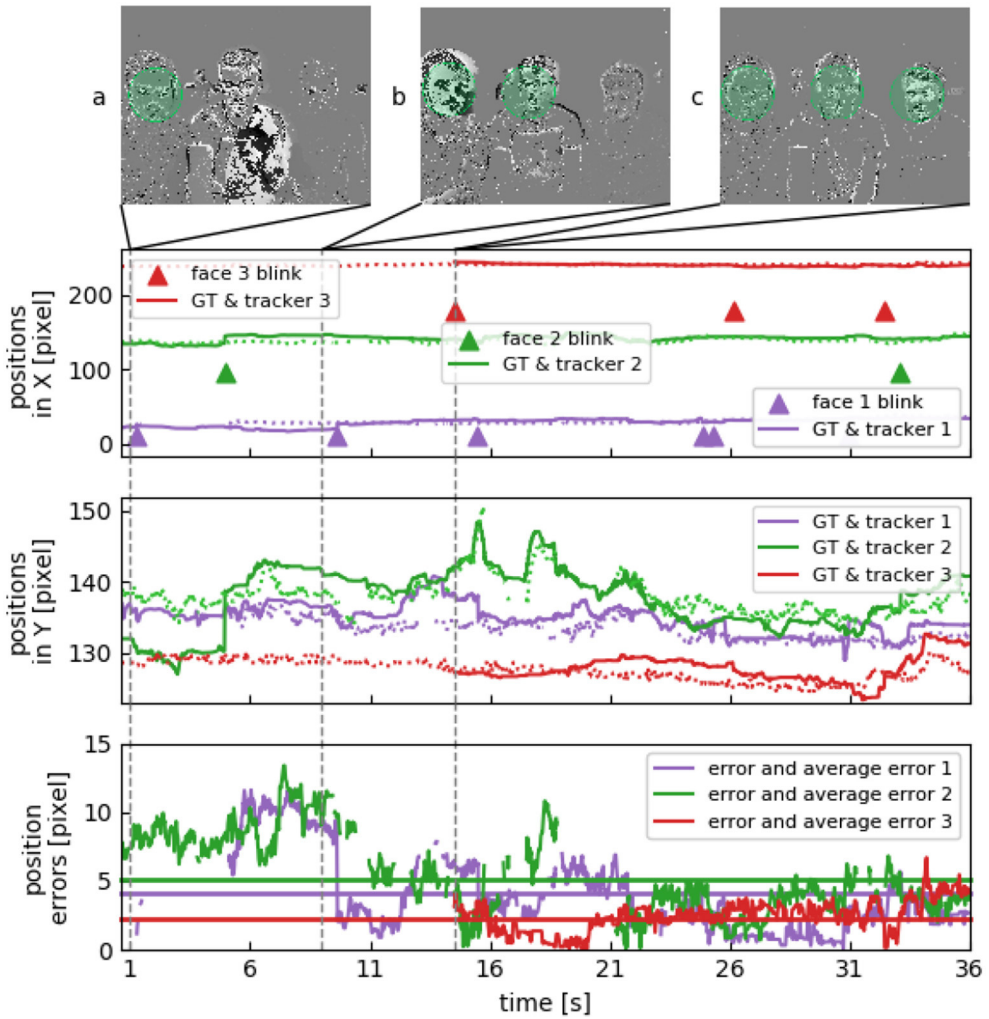


FIGURE 7 | Multiple face tracking in parallel. Face positions in X and Y show three subjects sitting next to each other, their heads are roughly on the same height. **(A)** Subject to the left blinks at first. **(B)** Subject in the center blinks next, considerably varying their face orientation when looking at the other two. **(C)** Third subject stays relatively still.

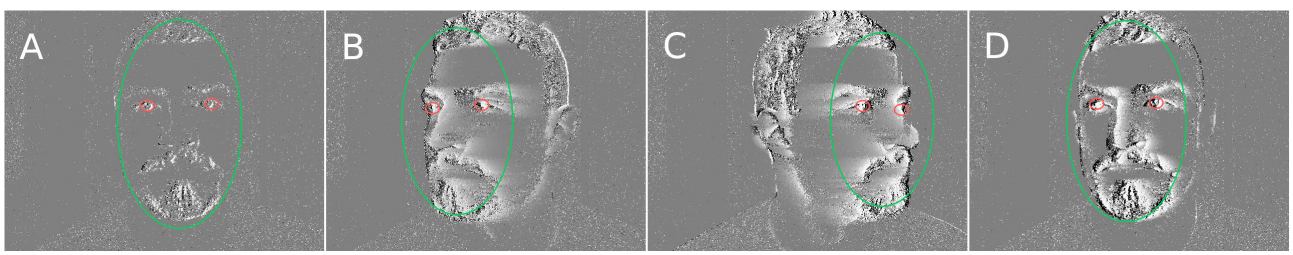


FIGURE 8 | Pose variation experiment. **(A)** Face tracker is initialized after blink. **(B)** Subject turns to the left. **(C,D)** One eye is occluded, but tracker is able to recover.

of the camera ranging from 25 cm to 1.50 m. A blink provides robust temporal signatures as its overall duration changes little from subject to subject.

The amount of events received and therefore the resulting activity amplitude varies only substantially when lighting of the scene is extremely different (i.e., indoor office lighting vs bright outdoor sunlight). The model generated from an initial set of manually annotated blinks has proven to be robust to those changes across a wide set of sequences. The algorithm mechanism is also robust to eye occlusions and can

still operate when face moves from side to side allowing only a single blink to be detected. In the most severe cases of occlusion, the tracker manages to reset correctly at the next detected blink.

The occlusion problem could be further mitigated by using additional trackers to track more facial features such as the mouth or the nose and by linking them to build a part-based model of a face as it has been tested successfully in Reverter Valeiras et al. (2015).

The blink detection approach is simple and yet robust enough for the technique to handle up to several faces simultaneously. We expect to be able to improve detection accuracy by learning the dynamics of blinks via techniques such as HOTS (Lagorce et al., 2016) or HATS (Sironi et al., 2018). At the same time with increasingly efficient event-based cameras providing higher spatial resolution, the algorithm is expected to increase its performance and range of operations. We roughly estimated the power consumption of the compared algorithms to provide numbers in terms of efficiency:

- The presented event-based algorithm runs in real-time using 70% of the resources of a single core of an Intel i5-7200U CPU for mobile Desktops, averaging to 5.5 W of power consumption while handling a temporal precision of $1\mu s$ (Intel Corporation, 2017).
- The OpenCV implementation of VJ is able to operate at 24 of the 25 fps in real-time, using a full core at 7.5 W (Intel Corporation, 2017).
- The FRCNN Caffe implementation running on the GPU uses 175 W on average on a Nvidia Tesla K40c with 4–5 fps.
- The SSD implementation in Tensorflow runs in real-time, using 106 W on average on the same GPU model.

DATA AVAILABILITY STATEMENT

The face detection dataset for this study can be found under https://www.neuromorphic-vision.com/public/downloads/data-set-face-detection.tar.gz.

ETHICS STATEMENT

Ethical review and approval was not required for the study on human participants in accordance with the local legislation and institutional requirements. Written informed consent to participate was not required in accordance with the national legislation and the institutional requirements. Written informed consent was obtained from the individuals for the publication of any potentially identifiable images or data included in this article.

AUTHOR CONTRIBUTIONS

GL, S-HI, and RB designed the algorithm. GL was responsible for data collection and programming. All authors participated in writing and editing the manuscript.

ACKNOWLEDGMENTS

This manuscript has been released as a Pre-Print at Lenz et al. (2018).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00587/full#supplementary-material

REFERENCES

Benedetto, S., Pedrotti, M., Minin, L., Baccino, T., Re, A., and Montanari, R. (2011). Driver workload and eye blink duration. *Transport. Res. Part F* 14, 199–208. doi: 10.1016/j.trf.2010.12.001

Bentivoglio, A. R., Bressman, S. B., Cassetta, E., Carretta, D., Tonali, P., and Albanese, A. (1997). Analysis of blink rate patterns in normal subjects. *Mov. Disord.* 12, 1028–1034. doi: 10.1002/mds.870120629

Bolme, D. S., Beveridge, J. R., Draper, B. A., and Lui, Y. M. (2010). "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (San Francisco, CA), 2544–2550. doi: 10.1109/CVPR.2010.5539960

Häkkänen, H., Summala, H., Partinen, M., Tiihonen, M., and Silvo, J. (1999). Blink duration as an indicator of driver sleepiness in professional bus drivers. *Sleep* 22, 798–802. doi: 10.1093/sleep/22.6.798

Intel Corporation (2017). *7th Generation Intel® Processor Family and 8th Generation Intel® Processor Family for U Quad Core Platforms Specification Data sheet.*

Jiang, H., and Learned-Miller, E. (2017). "Face detection with the faster R-CNN," in *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)* (Washington, DC), 650–657.

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

Lagorce, X., Orchard, G., Gallupi, F., Shi, B. E., and Benosman, R. (2016). HOTS: a hierarchy Of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 1346–1359. doi: 10.1109/TPAMI.2016.2574707

Lenz, G., Ieng, S.-H., and Benosman, R. (2018). High speed event-based face detection and tracking in the blink of an eye. *arXiv[preprint]arXiv:1803.10106.*

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 128 × 128 120 db 15 μ s latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits* 43, 566–576. doi: 10.1109/JSSC.2007.914337

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., et al. (2016). "SSD: Single shot multibox detector," in *European Conference on Computer Vision* (Amsterdam: Springer), 21–37.

Nakano, T., Kato, M., Morito, Y., Itoi, S., and Kitazawa, S. (2013). Blink-related momentary activation of the default mode network while viewing videos. *Proc. Natl. Acad. Sci. U.S.A.* 110, 702–706. doi: 10.1073/pnas.1214804110

Noman, M. T. B., and Ahad, M. A. R. (2018). Mobile-based eye-blink detection performance analysis on android platform. *Front. ICT* 5:4. doi: 10.3389/fict.2018.00004

Posch, C., Matolin, D., and Wohlgenannt, R. (2011). A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. *IEEE J. Solid-State Circ.* 46, 259–275. doi: 10.1109/JSSC.2010.2085952

Ren, J., Kehtarnavaz, N., and Estevez, L. (2008). "Real-time optimization of viola-jones face detection for mobile platforms," in *Circuits and Systems Workshop: System-on-Chip-Design, Applications, Integration, and Software, 2008 IEEE Dallas* (Dallas, TX), 1–4.

Ren, S., He, K., Girshick, R., and Sun, J. (2015). "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, 91–99.

Reverter Valeiras, D., Lagorce, X., Clady, X., Bartolozzi, C., Ieng, S.-H., and Benosman, R. (2015). An asynchronous neuromorphic event-driven visual part-based shape tracking. *IEEE Trans. Neural Netw. Learn. Syst.* 26, 3045–3059. doi: 10.1109/TNNLS.2015.2401834

Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., and Benosman, R. (2018). "Hats: Histograms of averaged time surfaces for robust event-based object classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 1731–1740.

Stern, J. A., Boyer, D., and Schroeder, D. (1994). Blink rate: a possible measure of fatigue. *Hum. Factors* 36, 285–297. doi: 10.1177/001872089403600209

Sun, X., Wu, P., and Hoi, S. C. (2018). Face detection using deep learning: an improved faster rcnn approach. *Neurocomputing* 299, 42–50. doi: 10.1016/j.neucom.2018.03.030

Viola, P., and Jones, M. J. (2004). Robust real-time face detection. *Int. J. Comput. Vis.* 57, 137–154. doi: 10.1023/B:VISI.0000013087.49260.fb

Walker, J. C., Kendal-Reed, M., Utell, M. J., and Cain, W. S. (2001). Human breathing and eye blink rate responses to airborne chemicals. *Environ. Health Perspect.* 109(Suppl. 4), 507–512. doi: 10.1289/ehp.01109s4507

Wang, Q., Yang, J., Ren, M., and Zheng, Y. (2006). "Driver fatigue detection: a survey," in *The Sixth World Congress on Intelligent Control and Automation, 2006, Vol. 2* (Dalian), 8587–8591.

Yang, S., Luo, P., Loy, C.-C., and Tang, X. (2016). "Wider face: a face detection benchmark," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 5525–5533.

Yang, S., Luo, P., Loy, C. C., and Tang, X. (2017). Faceness-net: face detection through deep facial part responses. *arXiv[preprint]arXiv:1701.08393*. doi: 10.1109/TPAMI.2017.2738644

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 Neuroscience

ORIGINAL RESEARCH
published: 05 August 2020
doi: 10.3389/fnins.2020.00637

Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing

Enea Ceolini[1†], Charlotte Frenkel[1,2†], Sumit Bam Shrestha[3†], Gemma Taverni[1], Lyes Khacef[4], Melika Payvand[1] and Elisa Donati[1]*

[1] Institute of Neuroinformatics, University of Zurich, ETH Zurich, Zurich, Switzerland, [2] ICTEAM Institute, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, [3] Temasek Laboratories, National University of Singapore, Singapore, Singapore, [4] Université Côte d'Azur, CNRS, LEAT, Nice, France

OPEN ACCESS

Edited by:
Fabio Stefanini, Columbia University, United States

Reviewed by:
Arren Glover, Italian Institute of Technology (IIT), Italy
Chetan Singh Thakur, Indian Institute of Science (IISc), India

***Correspondence:**
Elisa Donati
elisa@ini.uzh.ch

[†] *These authors have contributed equally to this work*

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *15 December 2019*
Accepted: *22 May 2020*
Published: *05 August 2020*

Citation:
Ceolini E, Frenkel C, Shrestha SB, Taverni G, Khacef L, Payvand M and Donati E (2020) Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing. Front. Neurosci. 14:637. doi: 10.3389/fnins.2020.00637

Hand gestures are a form of non-verbal communication used by individuals in conjunction with speech to communicate. Nowadays, with the increasing use of technology, hand-gesture recognition is considered to be an important aspect of Human-Machine Interaction (HMI), allowing the machine to capture and interpret the user's intent and to respond accordingly. The ability to discriminate between human gestures can help in several applications, such as assisted living, healthcare, neuro-rehabilitation, and sports. Recently, multi-sensor data fusion mechanisms have been investigated to improve discrimination accuracy. In this paper, we present a sensor fusion framework that integrates complementary systems: the electromyography (EMG) signal from muscles and visual information. This multi-sensor approach, while improving accuracy and robustness, introduces the disadvantage of high computational cost, which grows exponentially with the number of sensors and the number of measurements. Furthermore, this huge amount of data to process can affect the classification latency which can be crucial in real-case scenarios, such as prosthetic control. Neuromorphic technologies can be deployed to overcome these limitations since they allow real-time processing in parallel at low power consumption. In this paper, we present a fully neuromorphic sensor fusion approach for hand-gesture recognition comprised of an event-based vision sensor and three different neuromorphic processors. In particular, we used the event-based camera, called DVS, and two neuromorphic platforms, Loihi and ODIN + MorphIC. The EMG signals were recorded using traditional electrodes and then converted into spikes to be fed into the chips. We collected a dataset of five gestures from sign language where visual and electromyography signals are synchronized. We compared a fully neuromorphic approach to a baseline implemented using traditional machine learning approaches on a portable GPU system. According to the chip's constraints, we designed specific spiking neural networks (SNNs) for sensor fusion that showed classification accuracy comparable to the software baseline. These neuromorphic alternatives have increased inference time, between 20 and 40%,

with respect to the GPU system but have a significantly smaller energy-delay product (EDP) which makes them between 30× and 600× more efficient. The proposed work represents a new benchmark that moves neuromorphic computing toward a real-world scenario.

Keywords: hand-gesture classification, spiking neural networks (SNNs), electromyography (EMG) signal processing, event-based camera, sensor fusion, neuromorphic engineering

1. INTRODUCTION

Hand-gestures are considered a powerful communication channel for information transfer in daily life. Hand-gesture recognition is the process of classifying meaningful gestures of the hands and is currently receiving renewed interest. The gestural interaction is a well-known technique that can be utilized in a vast array of applications (Yasen and Jusoh, 2019), such as sign language translation (Cheok et al., 2019), sports (Loss et al., 2012), Human-Robot Interaction (HRI) (Cicirelli et al., 2015; Liu and Wang, 2018), and more generally in Human-Machine Interaction (HMI) (Haria et al., 2017). Hand-gesture recognition systems also target medical applications, where they are detected via bioelectrical signals instead of vision. In particular, among the biomedical signals, electromyography [Electromyography (EMG)] is the most used for hand-gesture identification and for the design of prosthetic hand controllers (Benatti et al., 2015; Donati et al., 2019; Chen et al., 2020).

EMG measures the electrical signal resulting from muscle activation. The source of the signal is the motor neuron action potentials generated during the muscle contraction. Generally, EMG can be detected either directly with electrodes inserted in the muscle tissue, or indirectly with surface electrodes positioned above the skin [surface EMG (sEMG), for simplicity we will refer to it as EMG]. The EMG is more popular for its accessibility and non-invasive nature. However, the use of EMG to discriminate between hand-gestures is a non-trivial task due to several physiological processes in the skeletal muscles underlying their generation.

One way to overcome these limitations is to use a multimodal approach, combining EMG with recordings from other sensors. Multi-sensor data fusion is a direct consequence of the well-accepted paradigm that certain natural processes and phenomena are expressed under completely different physical guises (Lahat et al., 2015). In fact, multi-sensor systems provide higher accuracy by exploiting different sensors that measure the same signal in different but complementary ways. The higher accuracy is achieved thanks to a redundancy gain that reduces the amount of uncertainty in the resulting information. Recent works show a growing interest toward multi-sensory fusion in several application areas, such as developmental robotics (Droniou et al., 2015; Zahra and Navarro-Alarcon, 2019), audio-visual signal processing (Shivappa et al., 2010; Rivet et al., 2014), spatial perception (Pitti et al., 2012), attention-driven selection (Braun et al., 2019) and tracking (Zhao and Zeng, 2019), memory encoding (Tan et al., 2019), emotion recognition (Zhang et al., 2019), multi-sensory classification (Cholet et al., 2019), HMI (Turk, 2014), remote sensing and earth observation (Debes et al., 2014), medical diagnosis (Hoeks et al., 2011), and understanding brain functionality (Horwitz and Poeppel, 2002).

In this study we consider the complementary system comprising of a vision sensor and EMG measurements. Using EMG or camera systems separately presents some limitations, but their fusion has several advantages, in particular EMG-based classification can help in case of camera occlusion, whereas the vision classification provides an absolute measurement of hand state. This type of sensor fusion which combines vision and proprioceptive information is intensively used in biomedical applications, such as in the transradial prosthetic domain, to improve control performance (Markovic et al., 2014, 2015), or to focus on recognizing objects during grasping to adjust the movements (Došen et al., 2010). This last task can also use Convolutional Neural Networks (CNNs) as feature extractors (Ghazaei et al., 2017; Gigli et al., 2018).

While improving accuracy and robustness, the multiple input modalities also increase the computational cost, due to the amount of data generated to process in real-time which can affect the communication between the subject and the prosthetic hand. Neuromorphic technology offers a solution to overcome these limitations providing the possibility to process multiple inputs in parallel in real-time, and with very low power consumption. Neuromorphic systems consist of circuits designed with principles based on the biological nervous systems that, similar to their biological counterparts, process information using energy-efficient, asynchronous, event-driven methods (Liu et al., 2014). These systems are often endowed with on-line learning abilities that allow adapting to different inputs and conditions. Lots of neuromorphic computing platforms have been developed in the past for modeling cortical circuits and their number is still growing (Benjamin et al., 2014; Furber et al., 2014; Merolla et al., 2014; Meier, 2015; Qiao et al., 2015; Moradi et al., 2017; Davies et al., 2018; Neckar et al., 2018; Thakur et al., 2018; Frenkel et al., 2019a,b).

In this paper we present a fully-neuromorphic implementation of sensor fusion for hand-gesture recognition. The proposed work is based on a previous work of sensor fusion for hand-gesture recognition, using standard machine learning approaches implemented in a cell phone application for personalized medicine (Ceolini et al., 2019b). The paper showed how a CNN performed better, in terms of accuracy, than a Support Vector Machine (SVM) on the hand-gesture recognition task. The novelty introduced here is that the sensor fusion is implemented on a fully neuromorphic system, from the event-based camera sensor to the classification phase, performed using three event-based neuromorphic circuits: Intel's Loihi research processor (Davies et al., 2018) and a combination

of the ODIN and MorphIC Spiking Neural Network (SNN) processors (Frenkel et al., 2019a,b). The two neuromorphic systems present different features, in particular, depending on the number of neurons available and on the input data, we implemented different SNN architectures. For example, for visual data processing, a spiking CNN is implemented in Loihi while a spiking Multi-Layer Perceptron (MLP) is chosen for ODIN + MorphIC (see section 2.3). For the case of EMG, the data was collected using the Myo armband that senses electrical activity in the forearm muscles. The data was later converted into spikes to be fed into the neuromorphic systems. Here, we propose a feasible application to show the neuromorphic performance in terms of accuracy, energy consumption, and latency (stimulus duration + inference time). The performance metric for the energy consumption is the Energy-Delay Product (EDP), a metric suitable for most modern processor platforms defined as the average energy consumption multiplied by the average inference time. The inference time is defined as the time elapsed between the end of the stimulus and the classification. To validate the neuromorphic results, we are comparing it to a baseline consisting of the network implemented, using a standard machine learning approach, where the inputs are fed as continuous EMG signals and video frames. We propose this comparison for a real case scenario as a benchmark, in order for the neuromorphic research field to advance into mainstream computing (Davies, 2019).

2. MATERIALS AND METHODS

In the following section, we describe the overall system components. We start from the description of the sensors used to collect the hand-gesture data, namely the event-based camera, Dynamic Vision Sensor (DVS), and the EMG armband sensor, Myo. We then describe the procedure with which we collected the dataset used for the validation experiments presented here and which is publicly available. Afterwards, the two neuromorphic systems under consideration, namely Loihi and ODIN + MorphIC, will be described, focusing on their system specifics, characteristics, and the model architectures that will be implemented on them. Finally, we describe the system that we call baseline and which represents the point of comparison between a traditional von-Neumann approach and the two neuromorphic systems.

2.1. DVS and EMG Sensors

2.1.1. DVS Sensor

The DVS (Lichtsteiner et al., 2006) is a neuromorphic camera inspired by the visual processing in the biological retina. Each pixel in the sensor array responds asynchronously to logarithmic changes in light. Whenever the incoming illumination increases or decreases above a certain threshold, it generates a polarity spike event. The polarity corresponds to the sign of the change; ON polarity for an increase in light, and OFF polarity for a decrease in light. The output is a continuous and sparse train of events, interchangeably called spikes throughout this paper, that carries the information of the active pixels in the scene (represented in **Figure 1**). The static information is directly removed on the hardware side and only the dynamic one, corresponding to the movements in the scene, is actually transmitted. In this way the DVS can reach low latency, down to 10 μs, reducing the power consumption needed for computation and the amount of transmitted data. Each spike is encoded using the Address Event Representation (AER) communication protocol (Deiss et al., 1999) and is represented by the address of the pixel (in x-y coordinates), the polarity (1 bit for the sign), and the timestamp (in microsecond resolution).

2.1.2. EMG Sensor

In the proposed work, we collected the EMG corresponding to hand gestures using the Myo armband by Thalmic Labs Inc. The Myo armband is a wearable device provided with eight equally spaced non-invasive EMG electrodes and a Bluetooth transmission module. The EMG electrodes detect signals from the forearm muscles activity and afterwards the acquired data is sent to an external electronic device. The sampling rates for Myo data are fixed at 200Hz and the data is returned as a unitless 8-bit unsigned integer for each sensor representing "activation" and does not translate to millivolts (mV).

2.2. DVS-EMG Dataset

The dataset is a collection of five hand gestures recorded with the two sensor modalities: muscle activity from the Myo and visual input, in the form of DVS events. Moreover, the dataset also provides the video recording using a traditional frame-based camera, referred to as Active Pixel Sensor (APS) in this paper. The frames from the APS are used as ground truth and as input in the baseline models. The APS-frames provided in the dataset are gray-scale, 240 × 180 resolution. The dataset contains recordings from 21 subjects: 12 males and nine females aged from 25 to 35 (see Data Availability Statement for the full access to the dataset). The structure is the following: each subject repeats three sessions, in each session the subject performs five hand gestures: *pinky*, *elle*, *yo*, *index*, and *thumb* (see **Figure 2**), repeated 5 times. Each single gesture recording lasts 2s. The gestures are separated by a relaxing time of 1s, to remove any residual activity from the previous gesture. Every recording is cut in 10 chunks of 200ms each, this duration was selected to match the requirements of a real-case scenario of low latency prosthesis control where there is a need for the classification and creation of the motor command within 250 ms (Smith et al., 2011). Therefore, the final number of samples results in 21 (subjects) × 3 (trials) × 5 (repetitions) × 5 (gestures) × 10 (chunks) for a total of 15,750. The Myo records the superficial muscle activity at the middle forearm from eight electrodes with a sampling rate of 200Hz. During the recordings, the DVS was mounted on a random moving system to generate relative movement between the sensor and the subject's hand. The hand remains static during the recording to avoid noise in the Myo sensor and the gestures are performed in front of a static white background, see **Figure 2** for the full setup.

2.2.1. Implementation on Neuromorphic Devices

SNNs, in general, and their implementation on neuromorphic devices require inputs as spike trains. In the case of the DVS, the

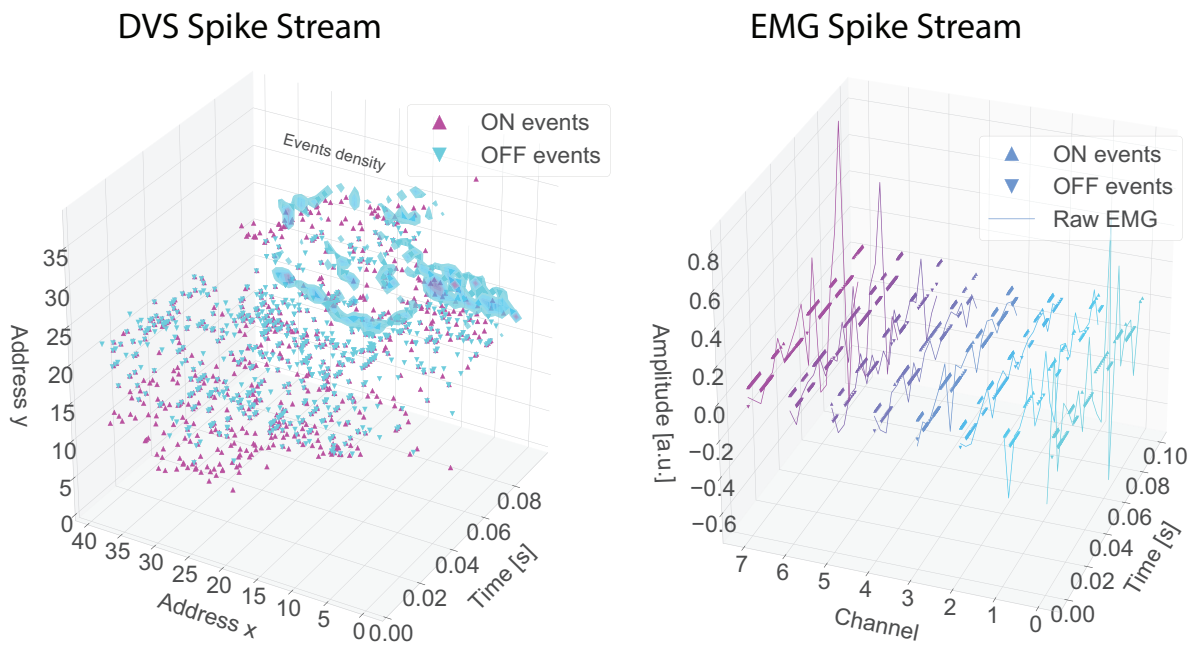


FIGURE 1 | Example, for a gesture "elle," of spike streams for DVS (left) and EMG (right). In the EMG figure the spikes are represented by dots while the continuous line is the raw EMG. Different channels have different colors.

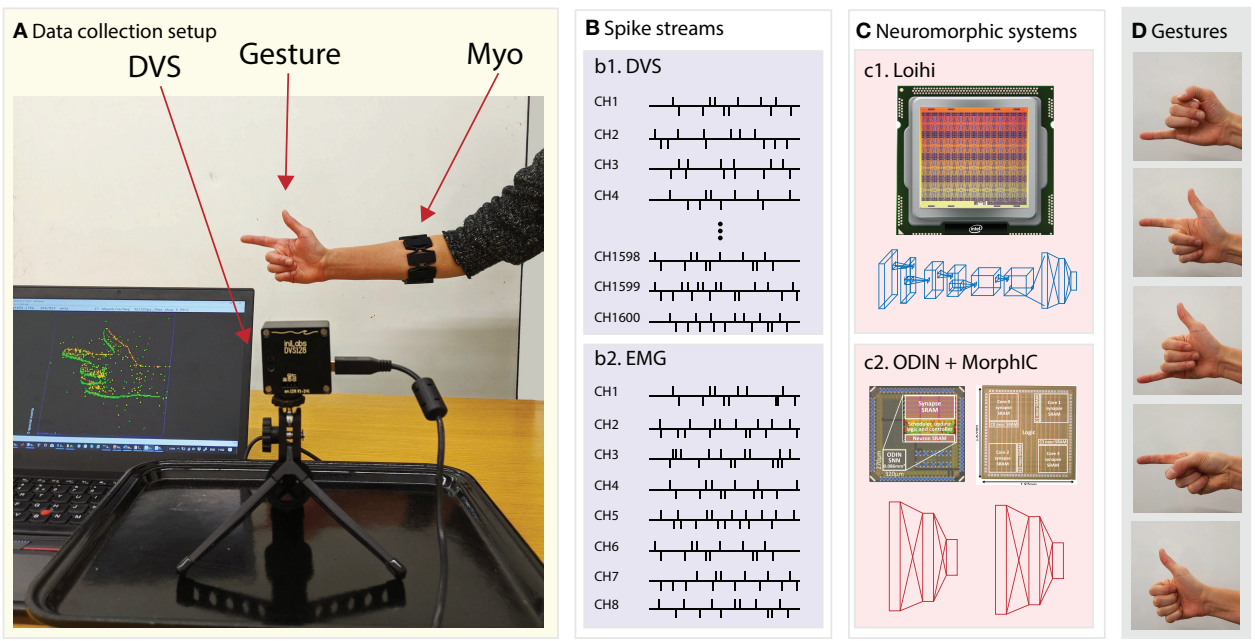


FIGURE 2 | System overview. From left to right: **(A)** data collection setup featuring the DVS, the traditional camera and the subject wearing the EMG armband sensor, **(B)** data streams of (b1) DVS and (b2) EMG transformed into spikes via the Delta modulation approach, **(C)** the two neuromorphic systems namely (c1) Loihi and (c2) ODIN + MorphIC, **(D)** the hand gestures that the system is able to recognize in real time.

sensor output is already in the form of spikes and polarity. The only requirement that we need to take into account is the limited number of neurons in the available neuromorphic processors. For this reason, we decided to crop the 128 × 128 input of the DVS to 40 × 40 centered on the hand-gesture. On the contrary, for the EMG, a conversion in the event-based domain is required.

The solution used here is the delta-modulator ADC algorithm, based on a sigma-delta modulator circuit (Corradi and Indiveri, 2015). This mechanism is particularly used in low frequency, high performance and low power applications (Lee et al., 2005), such as biomedical circuits. Moreover, this modulator represents a good interface for neuromorphic devices because it has much less circuit complexity and lower power consumption than multi-bit ADCs.

The delta-modulator algorithm transforms a continuous signal into two digital pulse outputs, UP or DOWN, according to the signal derivative. The UP (DOWN) spikes are generated every time the signal exceeds a positive (negative) threshold, like the ON (OFF) events from the DVS. As described before, the signal is sampled at 200Hz, this means that a new sample is acquired every 5 ms. To increase the time resolution of the generated spike train, which otherwise would contain too few spikes, the EMG signals are over-sampled to a higher frequency before undergoing the transformation into spikes (Donati et al., 2019).

For our specific EMG acquisition features, we set the threshold at 0.05 and an interpolation factor of 3500; these values have been selected from previous studies which looked at quality of signal reconstruction (Donati et al., 2018, 2019).

2.3. Neuromorphic Processors

2.3.1. ODIN + MorphIC

The ODIN (Online-learning DIgital spiking Neuromorphic) processor occupies an area of only 0.086 mm^2 in 28 nm FDSOI CMOS (Frenkel et al., 2019a)[1]. It consists of a single neurosynaptic core with 256 neurons and 256^2 synapses. Each neuron can be configured to phenomenologically reproduce the 20 Izhikevich behaviors of spiking neurons (Izhikevich, 2004). The synapses embed a 3-bit weight and a mapping table bit that allows enabling or disabling Spike-Dependent Synaptic Plasticity (SDSP) locally (Brader et al., 2007), thus allowing for the exploration of both off-chip training and on-chip online learning setups.

MorphIC is a quad-core digital neuromorphic processor with 2k LIF neurons and more than 2M synapses in 65nm CMOS (Frenkel et al., 2019b). MorphIC was designed for high-density large-scale integration of multi-chip setups. The four 512-neuron crossbar cores are connected with a hierarchical routing infrastructure that enables neuron fan-in and fan-out values of 1k and 2k, respectively. The synapses are binary and can be either programmed with offline-trained weights or trained online with a stochastic version of SDSP.

Both ODIN and MorphIC follow a standard synchronous digital implementation, which allows their operation to be predicted with one-to-one accuracy by custom Python-based chip simulators. As both chips rely on crossbar connectivity, CNN topologies can be explored but are limited to small networks due to an inefficient resource usage in the absence of a weight reuse mechanism (Frenkel et al., 2019b). The selected SNN architectures are thus based on fully-connected MLP topologies. Training is carried out in Keras with quantization-aware stochastic gradient descent following a standard ANN-to-SNN mapping approach (Hubara et al., 2017; Moons et al., 2017; Rueckauer et al., 2017), the resulting SNNs process the EMG and DVS spikes without further preprocessing.

In order to process the spike-based EMG gesture data, we selected ODIN so as to benefit from 3-bit weights. Indeed, due to the low input dimensionality of EMG data, satisfactory performance could not be reached with the binary weight resolution of MorphIC. A 3-bit-weight `16-230-5` SNN is thus implemented in ODIN, this setup will be referred to as the EMG-ODIN network.

For the DVS gesture data classification, we selected MorphIC, to benefit from its higher neuron and synapse resources. ON/OFF DVS events are treated equally and their connections to the network are learned, so that any of them can be either excitatory or inhibitory. Similarly to a setup previously proposed for MNIST benchmarking (Frenkel et al., 2019b), the input 40 × 40-pixel DVS event streams can be subsampled into four 20 × 20-pixel event streams and processed independently in the four cores of MorphIC, thus leading to an accuracy boost when combining the outputs of all subnetworks, subsequently denoted as subMLPs. The four subMLPs have a `400-210-5` topology with binary weights, this setup will thus be referred to as the DVS-MorphIC network.

To ease sensor fusion, the hidden layer sizes of the EMG-ODIN and DVS-MorphIC networks and the associated firing thresholds were optimized by parameter search so as to balance their activities. These hidden layers were first flattened into a 1,070-neuron layer, then a 5-neuron output layer was retrained with 3-bit weights and implemented in ODIN. This setup will be referred to as the Fusion-ODIN network, which thus encapsulates EMG processing in ODIN, DVS processing in MorphIC, and sensor fusion in ODIN. From an implementation point of view, mapping the MorphIC hidden layer output spikes back to ODIN as sensor fusion requires an external mapping table. Its overhead is excluded from the results provided in section 3.

2.3.2. Loihi and Its Training Framework SLAYER

Intel's Loihi (Davies et al., 2018) is an asynchronous neuromorphic research processor. Each Loihi chip consists of 128 neurocores, with each neurocore capable of implementing up to 1,024 current based (CUBA) Leaky Integrate and Fire (LIF) neurons. The network state and configuration is stored entirely in on-chip SRAMs local to each core, this allows each core to access its local memories independently of other cores without needing to share a global memory bus (and in fact removing the need for off-chip memory). Loihi supports a number of different encodings for representing network connectivity, thus allowing the user to choose the most efficient encoding for their task. Each Loihi chip also contains three small synchronous ×86 processors which help monitor and configure the network, as well as assisting with the injection of spikes and recording of output spikes.

SLAYER (Shrestha and Orchard, 2018) is a backpropagation framework for evaluating the gradient of any kind of SNN

[1]The HDL source code and documentation of ODIN are publicly available at https://github.com/ChFrenkel/ODIN.

[i.e., spiking MLP and spiking CNN] directly in the spiking domain. It is a dt-based SNN backpropagation algorithm that keeps track of the internal membrane potential of the spiking neuron and uses it during gradient propagation. There are two main guiding principles of SLAYER: temporal credit assignment policy and probabilistic spiking neuron behavior during error backpropagation. Temporal credit assignment policy acknowledges the temporal nature of a spiking neuron where a spike event at a particular time has its effect on future events. Therefore, the error credit of an error at a particular time needs to be distributed back in time. SLAYER is one of the few methods that consider temporal effects during backpropagation. The use of probabilistic neurons during backpropagation helps estimate the spike function derivative, which is a major challenge for SNN backpropagation, with the spike escape rate function of a probabilistic neuron. The end effect is that the spike escape rate function is used to estimate the spike function derivative, similar to the surrogate gradient concept (Zenke and Ganguli, 2018; Neftci et al., 2019). With SLAYER, we can train synaptic weights as well as axonal delays and achieve state of the art performances (Shrestha and Orchard, 2018) on neuromorphic datasets.

SLAYER uses the versatile Spike Response Model (SRM) (Gerstner, 1995) which can be customized to represent a wide variety of spiking neurons with a simple change of spike response kernels. It is implemented[2] atop the PyTorch framework with automatic differentiation support (Paszke et al., 2017) with the flexibility of feedforward dense, convolutional, pooling, and skip connections in the network.

SLAYER-PyTorch also supports training with the exact CUBA Leaky Integrate and Fire neuron model in Loihi (Davies et al., 2018). To train for the fixed precision constraints on weights and delays of Loihi hardware, it trains the network with the quantization constraints and then trains using the strategy of shadow variables (Courbariaux et al., 2015; Hubara et al., 2016) where the constrained network is used in the forward propagation phase and the full precision shadow variables are used during backpropagation.

We used SLAYER-PyTorch to train a Loihi compatible network for the hand-gesture recognition task. The networks were trained offline using GPU and trained weights and delays were used to configure the network on Loihi hardware for inference purposes. All the figures reported here are for inference using Loihi, with one algorithmic time tick in Loihi of 1 *ms*.

A spiking MLP of architecture `16-128d-128d-5` was trained for EMG gestures converted into spikes (section 2.2.1). Here, `128d` means the fully connected layer has 128 neurons with trained axonal delays. The Loihi neuron with current and voltage decay constants of 1,024 (32 ms) was used for this network.

For the gesture classification using DVS data we used both a spiking MLP, with the same architecture as the one deployed on MorphIC and described in section 2.3.1, and a spiking CNN with architecture `40x40x2-8c3-2p-16c3-2p-32c3-512-5`. Here, `XcY` denotes a convolution layer with `X` kernels of shape `Y-by-Y`, while `2p` denotes a `2-by-2` max pooling layer. Zero padding was applied for all convolution layers. No preprocessing on the spike events was performed, the ON/OFF events are treated as different input channels, hence the input shape `40x40x2`. For this network, current and voltage decay constants for the Loihi neurons were set to 1,024 (32 ms) and 128 (4 ms).

Finally, a third network where the penultimate layer neurons of DVS and EMG networks were fused together was trained. Only the last fully connected weights (`640-5`) were trained. The parameters of the network before fusion were preserved. The current and voltage decay constants of 1,024 (32 ms) and 128 (4 ms), respectively, were used for the final fusion layer neurons. From now on, we will refer to these three networks as EMG-Loihi, DVS-Loihi, and Fusion-Loihi whenever there is ambiguity.

2.4. Traditional Machine Learning Baselines

Machine Learning (ML) methods, and in general data-driven approaches, are currently the dominant tools used to solve complex classification tasks since they give the best performance compared to other approaches. We compare the performance of the two fully neuromorphic systems described in the above sections, against a traditional machine learning pipeline that uses frame-based inputs, i.e., traditionally sampled EMG signals and traditionally sampled video frames. For the comparisons to be fair, in the traditional approach we maintain the same constraints imposed by the neuromorphic hardware. In particular, we used the same neural network architectures as those used in the neuromorphic systems. Note that two different networks were implemented, spiking MLP and spiking CNN (see **Figure 3** for more details on the architectures). For this reason, we have two different baseline models that are paired to the two considered neuromorphic systems.

2.4.1. EMG Feature Extraction

Traditional EMG signal processing consists of various steps. First, signal pre-processing is used to extract useful information by applying filters and transformations. Then, feature extraction is used to highlight meaningful structures and patterns. Finally, a classifier maps the selected features to output classes. In this section we describe the EMG feature extraction phase, in particular we consider time domain features used for the classification of gestures with the baseline models. We extracted two time domain features generally used in literature (Phinyomark et al., 2018), namely Mean Absolute Value (MAV) and Root Mean Square (RMS) shown in Equation (1). The MAV is the average of the muscles activation value and it is calculated by a stride-moving window. The RMS is represented as amplitude relating to a gestural force and muscular contraction. The two features are calculated across a window of 40 samples, corresponding to 200 ms:

$$MAV(x_c) = \frac{1}{T}\sum_{t=0}^{T}|x_c(t)| \qquad RMS(x_c) = \sqrt{\frac{1}{T}\sum_{t=0}^{T}x_c^2(t)} \quad (1)$$

[2]SLAYER-PyTorch is publicly available at https://github.com/bamsumit/slayerPytorch.

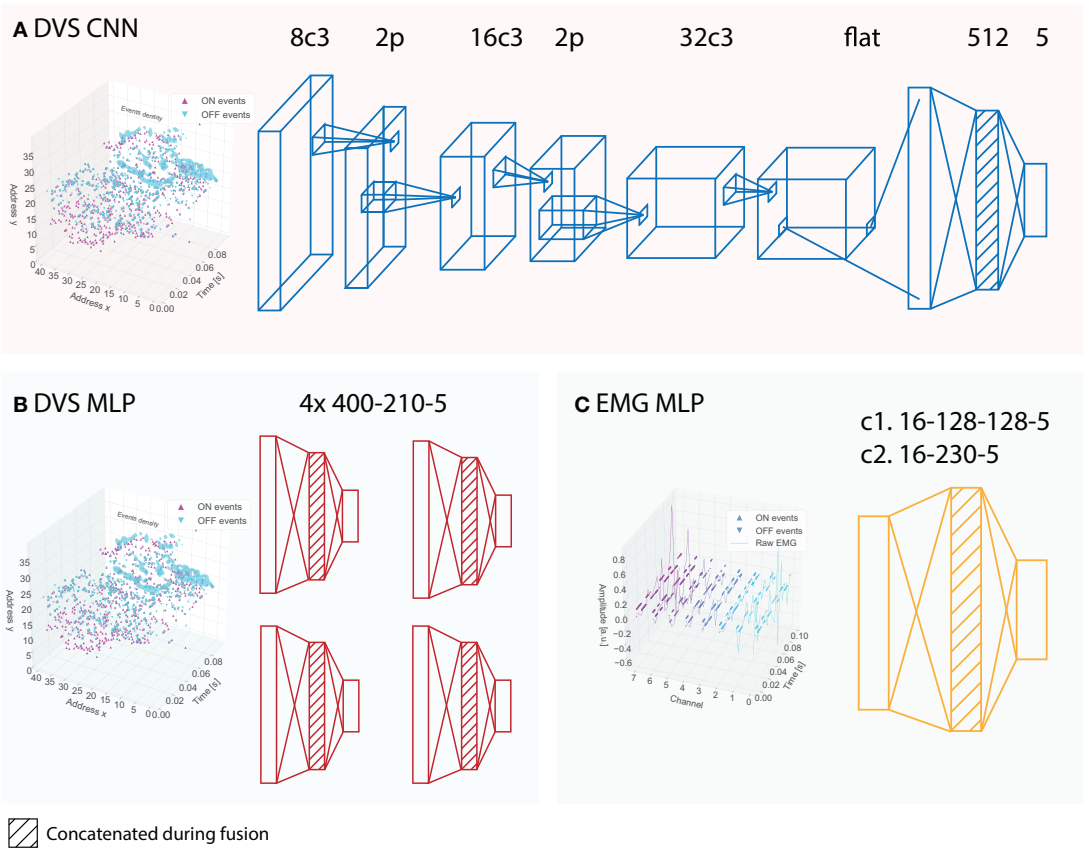


FIGURE 3 | Architectures of the neural networks implemented on the neuromorphic systems and used in the baselines. **(A)** CNN architecture implemented on Loihi; the corresponding baseline CNN receives APS frames instead of DVS events. **(B)** subMLP architectures implemented on MorphIC, the corresponding baseline subMLPs receive APS frames instead of DVS events. **(C)** MLP architecture for the EMG data implemented on Loihi (c1) and on ODIN (c2), the corresponding baseline MLPs receive EMG features instead of EMG events. The shading indicates those layers that are concatenated during the fusion of the networks.

where $x_c(t)$ is the signal in the time domain for the EMG channel with index c and T is the number of samples in the considered window, which was set to $T = 40$ ($N = 200$ ms) across this work. The features were calculated for each channel separately and the resulting values were concatenated in a vector $\mathbf{F}(n)$ described in Equation (2):

$$\mathbf{F}(n) = \left[F(x_1), \ldots, F(x_C)\right]^T \tag{2}$$

where $\mathbf{F}$ is MAV or RMS, n is the index of the window and C is the number of EMG channels. The final feature vector $\mathbf{E}(n)$ for window n is shown in Equation (3), it is used for the classification and is obtained by concatenating the two single feature vectors.

$$\mathbf{E}(n) = \left[\mathbf{MAV}(n)^T, \mathbf{RMS}(n)^T\right]^T \tag{3}$$

2.4.2. Baseline ODIN + MorphIC

As described in section 2.3.1, a CNN cannot be efficiently implemented on crossbar cores, which is the architecture ODIN and MorphIC rely on. We will therefore rely solely on fully-connected MLPs networks for both visual and EMG data processing. For the visual input, we used the same subMLP-based network structure as the one described in section 2.3.1, but with gray-scale APS frames. The 40 × 40 cropped APS frames are sub-sampled and fed into four 2-layer subMLPs of architecture `400-210-5`, as shown in **Figure 3B**. The outputs of the four subMLPs are then summed when classifying with a single sensor and are concatenated for the fusion network. The EMG neural network is a 2-layer MLP of architecture `16-230-5`. The fusion network is obtained as described above for the Loihi baseline.

2.4.3. Baseline Loihi

As described in section 2.3.2, we used a spiking MLP and a spiking CNN to process and classify DVS events. For the Loihi baseline, we kept the exact same architectures, except for the axonal delays. Moreover, both architectures of the baseline receive the corresponding gray-scale APS frames instead of the DVS events. The baseline MLP architecture and the CNN architectures are shown in **Figures 3A,B**, respectively. Note that the number of parameters between the baseline networks and the spiking networks implemented on Loihi is slightly different since the input has one channel (gray-scale) in the case of the baseline

TABLE 1 | Comparison of traditional and neuromorphic systems on the task of gesture recognition for both single sensor and sensor fusion.

System	Modality	Accuracy (%)	Energy (uJ)	Inference time (ms)	EDP (uJ * s)
Spiking CNN (Loihi)	EMG	55.7 ± 2.7	173.2 ± 21.2	5.89 ± 0.18	1.0 ± 0.1
	DVS	92.1 ± 1.2	815.3 ± 115.9	6.64 ± 0.14	5.4 ± 0.8
	EMG+DVS	96.0 ± 0.4	1104.5 ± 58.8	7.75 ± 0.07	8.6 ± 0.5
CNN (GPU)	EMG	68.1 ± 2.8	$(25.5 \pm 8.4) \cdot 10^3$	3.8 ± 0.1	97.3 ± 4.4
	APS	92.4 ± 1.6	$(31.7 \pm 7.4) \cdot 10^3$	5.9 ± 0.1	186.9 ± 3.9
	EMG+APS	95.4 ± 1.7	$(32.1 \pm 7.9) \cdot 10^3$	6.9 ± 0.05	221.1 ± 4.1
Spiking MLP (ODIN + MorphIC)	EMG	53.6 ± 1.4	7.42 ± 0.11	23.5 ± 0.35	0.17 ± 0.01
	DVS	85.1 ± 4.1	57.2 ± 6.8	17.3 ± 2.0	1.00 ± 0.24
	EMG+DVS	89.4 ± 3.0	37.4 ± 4.2	19.5 ± 0.3	0.42 ± 0.08
MLP (GPU)	EMG	67.2 ± 3.6	$(23.9 \pm 5.6) \cdot 10^3$	2.8 ± 0.08	67.2 ± 2.9
	APS	84.2 ± 4.3	$(30.2 \pm 7.5) \cdot 10^3$	6.9 ± 0.1	211.3 ± 6.1
	EMG+APS	88.1 ± 4.1	$(32.0 \pm 8.9) \cdot 10^3$	7.9 ± 0.05	253.0 ± 3.9

The results of the accuracy are reported with mean and standard deviation obtained over a 3-fold cross validation.

TABLE 2 | Inference statistics of Loihi models on 200 ms-long samples.

Network	Accuracy %	Core utilization	Dynamic power (mW)	Inference speedup
EMG-Loihi	55.74 ± 2.74	6	29.4 ± 3.6	$(34.01 \pm 1.01)\times$
DVS-Loihi	92.14 ± 1.23	95	109.0 ± 15.5	$(30.14 \pm 0.65)\times$
Fusion-Loihi	96.04 ± 0.48	100	137.2 ± 7.3	$(25.82 \pm 0.24)\times$

that uses APS frames while it has two channels (polarity) in the input for Loihi.

The MLP architecture used for the EMG classification is instead composed of two layers of 128 followed by one layer of 5 units. While the input stays of the same size (16) with respect to the network implemented on Loihi, the input features are different since the baseline MLP receives MAV and RMS features while the Loihi receives spikes obtained from the raw signal.

To obtain the fusion network, we eliminate the last layer (classification layer) from both the single sensor networks, concatenate the two penultimate layers of the single sensor networks, and add a common classification layer with five units, one per each class.

2.4.4. Training and Deployment

The models are trained with Keras using Adam optimizer with standard parameters. First, the single modality networks are trained separately, each for 30 epochs. For sensor fusion, output layer retraining is also carried out for 30 epochs. In order to compare the baselines against the neuromorphic systems in terms of energy consumption and inference time, we deployed the baseline models onto the NVIDIA Jetson Nano, an embedded system with a 128-Core Maxwell GPU with 4GB 64-bit LPDDR4 memory 25.6 GB/s[3].

[3]https://developer.nvidia.com/embedded/jetson-nano-developer-kit

3. RESULTS

Table 1 summarizes the results for Loihi and ODIN+MorphIC with the respective baselines. More details are described in the following sections.

3.1. Loihi Results

The classification performances of these three networks, EMG-Loihi, DVS-Loihi, and Fusion-Loihi, with 3-fold cross-validation and inferenced using 200 *ms* data, are tabulated in **Table 2**. The core utilization, dynamic power consumption, and inference time in the Loihi hardware are also listed in **Table 2**. The dynamic power is measured as the difference of total power consumed by the network and the static power when the chip is idle. Since one algorithmic time tick is 1ms long, inference time represents the speedup factor compared to real time.

With the spiking MLP implemented on Loihi, we obtained an accuracy of 50.3 ± 1.5, 83.1 ± 3.4, and $83.4 \pm 2.1\%$ for the hand-gesture classification task using EMG, DVS and fusion, respectively. Being that these results were significantly worse than the ones obtained with the spiking CNN, we do not report them in **Tables 1**, **2** and prefer to focus our analysis on the CNN which is better suited for visual tasks. This poor performance is due to temporal resolution of Loihi that causes a drop in the number of spikes in the MLP architecture while this does not happen in the CNN architecture.

The EMG network does not perform as well as in the baseline as shown in **Table 1**. The reason for this discrepancy can be found in the fact that the baseline method uses EMG from the raw signal of the sensor. However, to process this signal using neuromorphic chips (Loihi and ODIN + MorphIC), the EMG signal is encoded into spikes. With this encoding, part of the information is lost (as is the case for any encoding). Therefore, the baseline method has the advantage of using a signal that has more information and thus it outperforms the neuromorphic approach. Note that these

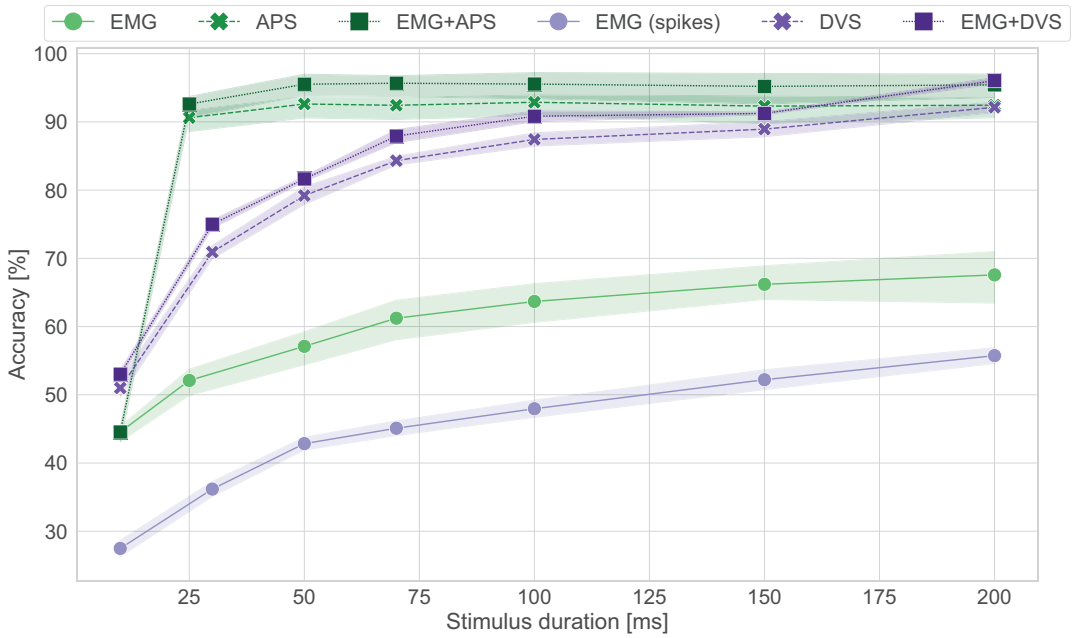


FIGURE 4 | Accuracy vs. stimulus duration for the Loihi system and its software baseline counterpart. In green the results for the CNN (GPU), in purple the results for the spiking CNN (Loihi). No classification is present for APS frames before 25 ms since the frame rate is 20 fps.

Loihi networks are restricted to 8-bit fixed precision weights and 6-bit fixed precision delays.

To evaluate the performance over time of the Loihi networks, stimulus duration vs. testing accuracy is plotted in **Figure 4**. We can see that the EMG-Loihi network continues to improve with longer stimulus duration. **Table 1** and **Figure 4** show the results of the Loihi baseline. From an accuracy point of view the baseline reaches a higher classification accuracy only in the EMG classification, while both the visual classification and fusion are on par with the Loihi networks and show only a non-significant difference. In terms of inference time, the baseline running on the GPU system is systematically faster than Loihi, but never more than 40% faster. As expected, the energy consumption of the GPU system is significantly higher than the Loihi system. Loihi is around 30× more efficient than the baseline with concern to the fusion network and more than 150× and 40× more efficient with concern to the EMG and DVS processing, respectively. **Figure 4** shows in more details the effect of stimulus duration on the classification accuracy. As expected, EMG is the modality that suffers more from classification based on short segments (Smith et al., 2011), reaching the best accuracy only after 200 ms for both the neuromorphic system and the baseline, while the accuracy for vision and fusion modalities saturate much more quickly, in around 100 ms for the neuromorphic system and 50 ms for the baseline. The traditional system reaches its best performance after 50 ms while the neuromorphic system reaches its best performance after 200ms. One should, however, also note that the DVS sensor contains only the edge information of the scene whereas the baseline network uses the image frame. Therefore, the spiking CNN requires some time to integrate the input information from DVS. Despite the inherent delays in a spiking CNN, the Loihi CNN can respond to the input within a few ms of inputs. However, for the vision modality, notice that, because the frame rate of the camera is 20 fps, there is no classification before 25ms. Therefore, for short stimulus duration, the neuromorphic system has higher accuracy than the traditional system.

3.2. ODIN + MorphIC Results

Inference statistics for a 200 ms sample duration are reported in **Table 3** for the EMG-ODIN, DVS-MorphIC, and Fusion-ODIN networks. Chip utilization is computed as the percentage of neuron resources taken by the hidden and output layers in ODIN and MorphIC, while the power consumption P of the crossbar cores of both chips can be decomposed as

$$P = P_{\text{leak}} + P_{\text{idle}} f_{\text{clk}} + E_{\text{SOP}} r_{\text{SOP}}, \tag{4}$$

where P_{leak} is the chip leakage power and $P_{\text{leak}} + P_{\text{idle}} f_{\text{clk}}$ represents the static power consumption when a clock of frequency f_{clk} is connected, without network activity. The term $E_{\text{SOP}} r_{\text{SOP}}$ thus represents the dynamic power consumption, where E_{SOP} is the energy per synaptic operation (SOP) and r_{SOP} is the SOP processing rate, each SOP taking two clock cycles. Detailed power models extracted from chip measurements of ODIN and MorphIC are provided in Frenkel et al. (2019a,b), respectively. The results reported in **Tables 1**, **3** are obtained with ODIN and MorphIC optimizing for power, under the conditions summarized in **Table 4**. The dynamic power consumption reported in **Table 4** reflects the regime in which ODIN and the four cores of MorphIC run at the maximum SOP processing rate $r_{\text{SOP}} = f_{\text{clk}}/2$.

A limitation of the crossbar-based architecture of ODIN and MorphIC is that each neuron spike leads to a systematic

TABLE 3 | Inference statistics of ODIN and MorphIC models on 200 ms-long samples.

Network	Accuracy (%)	Chip utilization (%)		Dyn. power (mW)		Processing time (ms)		Inference speedup
		ODIN	MorphIC	ODIN	MorphIC	ODIN	MorphIC	
EMG-ODIN	53.65 ± 1.37	91.8	–	0.315	–	23.5	–	8.5×
DVS-MorphIC	85.17 ± 4.11	–	42.0	–	3.3	–	17.3	11.6×
Fusion-ODIN	89.44 ± 3.02	91.8	41.0	0.315	3.3	19.5	9.5	10.3×

TABLE 4 | Low-power operating conditions of ODIN and MorphIC at minimum supply voltage.

Chip	Supply voltage (V)	E_{SOP} (pJ)	Max. f_{clk} (MHz)
ODIN	0.55	8.4	75
MorphIC	0.8	30	55

processing of all neurons in the core, thus potentially leading to a significant amount of dummy operations (Frenkel et al., 2019b). Taking the example of the DVS-MorphIC network with a crossbar core of 512 neurons (**Figure 3B**), each input spike leads to 512 SOPs, of which only 210 are useful for hidden layer processing. Similarly, each spike from a hidden layer neuron leads to 512 SOPs, of which only five are actually used for output layer processing. The induced overhead is thus particularly critical for output layer processing, which degrades both the energy per inference and the inference time[4]. However, this problem is partly mitigated in the Fusion-ODIN network for output layer processing. Indeed, when resorting to an external mapping table (section 2.3.1), hidden layer spikes can be remapped back to the sensor fusion output layer of ODIN with specific single-SOP AER events (Frenkel et al., 2019a), thus avoiding the dummy SOP overhead and leading to a lower energy and inference time compared to the standalone EMG-ODIN and DVS-MorphIC networks (**Tables 1**, **3**). As described in section 2.3.1, the fusion results exclude the mapping table overhead.

The comparison of the results obtained with ODIN + MorphIC to those obtained with its GPU baseline counterpart (**Table 1** and **Figure 5**) leads to conclusions similar to those already drawn with Loihi in section 3.1, with the difference that while the GPU system is significantly faster, between 2× and 10× faster, the ODIN + MorphIC neuromorphic system is between 500× and 3,200× more energy-efficient. Moreover, it appears from **Figure 5** that the EMG-ODIN, DVS-MorphIC and Fusion-ODIN networks basically perform at chance level for a 10-ms stimulus duration. This comes from the fact that the firing thresholds of the networks were selected based on a 200-ms stimulus duration, which leads the output neurons to remain silent and never cross their firing threshold when insufficient input spike data is provided. This problem could be alleviated by reducing the neuron firing thresholds for shorter stimulus durations.

[4]As discussed in (Frenkel et al., 2019b), a simple extension providing post-synaptic start and end addresses would avoid these dummy SOPs and allow for an efficient processing of fully-connected layers, which will be included in future generations of the chips.

3.3. EDP and Computational Complexity

Figure 6 shows a comparison between the Loihi system and the ODIN + MorphIC system in terms of EDP, number of operations per classification and a ratio between these two quantities. While panel (a) reports the same numbers as in **Table 1**, panels (b) and (c) allow for a more fair comparison of energy consumption between the two neuromorphic systems. From panel (b), we can see how the number of operations is similar for the EMG networks, both being MLPs for the two neuromorphic systems. Differently, the number of operations for the visual input and the fusion differ substantially between the two systems due to the use of a CNN in the Loihi system. Taking this into account, we can see in panel (c) that the normalized energy consumption tends to be similar for both systems, more than the EDP in panel (a) is.

4. DISCUSSIONS

As it has been discussed in Davies (2019), there is a real need for a benchmark in the neuromorphic engineering field to compare the metrics of accuracy, energy, and latency. ML benchmarks, such as ImageNet for image classification (Deng et al., 2009), Chime challenges for speech recognition (Barker et al., 2015), and the Ninapro dataset containing kinematic and surface EMG for prosthetic applications (Atzori et al., 2014) are not ideal for neuromorphic chips as they require high performance computing for processing. For example, floating point bit resolution, large amounts of data and large power consumption. There have been some efforts in creating relevant event-based datasets, such as N-MNIST (Orchard et al., 2015), the spiking version of the widespread MNIST digits recognition dataset, N-TIDIGITS18 (Anumula et al., 2018), the spiking version of the spoken digits recognition dataset from LDC TIDIGITS, and the DVS gesture recognition dataset from IBM (Amir et al., 2017). These datasets are either toy examples or are not meant for real-world applications. Here, we are introducing a hand gesture benchmark in English sign language (e.g., ILY) using the DVS and Myo sensors. This kind of benchmark can be directly used as a preliminary test for Brain-Machine Interface (BMI)/personalized medicine applications. We have collected this dataset from 21 people and in this paper have benchmarked it on three digital neuromorphic chips, measuring the accuracy, energy, and inference time. We believe this work takes an important first step in the direction of a real use-case (e.g., rehabilitation, sports applications, and sign

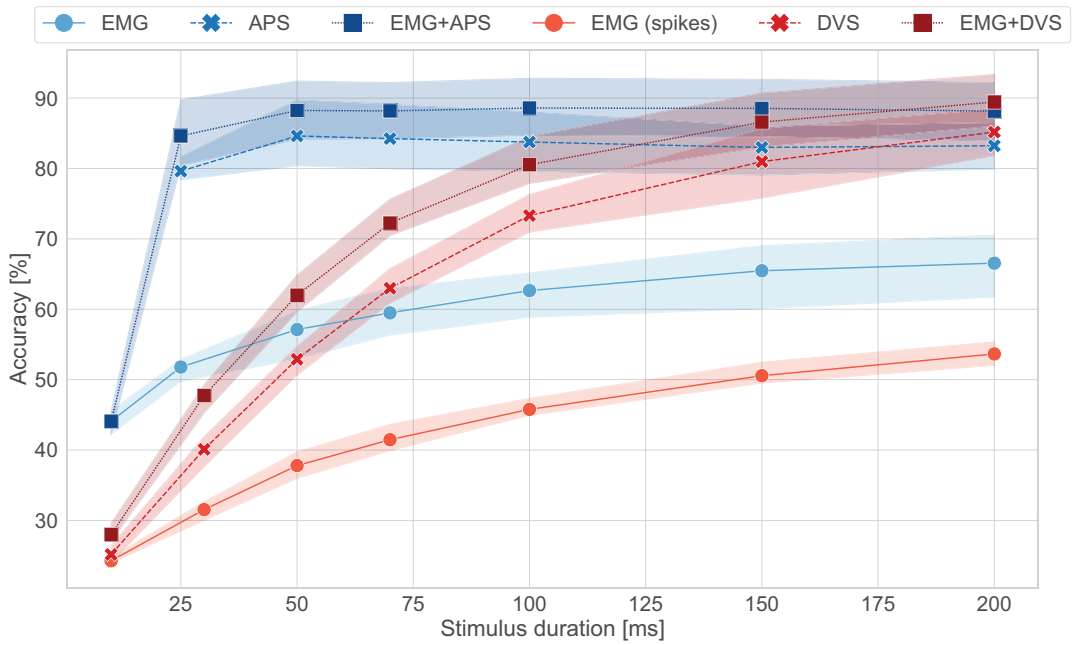


FIGURE 5 | Accuracy vs. stimulus duration for the ODIN + MorphIC system and its software baseline counterpart. In blue the results for the MLP (GPU), in red the results for the spiking MLP (ODIN + MorphIC). No classification is present for APS frames before 25 ms since the frame rate is 20 fps.

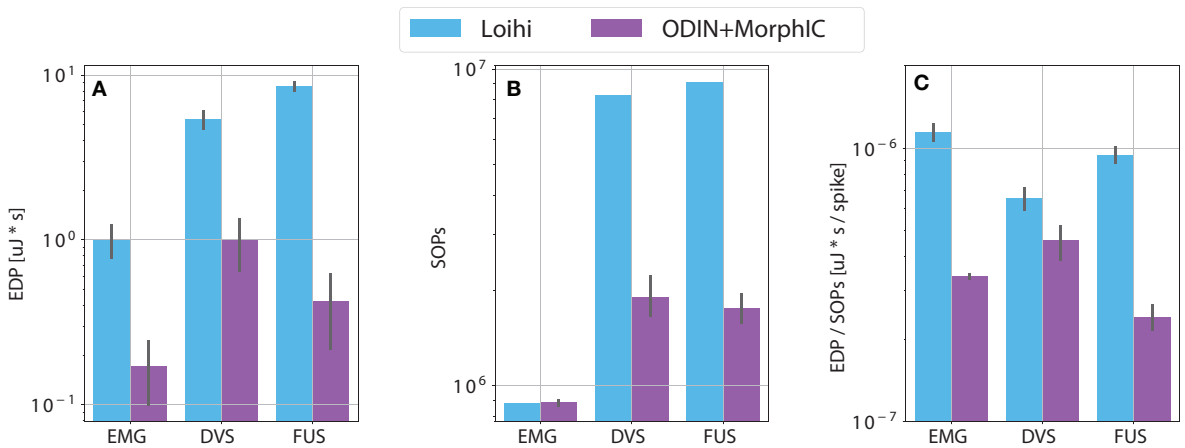


FIGURE 6 | Comparison between the two neuromorphic system with respect to **(A)** energy delay product (EDP) (see section 1), **(B)** number of synaptic operations (SOPs) (see section 2.3.1), **(C)** EDP normalized by the number of SOPs.

interpretation) which we would like to encourage the community to use.

Although the dataset we provided is on static gestures, the DVS and the spiking EMG signals provide the capability for low-power processing using event-based neuromorphic chips and enable embedded systems with online on-site processing without having to send the data to remote sensors. Therefore, this work is an important first step toward edge-computing applications. The static dataset also helps with reducing the noise from the EMG signals as we mentioned in section 2.2. However, this does not move away from the real application as we have shown in a live demo in Ceolini et al. (2019a).

The selected multi-sensor data fusion, which combines vision and EMG sensors, derives from the need of multiple sources to help the classification in real-scenario cases. Although the results show a small improvement due to the EMG sensors, they still provide some classification in case light conditions or camera occlusions are not ideal. In addition,

for specific applications, such as neuroprosthetic control, the EMG is integrated in the prosthetic device and, eventually, the camera can act as a support input helping during calibration or more advanced tasks, such as sensory-motor closed loop (Jiang et al., 2012).

Since the event-based neuromorphic chips require inputs in the form of events, the continuous sensory signals have to be encoded into spikes for an event-driven processing. This quantization loses information (and hence accuracy) in comparison to the analog information processing in trade-off with the low power consumption of event-based systems which is required for edge computing. To compensate for the loss of information and accuracy, it is important to merge information from multiple sensors in a sensory fusion setup. In this setting, the information loss by quantization from one sensor can be made up for by another one. This is similar to how humans and animals perceive their environment through diverse sensory channels: vision, audition, touch, smell, proprioception, etc. From a biological perspective, the fundamental reason lies in the concept of degeneracy in neural structures (Edelman, 1987), which means that any single function can be carried out by more than one configuration of neural signals, so that the biological system still functions with the loss of one component. It also means that sensory systems can educate each other, without an external teacher (Smith and Gasser, 2005). The same principles can be applied for artificial systems, as information about the same phenomenon in the environment can be acquired from various types of sensors: cameras, microphones, accelerometers, etc. Each sensory-information can be considered as a modality. Due to the rich characteristics of natural phenomena, it is rare that a single modality provides a complete representation of the phenomenon of interest (Lahat et al., 2015).

There are mainly two strategies for multi-modal fusion in the literature (Cholet et al., 2019): (1) data-level fusion (early fusion) where modalities are concatenated then learned by a unique model, and (2) score-level fusion (late fusion) where modalities are learned by distinct models and only after their predictions are fused with another model that provides a final decision. Early fusion, including feature-level fusion, suffers from a compatibility problem (Peng et al., 2016) and does not generalize well. Additionally, neural-based early fusion increases the memory footprint and the computational cost of the process, by inducing a full connectivity at the first classification stages. It is an important factor to take into consideration when choosing a fusion strategy (Castanedo, 2013), especially for embedded systems. Therefore, we follow a late fusion approach with a classifier-level fusion, which has been shown to perform better than feature-level fusion for classification tasks (Guo et al., 2014; Peng et al., 2016; Biagetti et al., 2018). It is close to score-level fusion by combining the penultimate layers of the base (unimodal) classifiers in a meta-level (multimodal) classifier that uses the natural complementarity of different modalities to improve the overall classification accuracy.

In this context, to have a fair comparison, the central question is the difference between the completely traditional approaches, such as the CNN and MLP baselines, vs. the event-based neuromorphic one. In the baseline, the EMG features are manually extracted, and the classification is done on the extracted features. Note that this pipeline is completely different from the event-based neuromorphic approach which extracts the features directly from the events. Another important thing to mention here is that although we have encoded the signals separately, this sensory information can be directly encoded to events at the front-end. This has already been established for audio and visual sensors (Lichtsteiner et al., 2006; Chan et al., 2007) and there have also recently been design efforts for other signals such the biomedical ones (Corradi and Indiveri, 2015).

To have a reference point for comparison, we trained the same network architecture used for the two neuromorphic setups. As can be seen in **Table 1**, the baseline accuracy on the fusion is on par with both Loihi and ODIN + MorphIC, despite the lower bit resolution on the neuromorphic chips in comparison with the 32-bit floating point resolutions on GPU in the baseline approach. We speculate that this is because the SLAYER training model already takes into account the low bit precision and thus calculates the gradients, respectively. Similar to that, ODIN and MorphIC take a quantization-aware training approach which calculates the weights based on the available on-chip precision. As can be seen from all the experiments in **Table 1**, the classification accuracy using only the EMG sensor is relatively low. However, it should be noted that this is the result of having a model which is trained across subjects and there are multiple sources of variability across subjects: (i) The placement of the EMG sensor is not necessarily in the same position (with respect to the forearm muscles) for every subject. (ii) Every subject performs the gestures in a unique manner. (iii) The muscle strength is different for every subject. In addition, since the EMG is directly measured from surface electrodes, it acquires noise while traveling through the skin, background noise from electronics, ambient noise, and so forth. In a real-world application, the network model can be trained on a single subject's data, yielding much higher accuracy. Moreover, having the online learning abilities on the neuromorphic chip can aid in adapting these models to every subject uniquely. Such online learning modules already exist in Loihi as well as in ODIN and MorphIC, which can be exploited in the future to boost the classification accuracy of EMG signals. Furthermore, it becomes apparent that the fusion accuracy is close, if not higher, at about 4% to the accuracy achieved with the DVS single sensor. However, the importance of the EMG signal is in the wearable application since it is a natural way to control prosthesis and it is a direct measure of the activity and movement in the muscles. Given the noisy nature of the EMG signal, it is critical to combine it with the visual input to boost the accuracy. But even given the noisy nature of the signal, it still allows to retrieve relevant information which helps boosting the accuracy of the fusion.

It is worth noting that while the accuracy between the spiking MLP on Loihi and ODIN + MorphIC are directly comparable, the results regarding the spiking CNN on Loihi and the spiking MLP on ODIN + MorphIC are not. This is because the two architectures use different features and resources on their respective neuromorphic systems (as already described in section 2.3). Based on this, there are different constraints

present in the two chips. Traditionally, a CNN architecture is used for image classification which is the network we used on the Loihi chip, given the large number of neurons that are available (128k) on this general-purpose platform. However, since ODIN and MorphIC are small-scale devices compared to Loihi, the number of neurons are a lot more constrained (i.e., 256 neurons for ODIN, 2k for MorphIC). Therefore, we resorted to using a fully-connected MLP topology instead of a CNN for image classification in MorphIC.

Regarding the latency, it is important to mention that for real-world prosthetic applications, the latency budget is below 250 ms (Smith et al., 2011). This means that if the processing happens within this budget, the patient will not feel the lag of the system. Hence, optimizing the system for having lower latency than 200 ms will not be beneficial as the patient will not feel the latency below 200 ms. Therefore, within this budget, other parameters can be optimized. The neuromorphic approach is very advantageous in this case since it trades-off power with latency, but it stays within the latency budget that is required. Contrarily, the GPU system has an overall faster inference time but uses much more energy. It is worth mentioning that our results are reported in accelerated time, however, the EMG and DVS are slowly changing signals, and thus, even though the classification is done very fast, the system has to wait for the inputs to arrive. Therefore, it is as if the system is being run in real-time. Here, there is a trade-off between the memory that is storing the streaming data for processing and the dynamic energy consumption. The accelerated time allows for lower energy consumption as the system is on for a shorter time, however, this comes with the caveat that the input has to be buffered for at least 200 ms in off-chip memory, therefore inducing a power and resource overhead.

The final comparison provided by **Figure 6** shows how the two systems have a similar energy consumption when this is normalized by the number of operations done to run the network and obtain one classification output. While ODIN + MorphIC consumes less per classification in absolute terms, when considering the number of operations, it performs comparably to Loihi. When deploying a neuromorphic system, one has to take into account all these aspects. Meaning not only is there a trade-off between speed and energy consumption but there is also one between accuracy and energy consumption, given the fact that a more complex network architecture may have more predictive power while having a higher energy demand. Overall, one has to look for the best trade-off in the context of a particular application, the malleability of neuromorphic hardware enables this adaptation to the task-dependent constraints within a framework of state of the art results with respect to system performance.

DATA AVAILABILITY STATEMENT

The datasets analyzed for this study can be found in the Zenodo, open access repository, http://doi.org/10.5281/zenodo.3663616. All the code used for the reported experiments can be found at https://github.com/Enny1991/dvs_emg_fusion.

AUTHOR CONTRIBUTIONS

EC, CF, and SS contributed equally to the work. EC, GT, MP, and ED participated equally to the development of the work idea and collected the dataset. EC and LK were responsible for the baseline experiments. CF and SS implemented the ODIN + MorphIC and Loihi pipelines, respectively. SS implemented the SLAYER framework and adapted it for the specific application. All authors contributed to the writing of the paper.

FUNDING

This work was supported by the EU's H2020 MSC-IF grant NEPSpiNN (Grant No. 753470), the Swiss Forschungskredit grants FK-18-103 and FK-l9-106, the Toshiba Corporation, the SNSF grant No. 200021_172553, the fonds Européen de Développement Régional FEDER, the Wallonia within the Wallonie-2020.EU program, the Plan Marshall, the FRS-FNRS of Belgium, the EU's H2020 project NEUROTECH (Grant No. 824103), and the H2020 MC SWITCHBOARD ETN (Grant No. 674901). The authors declare that this study received funding from Toshiba Corporation. The funder was not involved in this study design, collection, analysis, interpretation of data, the writing of this article, or the decision to submit it for publication.

ACKNOWLEDGMENTS

The authors would like to acknowledge the 2019 Capocaccia and Telluride Neuromorphic Workshops and all their participants for the fruitful discussions, and Intel Corporation for access to Loihi neuromorphic platform. We thank Prof. B. Miramond, Prof. D. Bol, Prof. S. Liu, Prof. T. Delbruck, and Prof. G. Indiveri. Finally, we thank Garrick Orchard for supporting us with the use of the Loihi platform and the useful comments to the paper.

REFERENCES

Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Nolfo, C. D., et al. (2017). "A low power, fully event-based gesture recognition system," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Honolulu, HI), 7388–7397. doi: 10.1109/CVPR.2017.781

Anumula, J., Neil, D., Delbruck, T., and Liu, S.-C. (2018). Feature representations for neuromorphic audio spike streams. *Front. Neurosci.* 12:23. doi: 10.3389/fnins.2018.00023

Atzori, M., Gijsberts, A., Castellini, C., Caputo, B., Hager, A.-G. M., Elsig, S., et al. (2014). Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Sci. Data* 1:140053. doi: 10.1038/sdata.2014.53

Barker, J., Marxer, R., Vincent, E., and Watanabe, S. (2015). "The third 'chime' speech separation and recognition challenge: dataset, task and baselines," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)* (Scottsdale, AZ), 504–511. doi: 10.1109/ASRU.2015.7404837

Benatti, S., Casamassima, F., Milosevic, B., Farella, E., Schönle, P., Fateh, S., et al. (2015). A versatile embedded platform for emg acquisition

and gesture recognition. *IEEE Trans. Biomed. Circuits Syst*. 9, 620–630. doi: 10.1109/TBCAS.2015.2476555

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

Biagetti, G., Crippa, P., and Falaschetti, L. (2018). Classifier level fusion of accelerometer and semg signals for automatic fitness activity diarization. *Sensors* 18:2850. doi: 10.3390/s18092850

Brader, J. M., Senn, W., and Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Comput*. 19, 2881–2912. doi: 10.1162/neco.2007.19.11.2881

Braun, S., Neil, D., Anumula, J., Ceolini, E., and Liu, S. (2019). "Attention-driven multi-sensor selection," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest), 1–8. doi: 10.1109/IJCNN.2019.8852396

Castanedo, F. (2013). A review of data fusion techniques. *TheScientificWorldJournal* 2013:704504. doi: 10.1155/2013/704504

Ceolini, E., Taverni, G., Khacef, L., Payvand, M., and Donati, E. (2019a). "Live demostration: sensor fusion using emg and vision for hand gesture classification in mobile applications," in *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Nara), 1. doi: 10.1109/BIOCAS.2019.8919163

Ceolini, E., Taverni, G., Khacef, L., Payvand, M., and Donati, E. (2019b). Sensor fusion using EMG and vision for hand gesture classification in mobile applications. *arXiv* 1910.11126. doi: 10.1109/BIOCAS.2019.8919210

Chan, V., Liu, S.-C., and van Schaik, A. (2007). Aer ear: A matched silicon cochlea pair with address event representation interface. *IEEE Trans. Circuits Syst. I Reg. Pap*. 54, 48–59. doi: 10.1109/TCSI.2006.887979

Chen, C., Yu, Y., Ma, S., Sheng, X., Lin, C., Farina, D., et al. (2020). Hand gesture recognition based on motor unit spike trains decoded from high-density electromyography. *Biomed. Signal Process. Control* 55:101637. doi: 10.1016/j.bspc.2019.101637

Cheok, M. J., Omar, Z., and Jaward, M. H. (2019). A review of hand gesture and sign language recognition techniques. *Int. J. Mach. Learn. Cybern*. 10, 131–153. doi: 10.1007/s13042-017-0705-5

Cholet, S., Paugam-Moisy, H., and Regis, S. (2019). "Bidirectional associative memory for multimodal fusion: a depression evaluation case study," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest), 1–6. doi: 10.1109/IJCNN.2019.8852089

Cicirelli, G., Attolico, C., Guaragnella, C., and D'Orazio, T. (2015). A kinect-based gesture recognition approach for a natural human robot interface. *Int. J. Adv. Robot. Syst*. 12:22. doi: 10.5772/59974

Corradi, F., and Indiveri, G. (2015). A neuromorphic event-based neural recording system for smart brain-machine-interfaces. *IEEE Trans. Biomed. Circuits Syst*. 9, 699–709. doi: 10.1109/TBCAS.2015.2479256

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). "Binaryconnect: training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, eds C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Montreal, QC: Curran Associates, Inc.), 3123–3131.

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

Debes, C., Merentitis, A., Heremans, R., Hahn, J., Frangiadakis, N., van Kasteren, T., et al. (2014). Hyperspectral and LiDAR data fusion: outcome of the 2013 grss data fusion contest. *IEEE J. Select. Top. Appl. Earth Observ. Rem. Sens*. 7, 2405–2418. doi: 10.1109/JSTARS.2014.2305441

Deiss, S. R., Douglas, R. J., and Whatley, A. M. (1999). "A pulse-coded communications infrastructure for neuromorphic systems," in *Pulsed Neural Networks*, eds W. Maass and C. M. Bishop (Cambridge, MA: MIT Press), 157–178.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). "Imagenet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL: IEEE), 248–255. doi: 10.1109/CVPR.2009.5206848

Donati, E., Payvand, M., Risi, N., Krause, R., Burelo, K., Indiveri, G., et al. (2018). Processing EMG signals using reservoir computing on an event-based neuromorphic system. in 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), pages 1–4. IEEE. doi: 10.1109/BIOCAS.2018.8584674

Donati, E., Payvand, M., Risi, N., Krause, R. B., and Indiveri, G. (2019). Discrimination of EMG signals using a neuromorphic implementation of a spiking neural network. *IEEE Trans. Biomed. Circuits Syst*. 13, 795–803. doi: 10.1109/TBCAS.2019.2925454

Došen, S., Cipriani, C., Kostić, M., Controzzi, M., Carrozza, M. C., and Popović, D. B. (2010). Cognitive vision system for control of dexterous prosthetic hands: experimental evaluation. *J. Neuroeng. Rehabil*. 7:42. doi: 10.1186/1743-0003-7-42

Droniou, A., Ivaldi, S., and Sigaud, O. (2015). Deep unsupervised network for multimodal perception, representation and classification. *Robot. Auton. Syst*. 71, 83–98. doi: 10.1016/j.robot.2014.11.005

Edelman, G. M. (1987). *Neural Darwinism: The Theory of Neuronal Group Selection*. New York, NY: Basic Books.

Frenkel, C., Lefebvre, M., Legat, J.-D., and Bol, D. (2019a). A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS. *IEEE Trans. Biomed. Circuits Syst*. 13, 145–158. doi: 10.1109/TBCAS.2018.2880425

Frenkel, C., Legat, J.-D., and Bol, D. (2019b). Morphic: a 65-nm 738k-synapse/mm^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning. *IEEE Trans. Biomed. Circuits Syst*. 13, 999–1010. doi: 10.1109/TBCAS.2019.2928793

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

Gerstner, W. (1995). Time structure of the activity in neural network models. *Phys. Rev. E* 51, 738–758. doi: 10.1103/PhysRevE.51.738

Ghazaei, G., Alameer, A., Degenaar, P., Morgan, G., and Nazarpour, K. (2017). Deep learning-based artificial vision for grasp classification in myoelectric hands. *J. Neural Eng*. 14:036025. doi: 10.1088/1741-2552/aa6802

Gigli, A., Gregori, V., Cognolato, M., Atzori, M., and Gijsberts, A. (2018). "Visual cues to improve myoelectric control of upper limb prostheses," in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)* (Enschede: IEEE), 783–788. doi: 10.1109/BIOROB.2018.8487923

Guo, H., Chen, L., Shen, Y., and Chen, G. (2014). "Activity recognition exploiting classifier level fusion of acceleration and physiological signals," in *UbiComp 2014–Adjunct Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Seattle, WA), 63–66. doi: 10.1145/2638728.2638777

Haria, A., Subramanian, A., Asokkumar, N., Poddar, S., and Nayak, J. S. (2017). Hand gesture recognition for human computer interaction. *Proc. Comput. Sci*. 115, 367–374. doi: 10.1016/j.procs.2017.09.092

Hoeks, C., Barentsz, J., Hambrock, T., Yakar, D., Somford, D., Heijmink, S., et al. (2011). Prostate cancer: multiparametric MR imaging for detection, localization, and staging. *Radiology* 261, 46–66. doi: 10.1148/radiol.11091822

Horwitz, B., and Poeppel, D. (2002). How can EEG/MEG and fMRI/PET data be combined? *Hum. Brain Mapp*. 17, 1–3. doi: 10.1002/hbm.10057

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). "Binarized neural networks," in *Advances in Neural Information Processing Systems*, eds D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Barcelona: Curran Associates, Inc.), 4107–4115.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: training neural networks with low precision weights and activations. *J. Mach. Learn. Res*. 18, 6869–6898. doi: 10.5555/3122009.3242044

Izhikevich, E. M. (2004). Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw*. 15, 1063–1070. doi: 10.1109/TNN.2004.832719

Jiang, N., Dosen, S., Muller, K.-R., and Farina, D. (2012). Myoelectric control of artificial limbs—is there a need to change focus? *IEEE Signal Process. Mag*. 29, 152–150. doi: 10.1109/MSP.2012.2203480

Lahat, D., Adali, T., and Jutten, C. (2015). Multimodal data fusion: an overview of methods, challenges, and prospects. *Proc. IEEE* 103, 1449–1477. doi: 10.1109/JPROC.2015.2460697

Lee, H.-Y., Hsu, C.-M., Huang, S.-C., Shih, Y.-W., and Luo, C.-H. (2005). Designing low power of sigma delta modulator for

biomedical application. *Biomed. Eng. Appl. Basis Commun*. 17, 181–185. doi: 10.4015/S1016237205000287

Lichtsteiner, P., Posch, C., and Delbruck, T. (2006). "A 128 × 128 120 dB 30 MW asynchronous vision sensor that responds to relative intensity change," in *2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers* (San Francisco, CA: IEEE), 2060–2069. doi: 10.1109/ISSCC.2006.1696265

Liu, H., and Wang, L. (2018). Gesture recognition for human-robot collaboration: a review. *Int. J. Ind. Ergon*. 68, 355–367. doi: 10.1016/j.ergon.2017.02.004

Liu, S.-C., Delbruck, T., Indiveri, G., Whatley, A., and Douglas, R. (2014). *Event-Based Neuromorphic Systems*. Hoboken, NJ: John Wiley & Sons.

Loss, J. F., Cantergi, D., Krumholz, F. M., La Torre, M., and Candotti, C. T. (2012). "Evaluating the electromyographical signal during symmetrical load lifting," in *Applications of EMG in Clinical and Sports Medicine*, ed C. Steele (Norderstedt: Books on Demand), 1.

Markovic, M., Dosen, S., Cipriani, C., Popovic, D., and Farina, D. (2014). Stereovision and augmented reality for closed-loop control of grasping in hand prostheses. *J. Neural Eng*. 11:046001. doi: 10.1088/1741-2560/11/4/046001

Markovic, M., Dosen, S., Popovic, D., Graimann, B., and Farina, D. (2015). Sensor fusion and computer vision for context-aware control of a multi degree-of-freedom prosthesis. *J. Neural Eng*. 12:066022. doi: 10.1088/1741-2560/12/6/066022

Meier, K. (2015). "A mixed-signal universal neuromorphic computing system," in *2015 IEEE International Electron Devices Meeting (IEDM)* (Washington, DC: IEEE), 4–6. doi: 10.1109/IEDM.2015.7409627

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

Moons, B., Goetschalckx, K., Van Berckelaer, N., and Verhelst, M. (2017). "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers* (Pacific Grove, CA: IEEE), 1921–1925. doi: 10.1109/ACSSC.2017.8335699

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

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

Neftci, E., Mostafa, H., and Zenke, F. (2019). Surrogate gradient learning in spiking neural networks. *arXiv* abs/1901.09948.

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

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., et al. (2017). "Automatic differentiation in PyTorch," in *NeurIPS Autodiff Workshop* (Long Beach, CA).

Peng, L., Chen, L., Wu, X., Guo, H., and Chen, G. (2016). Hierarchical complex activity representation and recognition using topic model and classifier level fusion. *IEEE Trans. Biomed. Eng*. 64, 1369–1379. doi: 10.1109/TBME.2016.2604856

Phinyomark, A., N., Khushaba, R., and Scheme, E. (2018). Feature extraction and selection for myoelectric control based on wearable EMG sensors. *Sensors* 18:1615. doi: 10.3390/s18051615

Pitti, A., Blanchard, A., Cardinaux, M., and Gaussier, P. (2012). "Gain-field modulation mechanism in multimodal networks for spatial perception," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)* (Osaka), 297–302. doi: 10.1109/HUMANOIDS.2012.6651535

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*. 9:141. doi: 10.3389/fnins.2015.00141

Rivet, B., Wang, W., Naqvi, S. M., and Chambers, J. A. (2014). Audiovisual speech source separation: an overview of key methodologies. *IEEE Signal Process. Mag*. 31, 125–134. doi: 10.1109/MSP.2013.2296173

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

Shivappa, S. T., Trivedi, M. M., and Rao, B. D. (2010). Audiovisual information fusion in human–computer interfaces and intelligent environments: a survey. *Proc. IEEE* 98, 1692–1715. doi: 10.1109/JPROC.2010.2057231

Shrestha, S. B., and Orchard, G. (2018). "SLAYER: spike layer error reassignment in time," in *Advances in Neural Information Processing Systems 31*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Montreal, QC: Curran Associates, Inc.), 1419–1428.

Smith, L., and Gasser, M. (2005). The development of embodied cognition: six lessons from babies. *Artif. Life* 11, 13–29. doi: 10.1162/1064546053278973

Smith, L. H., Hargrove, L. J., Lock, B. A., and Kuiken, T. A. (2011). Determining the optimal window length for pattern recognition-based myoelectric control: balancing the competing effects of classification error and controller delay. *IEEE Trans. Neural Syst. Rehabil. Eng*. 19, 186–192. doi: 10.1109/TNSRE.2010.2100828

Tan, A.-H., Subagdja, B., Wang, D., and Meng, L. (2019). Self-organizing neural networks for universal learning and multimodal memory encoding. *Neural Netw*. 120, 58–73. doi: 10.1016/j.neunet.2019.08.020

Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., et al. (2018). Large-scale neuromorphic spiking array processors: a quest to mimic the brain. *Front. Neurosci*. 12:891. doi: 10.3389/fnins.2018.00891

Turk, M. (2014). Multimodal interaction: a review. *Pattern Recogn. Lett*. 36, 189–195. doi: 10.1016/j.patrec.2013.07.003

Yasen, M., and Jusoh, S. (2019). A systematic review on hand gesture recognition techniques, challenges and applications. *PeerJ Comput. Sci*. 5:e218. doi: 10.7717/peerj-cs.218

Zahra, O., and Navarro-Alarcon, D. (2019). "A self-organizing network with varying density structure for characterizing sensorimotor transformations in robotic systems," in *Towards Autonomous Robotic Systems*, eds K. Althoefer, J. Konstantinova, and K. Zhang (Cham: Springer International Publishing), 167–178. doi: 10.1007/978-3-030-25332-5_15

Zenke, F., and Ganguli, S. (2018). SuperSpike: supervised learning in multilayer spiking neural networks. *Neural Comput*. 30, 1514–1541. doi: 10.1162/neco_a_01086

Zhang, Y., Wang, Z., and Du, J. (2019). "Deep fusion: an attention guided factorized bilinear pooling for audio-video emotion recognition," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest), 1–8. doi: 10.1109/IJCNN.2019.8851942

Zhao, D., and Zeng, Y. (2019). "Dynamic fusion of convolutional features based on spatial and temporal attention for visual tracking," in *2019 International Joint Conference on Neural Networks (IJCNN)* (Budapest), 1–8. doi: 10.1109/IJCNN.2019.8852301

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 Neuroscience

ORIGINAL RESEARCH
published: 12 August 2020
doi: 10.3389/fnins.2020.00772

Biologically Plausible Class Discrimination Based Recurrent Neural Network Training for Motor Pattern Generation

Parami Wijesinghe†, Chamika Liyanagedera*† and Kaushik Roy*

School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, United States

Biological brain stores massive amount of information. Inspired by features of the biological memory, we propose an algorithm to efficiently store different classes of spatio-temporal information in a Recurrent Neural Network (RNN). A given spatio-temporal input triggers a neuron firing pattern, known as an attractor, and it conveys information about the class to which the input belongs. These attractors are the basic elements of the memory in our RNN. Preparing a set of good attractors is the key to efficiently storing temporal information in an RNN. We achieve this by means of enhancing the "separation" and "approximation" properties associated with the attractors, during the RNN training. We furthermore elaborate how these attractors can trigger an action via the readout in the RNN, similar to the sensory motor action processing in the cerebellum cortex. We show how different voice commands by different speakers trigger hand drawn impressions of the spoken words, by means of our separation and approximation based learning. The method further recognizes the gender of the speaker. The method is evaluated on the TI-46 speech data corpus, and we have achieved 98.6% classification accuracy on the TI-46 digit corpus.

Keywords: echo state networks, separation property, approximation property, class discrimination, motor pattern generation

OPEN ACCESS

Edited by:
Emre O. Neftci,
University of California, Irvine,
United States

Reviewed by:
Shaista Hussain,
*Institute of High Performance Computing (A*STAR), Singapore*
Peng Li,
University of California, Santa Barbara, United States
Tariq Tashan,
Al-Mustansiriya University, Iraq

***Correspondence:**
Parami Wijesinghe
pwijesin@purdue.edu

†*These authors have contributed equally to this work*

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *09 December 2019*
Accepted: *30 June 2020*
Published: *12 August 2020*

Citation:
Wijesinghe P, Liyanagedera C and Roy K (2020) Biologically Plausible Class Discrimination Based Recurrent Neural Network Training for Motor Pattern Generation. Front. Neurosci. 14:772. doi: 10.3389/fnins.2020.00772

1. INTRODUCTION

The biological brain continues to be one of the most astounding enigmas of nature. Unearthing the brain's mysteries for inspiration is prevalent in recent artificial neuron modeling efforts. For instance, spiking neural networks have gained attention over the years due to their information representation with biological neurons (Davies et al., 2018; Wijesinghe et al., 2018). Owing to spike based inter-neuron communication, the brain has evolved to achieve its signal-processing capabilities, at a power consumption which is orders of magnitude smaller than the state-of-the-art super computers (Cruz-Albrecht et al., 2012). Similar to the spike based communication, the "memory" is another important aspect that makes the biological brain fascinating. Memory is the information stored inside the brain by tuning synaptic weights through supervised and unsupervised learning transpired over a duration of time (Reber, 2010). A human brain can typically store information worth $\sim$ 2.5 petabytes, which is equivalent to the amount of data telecast through a television over 300 years (Nabavi et al., 2014). Other recent studies have shown that it could potentially be even 10-folds higher than what it was estimated, due to the discovery of 26

distinguishable synaptic strengths (Bartol et al., 2015). In contrast to digital memories, the content inside the brain is not byte-addressable (Forsythe et al., 2014). Instead, the content operates within a dynamic dictionary that constantly shifts to make room for new meaning (Forsythe et al., 2014).

The memory of the biological brain is fundamentally associative. As hypothesized and based on experiments conducted on monkeys (Suzuki, 2007), the hippocampus is important for the early formation of the new associations in memory. A new piece of information can be absorbed well if it can be associated to an existing knowledge that is already anchored in the memory. For example, if one wants to learn a new word called *rubeus* in Latin, which means "red," he/she can potentially think about the "r" sound at the beginning of both the words. Here the word "red" is in the existing memory and sound "r" is the association to the new word. The person can now easily remember that *rubeus* means red. Finding an association to an existing content is not merely sufficient to properly remember new data. For instance, consider the same previous word *rubeus*. The person who just remembered the association of the "r" sound will only be able to answer the question "which color is *rubeus* in Latin?," but not "what is 'red' in Latin?." If one does not remember the actual word, the answer to question "which color is *ravus* in Latin?" would again be "red" since the person merely remembers some association with the sound "r." The answer is incorrect since *ravus* means gray. In a more complicated situation, assume one should remember the word *rot* which is red in German along with *rubeus*. Now the person should consider ways of distinguishing the two words despite the fact that they have the same meaning "red," in order to properly digest them simultaneously.

In this work, we consider the above phenomenon related to memory and construct an algorithm to help store significant amounts of data in a neural network. The brain is capable of remembering both static (example an image) and temporal (example a song) information. We will be focusing on the latter form of data learning for a recurrent neural network. One hypothesis for the way the brain stores temporal information is by means of attractors (Laje and Buonomano, 2013). This hypothesis is built upon the functionality of the cerebellum: a part of the biological brain that plays an important role in maintaining correct timing of motor actions. The role of cerebellum in sensory-motor actions is explained by means of experiments conducted on cerebellar patients (Jacobson et al., 2008). Such patients have increased temporal variability between motor actions, such as inaccurate timing of ball release when throwing a ball (Timmann et al., 2001) or variability shown during rhythmic tapping (Ivry et al., 1988). Cerebellum is also known for using associative learning to pair external stimuli with motor timing tasks (Paton and Buonomano, 2018). The classical eyeblink conditioning experiment shows how associative learning is used to program the cerebellum to react to a conditional stimulus such as a tone with an eyeblink reflex (Medina and Mauk, 2000; Johansson et al., 2016). This experiment is a perfect demonstration of the cerebellum's capacity for temporally specific learning. There are many standing theories as to how the cerebellum generates these temporal patterns and one such theory is the aforementioned attractor hypothesis (Laje and Buonomano, 2013). In this work we implement a biologically plausible reservoir computing (Wang and Li, 2016; Tanaka et al., 2019) network that uses this attractor hypothesis to emulate the temporal pattern generation capabilities of the cerebellum.

The temporal inputs that belong to a particular class trigger a certain internal neuron firing pattern. These patterns can be thought of as a representation of the existing knowledge in the memory corresponding to the temporal input. Let us call these anchored knowledge (or the internal dynamics of the network) as class attractors. The validity of the "attractor" hypothesis for large amounts of data and classes is yet to be analyzed. For instance, the work in Laje and Buonomano (2013) shows motor pattern generation application for voice commands but the number of inputs and classes are limited. As the number of different pattern classes increases, the corresponding class attractors are more likely to stay close to each other leading to more misclassifications. For example, one might mishear the word "bold" as "bald." Here we propose a mechanism to enhance the deviation between the attractor dynamics by extracting key differences between input pattern classes.

In order to recognize whether a projection of a particular input is a better representation (in the context of a classification task), certain properties must be considered. Two such properties are "separation" and "approximation." These are analogous to the phenomenon described previously on associativity in the biological memory. In a classification problem, projections of inputs corresponding to two different classes must stay apart from each other (separation). The projections that belong to the same class must stay close to each other (approximation). For instance, an utterance of the word "one" by male speakers should converge to one attractor (approximation). When this particular attractor is triggered, brain recognizes it as the word "one." If the same word spoken by females also triggers the same attractor, then the brain will not be able to recognize whether the speaker is female or male, despite the fact that it could recognize the spoken word. Therefore, in a scenario where the gender of the speaker must be identified, the attractor triggered by the male speakers and female speakers for the same word should be different (separation). Closer the attractors are, harder it would be to recognize the gender of the speaker. Our proposed learning approach (for a recurrent neural network) takes into account these properties, and improves class discrimination for better accuracy in a sensory motor task. i.e., we convert utterances of words (sensory data) into handwritten impressions (motor action) using reservoir computing. The network furthermore recognizes the gender of the speaker, and generates an impression of letter "f" (for female) or letter "m" (for male).

In the context of temporal information processing, one can find numerous studies investigating the speech recognition problem using strictly feed forward networks such as Convolutional Neural Networks (Swietojanski et al., 2014; Palaz et al., 2015), Deep Neural Networks (Hinton et al., 2012), Hidden Markov Models (Tran and Wagner, 1999), and Spiking Neural Networks (Liu et al., 2019; Zhang and Li, 2019). Recently, biologically inspired training methodologies (Neftci et al., 2016) and reservoir computing solutions such as Liquid State Machines

(Wang et al., 2015; Jin and Li, 2017) or Echo State Networks (ESN) (Skowronski and Harris, 2007; Laje and Buonomano, 2013) are been investigated extensively as an effort to bridge the gap between biological plausibility and practicality. Similarly, the work presented here are more geared toward replicating the activity of the cerebellum in generating complex motor patterns. We employ an ESN configured as a bio plausible practical implementation on how the cerebellum performs complex motor timing tasks. Like the cerebellum, the proposed network reacts to a temporal input and generates a timed motor response using a single reservoir network. A strictly feed forward network would be sufficient for the task if the objective was to simply classify the audio inputs into classes. However, in order to generate pre-determined timed responses such as motor tasks, a feed forward network would require additional timers and memory elements to store the sequences of movements to be performed.

An ESN is a simple form of a recurrent neural network with a pool of randomly interlinked leaky integrate analog neurons called the reservoir (Jaeger, 2007). The time varying inputs are connected to the reservoir by means of synapses of random weights. The reservoir neuron dynamics are directed toward a set of output neurons by means of a readout. These readout connections are trained using supervised methods. Some architectures use feedback connections from the output neurons to the reservoir neurons. However, in this work we do not use such feedback connections. In addition to training the readout connections, we also tune the input-reservoir connections and the recurrent connections within the reservoir itself.

The training mechanism consists of three major steps. 1. Separation based input-reservoir connection training, 2. Approximation based innate dynamic training of the reservoir connections, and 3. Readout connection training for motor pattern generation. During the first step, we obtain a set of well-separated innate dynamics per class (class attractors). Then in the second step, we converge all the reservoir dynamics of inputs in a given class, to its corresponding class attractor. Finally we convert the reservoir dynamics to a set of time varying coordinates to generate an impression of the spoken word, by means of the readout layer. We employ the entire TI46-digit and alphabet corpuses for our experiments. Following are the key contributions of this work.

1. Explaining the need of a set of well-separated attractors when dealing with bigger data sets.
2. Proposing a training algorithm to initially separate the attractors, and then make the reservoir dynamics for input instances, converge to their corresponding class attractor (discrimination based training).
3. Using two full data sets, validate how the accuracy improved with the separation based training.
4. Show the ability to generate motor patterns based on other attributes of the inputs. Apart from drawing the spoken character, the trained ESN can now recognize the gender of the speaker and generate a motor pattern corresponding to that simultaneously.
5. Use the network on an image based application to show the generality of the discrimination based training method.

2. MATERIALS AND METHODS

2.1. Echo State Networks—The Network Structure

In this section, the structure of the recurrent neural network involved in this work will be explained. For spatio-temporal data processing, we used an echo state network, a simple form of a recurrent neural network architecture (when compared with Long Short Term Memory networks or LSTMs; Hochreiter and Schmidhuber, 1997). An ESN (Jaeger, 2007) consists of a pool of neurons recurrently interlinked, called the reservoir, and a readout layer. Inputs are applied on the reservoir neurons by means of input-to-reservoir connections. Owing to the recurrent connections within the reservoir, a temporally varying input signal applied on the network at time $t = 0$, could potentially leave the neurons firing (an "echo" of the input) even after the input has been detached (hence the name echo state network). Such "echoes" or residuals of the inputs can be measured through the output layer in order to perform a particular task. The output connections are typically trained using supervised methods such as delta rule, backpropagation (Rumelhart et al., 1988) and recursive least square algorithm (RLS) (Haykin, 1991). Some architectures (Tanaka et al., 2019) also have a set of feedback connections from the output to the reservoir (**Figure 1**). There have been multiple opinions on whether the brain acts as a feedback system, and according to studies (Byrne and Dafny, 1997), the brain is mostly a feedforward system. Feedforward systems are fast and require certain knowledge about the outcome that correspond to a given input (similar to a lookup table). On the other hand, systems with feedbacks continuously monitor the output in order to modify the internal dynamics to achieve a certain target output. Such systems are sluggish than feedforward systems. Therefore, with the goal to achieve faster training, we did not use the feedback connections in our structure.

2.1.1. Reservoir Neurons

The neurons within the reservoir are leaky integrate neurons (Jaeger, 2007). The dynamics of the neurons are analog in fashion and can be given by the following equation.

$$-\tau_{uni}\frac{dx(t)}{dt} = -\alpha_l x(t) + W_{res}r(t) + W_{in}u(t) \tag{1}$$

$$r(t) = tanh\Big(x(t)\Big)$$

where $x(t)$ is the state of the neuron, $r(t)$ is the firing rate of the neuron (output of the neurons, which is simply a non-linear function of the neuron's state), $W_{res} \in \mathbb{R}^{n_{res}\times n_{res}}$ is the connection matrix inside the reservoir, and $W_{in} \in \mathbb{R}^{n_i\times n_{res}}$ is the connection matrix from the inputs $[u(t)]$ to the reservoir. τ_{uni} is the uniform time constant, and α_l is the leak coefficient. The output of the network is taken from the readout as follows.

$$y(t) = W_{out}r(t) \tag{2}$$

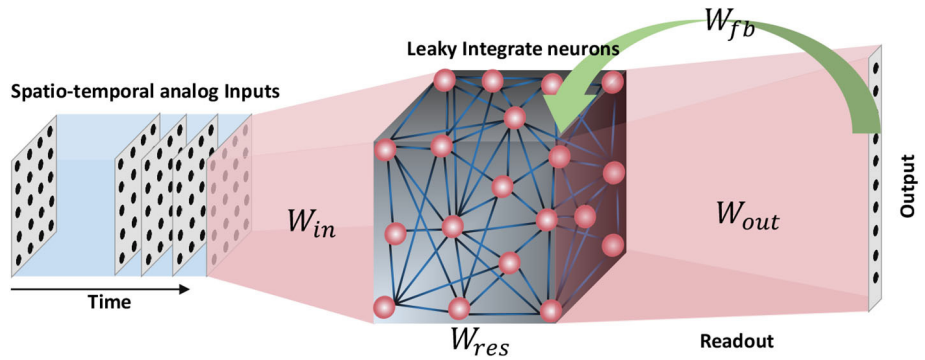


FIGURE 1 | The structure of an echo state network. A pool of randomly interlinked neurons known as the reservoir is the major component of an ESN. The neurons are analog leaky integrate neurons. Time varying inputs are connected to the reservoir and the reservoir neuron dynamics are decoded at the end of the readout. Some ESN architectures have feedback connections from the output to the input.

TABLE 1 | The hyper-parameters and their values used in this work.

Parameter name	Value
Sensory phase	300 ms
Motor phase	300 ms
Uniform time constant	0.04 ms
Leak constant	0.8
Time step	1 ms
Input channels	39
Output dimensions	2
Spectral radius scaling factor	1.4
Inverse learning rate (separation)	500
Inverse learning rate (approximation)	100

where $W_{out} \in \mathbb{R}^{n_{res} \times n_{out}}$ is the connection matrix from reservoir neurons to the outputs. The constant parameter values were selected as proposed in literature (Laje and Buonomano, 2013) and certain parameter were swept till the highest accuracy was achieved for a given number of neurons. The values are illustrated in **Table 1**. For solving the differential equations, we used standard Heun's method (Süli and Mayers, 2003) with a time step (dt) of 1 ms.

2.1.2. Network Connections

In a conventional echo state network, the reservoir and input-to-reservoir connections are randomly generated and only the final readout weights are trained. However, all the connections in the network in this work are trained using RLS learning rule. In a reservoir with randomly initialized weights (i.e., when no learning is involved to tune the connections), it is a good practice to have sparsity within the network in order to get better projections of the inputs. For example, multiple sparsely connected small reservoirs can give better class discrimination (hence better accuracy) for spatio-temporal data classification tasks using reservoir computing (Wijesinghe et al., 2019). This is due to the fact that different combinations from the same set of inputs were fed to the readout by means of an ensemble of reservoirs. However, in this work, since we are training all the network connections, we left the percentage connectivity equal to 100%. This gives more number of hyper parameters to change and finding the optimum set of weights is much faster using the RLS method (Sussillo and Abbott, 2009). Before training, the input-reservoir connections and within reservoir recurrent connections were randomly initialized using a normal Gaussian distribution. The reservoir connections were scaled by a factor in such a way that the spectral radius of the connection matrix is $r_s = 1.5$ (Laje and Buonomano, 2013).

2.2. Application

We perform a sensory motor application where the sensory input data are utterances of words, and the outputs are hand drawn impressions related to the spoken word and the speaker. For example, if the input voice command is an utterance of "six" by a female speaker, the output motor action would be to draw digit 6, and a letter "f." The inputs words are either utterances of digits or letters in the alphabet. In order to show the generality of our training method, we further included a third application that does not involve voice as an input command. In this application, the input is a hand drawn image, and the output is a time sequence that can be used to draw the corresponding digit. It further generates a letter "i" or "n" as another output at the same time, depending upon the face of the drawn digit ("i" for italic, "n" for normal character face). Refer to the **Supplementary Materials** for further details and results on this application.

2.2.1. Inputs

The first step is converting the input commands to a proper format to be processed by the network. For the input voice commands, the audio samples available in wave format were preprocessed based on Lyon's Passive Ear model (Lyon, 1982) of the human cochlea, using Slaney's MATLAB auditory toolbox (Slaney, 1998). The model was used to convert each audio sample to temporal variation in the amplitude of 39 frequency channels. The 39 signals were then down sampled ($\times 4$) in the temporal axis and applied as the input to the reservoir. The time during which the input data is applied on the network is the "sensory phase."

The two temporal (speech) data sets used in this work are:

1. Digit sub-vocabulary of the TI46 speech corpus (Liberman et al., 1993) (TI-10)
2. TI 26-word "alphabet set"; a sub-vocabulary in the TI46 speech corpus (Liberman et al., 1993) (TI-alpha).

TI-10 consists of utterances of the words "zero" through "nine" (10 classes) by 16 speakers. There are 1,594 instances in the training data set and 2,542 instances in the testing data set. TI-alpha, on the other hand, has utterances of the words "A" through "Z" (26 classes). There are 4,142 and 6,628 instances in the training and testing data sets, respectively.

2.2.2. Outputs

At the end of the sensory phase, the residual dynamics in the reservoir are converted to time varying signals at the output by means of a readout layer. The readout layer gives two sets of time varying x and y coordinates of hand drawn impressions (**Figure 2**). For the applications where the input is a set of voice commands, one such x and y coordinate set recognizes the gender of the speaker and draw either "f" (if female) or "m" (if male) accordingly. The other coordinates set generates the hand drawn impression of the uttered digit or letter. The duration within which these impressions are drawn is the "motor phase." The motor phase begins just after the sensory phase.

The network presented in this work is different from other networks that are used for traditional identification problems. The output of our network does not specifically say what class the input belongs to. The network responds to a spatio-temporal input with a spatio-temporal output based on prior knowledge, and the observer performs the classification task when they are reading the output. If an input that does not belong to any of the trained classes is presented to the network, the network can produce some temporal pattern that is not recognizable by any observer. Hence this is an open-set problem because the output can take infinitely different forms.

3. RESULTS

3.1. Training Methodology

The temporal inputs applied during the sensory phase trigger the neurons to fire in a certain way during the motor phase. The goal is to activate the same neuron firing pattern when inputs in a particular class are fed. i.e., there must be a specific firing pattern per class as shown in **Figure 3**. These reservoir neuron firing patterns are called the "class attractors." The key idea of the training methodology is to create a good set of class attractors by means of changing the input-reservoir and reservoir-reservoir connections, and changing the reservoir-readout connections to draw the corresponding impression. Following subsections will explain how the weights are systematically changed to craft these attractors. The entire training process has three major steps as explained below.

3.1.1. Step 1 : Separation Based Input-Reservoir Connection Training

The first step is creating a set of proper attractors which are triggered by input instances that belong to different classes. In order to assign an initial value to the class attractors, a set of inputs that represent each class (or class-template inputs) in the data set is required. We categorize the instances in the data set by both the spoken word and the gender of the speaker. For example, the TI-10 data set contains utterances of words by 8 male speakers and 8 female speakers, and each speaker utters the same word multiple times. Here the word "six" spoken by female speakers is considered as one class (notified as $Class_{6,f}$), and the word "six" spoken by male speakers is considered as another class (notified as $Class_{6,m}$). This class assignment is done since the readout layer recognizes both the spoken word and the gender of the speaker. Therefore, the total number of classes assigned for the TI-10 dataset is 20 (10 digits $\times$2 genders). Similarly, the total number of classes assigned for the TI-alpha dataset is 52 (26 alphabet letters $\times$ 2 genders).

A set of class-template inputs are created by taking the mean value of all the instances in each class. For example, assume there are f frequency channels, n_T number of time steps in the sensory phase, and n_f number of female speakers speaking the word "six" i times each. This gives $n_f \times i$ number of 2 dimensional ($f \times n_T$) examples in $Class_{6,f}$. The average 2 dimensional input among these $n_f \times i$ examples is evaluated and assigned as the input template of the particular class.

The generated class-template inputs are then applied on the reservoir to obtain the "innate dynamics" ("innate dynamics" are the firing rate dynamics of the neurons in the reservoir, for an applied input, under zero initial conditions and in the absence of noise) that can be considered as the initial assignment for class attractors. The work in Goudar and Buonomano (2018) uses these innate dynamics as the final class attractors

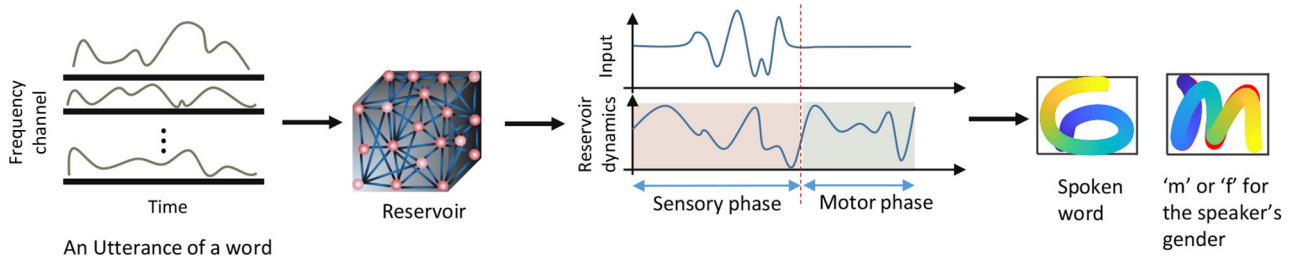


FIGURE 2 | The inputs and outputs of the recurrent network. The network dynamics are divided in to two phases. Sensory phase during which the input is applied, and the motor phase during which two motor patterns are generated.

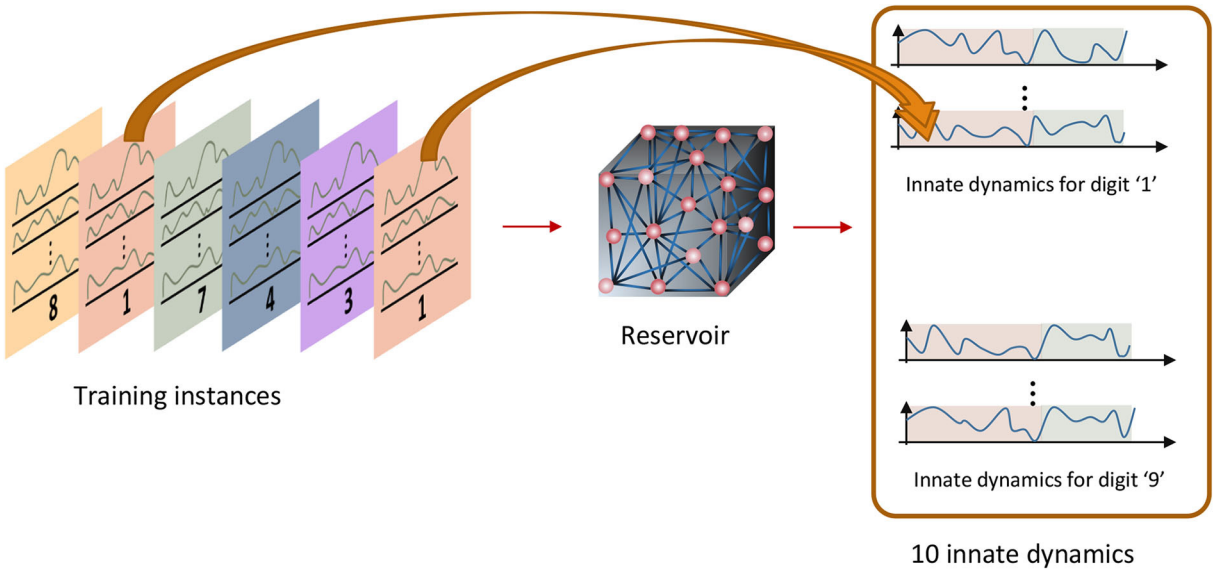


FIGURE 3 | Innate training: the instances that belong to one class should converge to a pre-assigned attractor.

(without any modifications), and the reason behind using the dynamics inherently generated by the reservoir as the attractors is not evident.

The difference among the innate dynamics indicate how separated the class attractors are. If two class attractors are close to each other, it is more likely that some input instances that belong to one class can trigger an attractor that corresponds to the other class, instead of converging to the correct class attractor. This will lead to erroneous classification, or improper motor pattern generation at the readout layer. When the number of classes and examples are higher, the possibility of two attractors staying close to each other in the multi-dimensional space is higher. Hence there is a need for separating the class attractors.

In order to separate the class attractors, a quantitative measure of separation is required. Multiple measures for separation (a measure of "kernel quality") in reservoirs are available in literature. Two such key ways of quantifying separation are known as *pairwiseseparationproperty* and *linearseparationproperty* (Maass et al., 2005; Legenstein and Maass, 2007; Wang et al., 2015). The pairwise separation property is defined as the distance between two continuous time states of a reservoir [$x_u(t)$ and $x_v(t)$], resultant from two separate inputs $u(t)$ and $v(t)$. The distance can be calculated by the Euclidean norm between $x_u(t_n)$ and $x_v(t_n)$ at sample point t_n. The average across all the sampled instances ($\forall t_n$) can be used to evaluate the final pairwise separation property, as explained in the following equation

$$SP_{PW} = \frac{1}{N_{samples}} \sum_{\mathrm{n}\,=\,1(0<t_n<T)}^{N_{samples}} ||x_u(t_n) - x_v(t_n)|| \qquad (3)$$

where $N_{samples}$ is the number of sample points. The pairwise separation property (SP_{PW}) can be used as a measure of the separation property for two given inputs. However, most real-life applications deal with more than two input spike trains. To address this, linear separation property is proposed as a more suitable quantitative measure to evaluate the reservoir computational power (Maass et al., 2005; Legenstein and Maass, 2007; Wang et al., 2015). The linear separation property (SP_{lin}) is the rank of the $N \times m$ matrix M_S, which contains the continuous time states $[x_{u_1}(t_0), ..., x_{u_m}(t_0)]$ of the reservoir as its columns. The continuous time state $x_{u_i}(t_0)$ is the reservoir response to the input u_i (from the training set), at time $t = t_0$. If the rank of the matrix is m, it guarantees that any given assignment of target outputs $y_i \in \mathbb{R}^{N_{out}}$ at time t_0 can be attained by means of a linear readout (Maass et al., 2005). The rank of M_S is the degree of freedom the linear readout has, when mapping x_{u_i} to y_i. Even though the rank is $< m$, it can still be used as a measure of reservoir quality(Maass et al., 2005).

$$\begin{aligned} M_S &= [x_{u_1}(t_o), ..., x_{u_i}(t_o), ..., x_{u_m}(t_o)] \\ SP_{lin} &= rank(M_S) \end{aligned} \qquad (4)$$

However, it is noteworthy that when the number of reservoir neurons is much larger than the number of inputs that is required to be separated ($N \gg m$), the rank of the matrix M_S is most likely equal to m ($SP_{lin} = m$). Furthermore, SP_{lin} is a discrete function and two reservoirs having the same SP_{lin} does not necessarily mean that their separation capability is identical. It is also noteworthy that the reservoir responses to m inputs can be further separated, even though the SP_{lin} has reached its highest possible value m.

In our work, it is required to increase the separation between the attractors. The number of attractors is equal to the number of classes, which is larger than two (SP_{PW} is not applicable) and much smaller than the reservoir neurons (SP_{lin} is not applicable). The

need for a quantitative measure of separation, that is a continuous function of the ESN weights arise. Therefore, we use insights from linear discriminant analysis (LDA) (Fisher, 1936; Fukunaga and Mantock, 1983; Hourdakis and Trahanias, 2013) to quantify the separation between the class attractors. The between class scatter matrix in the following equation contains information on how far each data point is located from the global mean, in the high dimensional space (Fukunaga and Mantock, 1983; Wijesinghe et al., 2019). Each data point is a vector that contains all the elements in an attractor matrix.

$$S_b = \sum_{i=1}^{L} P(\omega_i)(\mu_i - \mu_g)(\mu_i - \mu_g)^T \tag{5}$$

In the equation, μ_i is the sample mean vector (centroid) of class ω_i, $P(\omega_i)$ is the probability of class ω_i, L is the number of classes, and μ_g is the global sample mean vector. The single measure that quantifies the separation is given by the trace of the above matrix (Wijesinghe et al., 2019).

$$SP = trace(S_b) \tag{6}$$

Higher SP suggests better separation among the attractors. In the first step of the training process, we change the input-reservoir connections, such that the SP is increased. We use a modified version of the inverse of RLS for this purpose. The standard RLS learning rule, implemented according to the first-order reduced and controlled error (FORCE) algorithm can be used as follows to obtain a target dynamic in the reservoir (Sussillo and Abbott, 2009)

$$w(t) = w(t - \Delta t) - P(t)x_{in}(t)e(t) \tag{7}$$

$w(t - \Delta t) \in \mathbb{R}^{n_i \times n_{res}}$ is the input-reservoir connection matrix before the weight update, $x_{in}(t) \in \mathbb{R}^{n_i}$ is the input dynamics at time t. Each weight update is done in Δt time steps and it can be larger or equal to the simulation time step. The rule is similar to the delta rule but with multiple learning rates given by the matrix $P(= \left(x_{in}(t)x_{in}^T(t) + \alpha I\right)^{-1}, \alpha$ is a constant), which is a running estimate of the inverse of the correlation matrix of $x_{in}(t)$ (Sussillo and Abbott, 2009). $e(t) \in \mathbb{R}^{n_y}$ is the error between the target $f(t)$ and the actual reservoir dynamics at time t. n_i, n_{res}, and n_y are the number of input frequency channels, number of reservoir neurons and number of readout neurons, respectively.

$$e(t) = w^T(t - \Delta t)x_{in}(t) - f(t) \tag{8}$$

The learning rule makes the dynamics of the reservoir to reach a target function $f(t)$. However, the goal is to increase the distance between a set of attractors and to that effect, we modify the learning rule as follows. First we pick an input template of a particular class i, and apply it on the reservoir. The resultant reservoir dynamics of class i are then compared with previously evaluated attractors of class j ($j = 1, 2, ..., L; j \neq i$) to evaluate the difference $[e_{i,j}(t)]$ between them. Ideally we expect this difference to be large to obtain a set of well separated attractors. Considering this, the weight update rule can be modified as follows.

$$w(t) = w(t - \Delta t) + \gamma P(t)x_{in}(t)\left(\overrightarrow{1} \oslash e_{i,j}(t)\right) \tag{9}$$

where $\left(\overrightarrow{1} \oslash e_{i,j}(t)\right)$ gives the element-wise inverse of the error vector, and γ is a scaling factor. The input-reservoir weight update method extracts subtle differences in the input templates and exaggerates them, so that the differences are well-portrayed in the attractors. **Figures 4A,B** show how the attractors of $Class_{1,f}$ and $Class_{2,f}$ in the TI-10 dataset vary with time, before and after the weight update, respectively. Note that the separation between the two classes has visibly improved.

The main goal of the above elaborated first step (where we train the input-reservoir connections), is getting a set of well separated attractors to converge to. The attractors are directly dependent upon the applied inputs. If the input-reservoir weights were left randomly initialized and untrained, the contribution from the inputs will be random. Inputs from different classes can have different features. For instance, a female utterance of "*one*" can have more high frequency components than a male utterance of "*one*." We need to enhance the contribution from these distinguishing features in the input, to the reservoir. If this step is done collectively with the reservoir weights, we will have more hyper parameters to optimize. Therefore, during collective weight training, the changes in input-reservoir weights will be miniscule (in order to achieve the same separation between attractors given by only training the input-reservoir weights). For instance, when only input weights were used to train during the first step, 51% of the weight changes (Δw) were $> \pm 0.01$. In contrast, when the input-reservoir and reservoir-reservoir connections were trained collectively, only 0.25% of the input weight changes were $> \pm 0.01$. By training the input-reservoir weights separately, we get the best contribution from the input toward the reservoir and the optimum usage of input-reservoir weights which otherwise will be left almost untrained. Furthermore, collectively training the input - reservoir and reservoir-reservoir connections is computationally demanding. For instance, obtaining the attractors with collective weight training took 43× more time, than the input-reservoir weight training, for approximately the same separation amounts.

3.1.2. Step 2 : Approximation Based Reservoir Connection Training

After generating a well separated set of class attractors, the next step is converging all the instances in each class to their corresponding attractor. **Figure 5A** shows how the reservoir dynamics change for different instances in the same class. It is evident from the figure that all the instances in a class do not necessarily converge to the class attractor (shown in black dashed lines). To make them close to each other, here we train the reservoir-reservoir connections by means of the RLS rule implemented according to the FORCE algorithm (Sussillo and Abbott, 2009). The synaptic weight update is carried out as

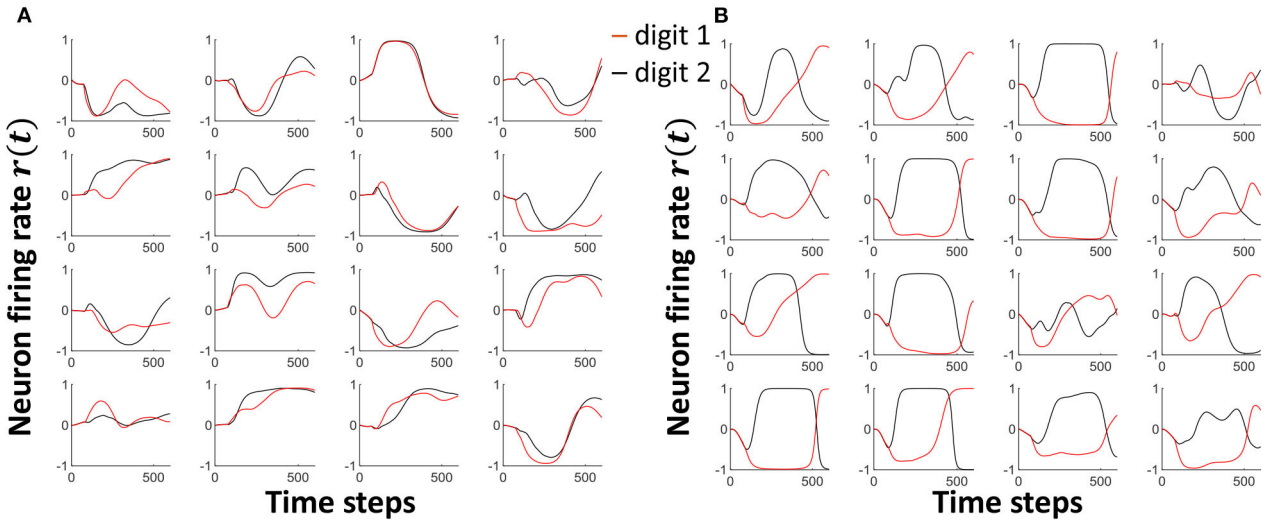


FIGURE 4 | The neuron attractor dynamics of 16 randomly picked neurons for two input voice utterance classes, viz. $Class_{1,f}$ and $Class_{2,f}$. **(A)** The dynamics before separation enhancement training **(B)** The dynamics after separation enhancement training. Note that the separation between the class attractors have significantly improved.

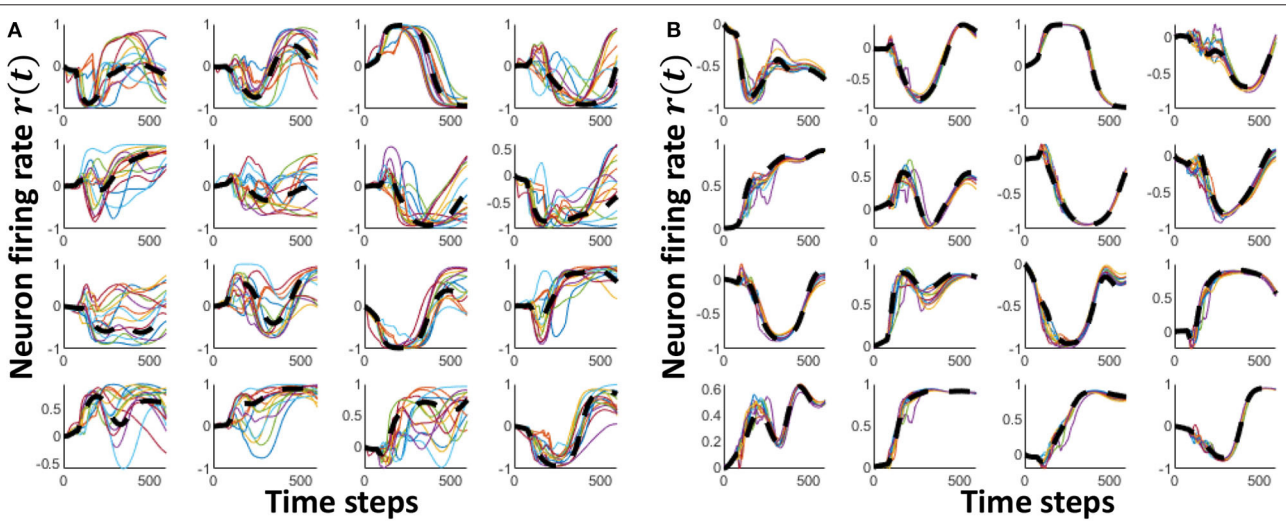


FIGURE 5 | The neuron firing rate dynamics of 16 randomly picked neurons for multiple input voice utterances of "one." **(A)** The dynamics before innate training **(B)** The dynamics after innate training. The black dashed line shows the innate dynamics of the same neurons for the class "one".

shown below.

$$w_{res}(t) = w_{res}(t - \Delta t) - P(t)r(t)e(t), \tag{10}$$

where $w_{res} \in \mathbb{R}^{n_res \times n_res}$ is the connection matrix within the reservoir, $r(t)$ gives the reservoir neuron firing rate at time t, $e(t)$ gives the error between the actual reservoir dynamics and the corresponding attractor dynamic. $P(t)$ is an $N \times N$ matrix updated along with the weights as follows

$$P(t) = P(t - \Delta t) - \frac{P(t - \Delta t)r(t)r^T(t)P(t - \Delta t)}{1 + r^T(t)P(t - \Delta t)r(t)} \tag{11}$$

The initial value of $P(t)$ is selected as $P(0) = I/\alpha$, where $1/\alpha$ is the learning rate. As shown in **Figure 5B**, the reservoir dynamics are close to the class attractor after the training step. In this second step, we have achieved proper approximation.

3.1.3. Step 3 : Readout Training for Motor Pattern Generation

In the previous two steps, we obtained a set of well separated attractors, and made all the instances of each class to converge to their corresponding class attractor. From these class attractors, now the readout layer generates two motor patterns that depict the gender of the speaker and the spoken word itself.

Closer the reservoir dynamics are to their class attractors, easier it is for the readout layer to clearly generate the motor pattern.

The target motor patterns are given in terms of temporally varying x and y coordinates. Hence there are two outputs (x and y) in the readout as shown in **Figure 6**. The figure shows how the x and y coordinates vary with time for a hand drawn impression of digit 6. All the spoken words (10 digits and 26 alphabet letters) were hand drawn in such a way that no lifting in the hand is required while drawing. The sequential coordinates of the images were extracted with the assistance of MATLAB *ginput*() function. The obtained $x-y$ coordinates were then resampled to generate equally distant points. Same RLS rule is used for training the readout weights w_{out} :

$$w_{out}(t) = w_{out}(t-\Delta t) - P(t)r(t)e(t) \tag{12}$$

where $r(t)$ is the reservoir dynamics and $e(t)$ gives the error between the expected coordinates and the actual coordinates at the output. Weight updates for the gender recognition is independently carried out from that of the spoken word recognition. **Figure 7** shows expected and actual (red) impressions drawn for the TI-10 motor pattern generation task. **Figure 8** shows expected and actual (red) impression drawn for the TI-alpha motor pattern generation task. The color of the motor pattern explains the time evolution of the coordinates at the readout. **Figure 9** shows the corresponding reservoir dynamics of 10 randomly selected neurons during the sensory and motor phases for the TI-10 data set. Refer to the **Supplementary Video** to view the RNN drawing digits.

3.2. Separation vs. Accuracy

We measure the error by means of the average squared difference between the actual and the expected output (at the readout) per input instance in the testing data set. The error d is given by

$$d = \frac{1}{N_{ex}} \sum_{i=1}^{N_{ex}} \sqrt{\sum_{j=1}^{N_{points}} \left(x_{i,j}^{t} - x_{i,j}^{a}\right)^2 + \left(y_{i,j}^{t} - y_{i,j}^{a}\right)^2} \tag{13}$$

where N_{ex} is the number of examples in the test data set, N_{points} is the number of sample points in the output motor pattern, $x_{i,j}^{t}$ is the target x coordinate and $x_{i,j}^{a}$ is the actual x coordinate at the output of the jth point in the ith example. Similarly, $y_{i,j}^{t}$ and $y_{i,j}^{a}$ are target and actual y coordinates at the output, respectively. In the TI-10 digit drawing task, we noticed an average error of 0.0151, on the entire test data set. This is a $\sim$ 37% reduction in average error with respect to a system without the class attractor separation step. We further evaluated the recognition accuracy of the ESN by means of an additional neural network.

The final output in our work is a motor pattern. To identify how well the digits were drawn, (i.e., can a human recognize the drawn character?), we used a Convolutional Neural Network (CNN). The CNN was pretrained to recognize hand drawn digits. The particular CNN used in the work has two convolutional layers followed by subsampling. Each convolutional kernel is of size 5×5 and there are 6 and 12 feature maps at the output of first and second convolutional layers respectively ($28 \times 28 - 6c5 - 2s - 12c5 - 2s - 10o$; Palm, 2012). The training set for the CNN consists of the MNIST training data set (T_{MNIST}^{Train}), and the 10 target hand drawn digits involved in this work (T_{TARGET}). Finally, the trained

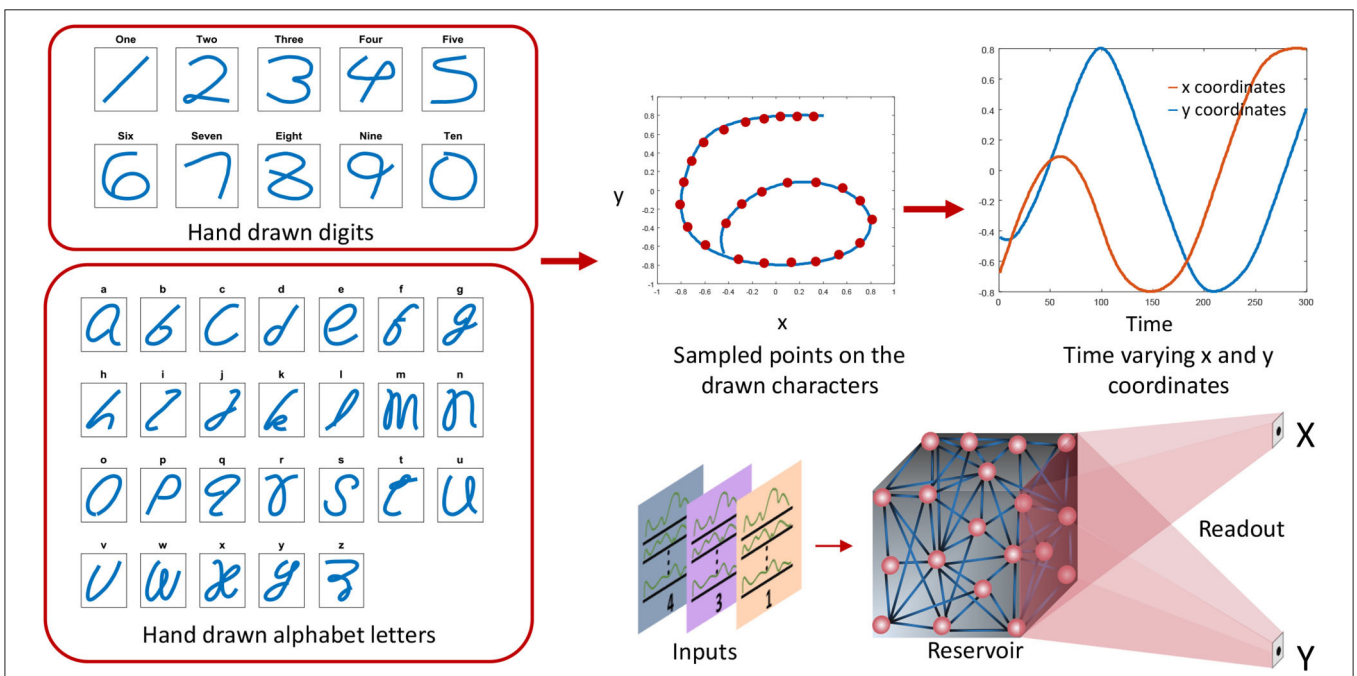


FIGURE 6 | The preprocessing steps for the outputs. The hand drawn digits and alphabet letters are converted to x, y coordinates and arranged sequentially. Two x and y coordinate signals are generated to draw the corresponding character.

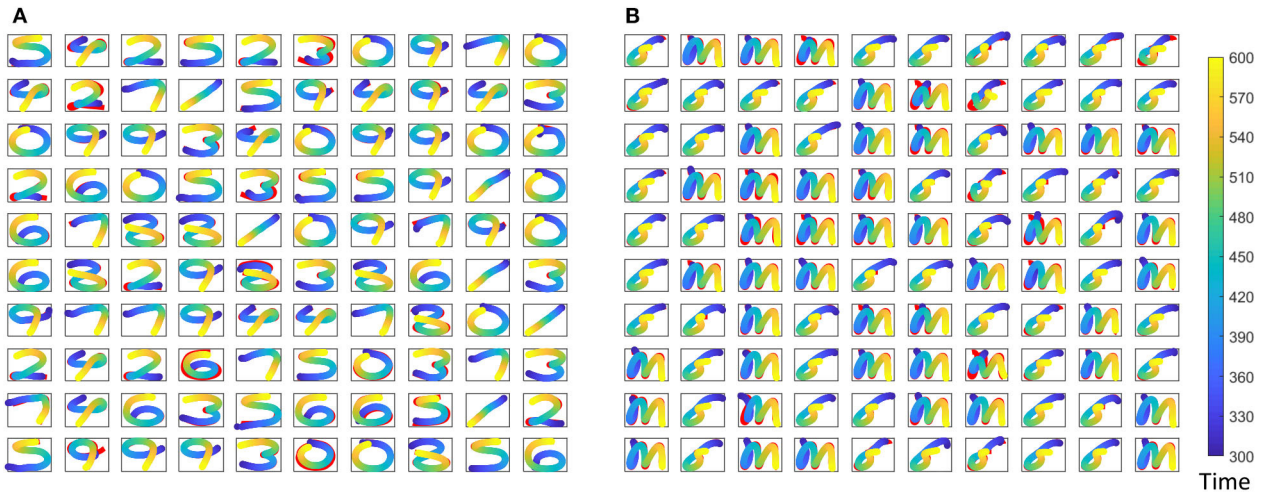


FIGURE 7 | Different motor patterns generated by the ESN for randomly picked 100 input utterances from the TI-10 test data set. **(A)** The spoken digit in the input instance. **(B)** The gender of the speaker with "f" for female and "m" for male. Color code shows the time evolution of the signal and shown in red is the expected motor pattern.

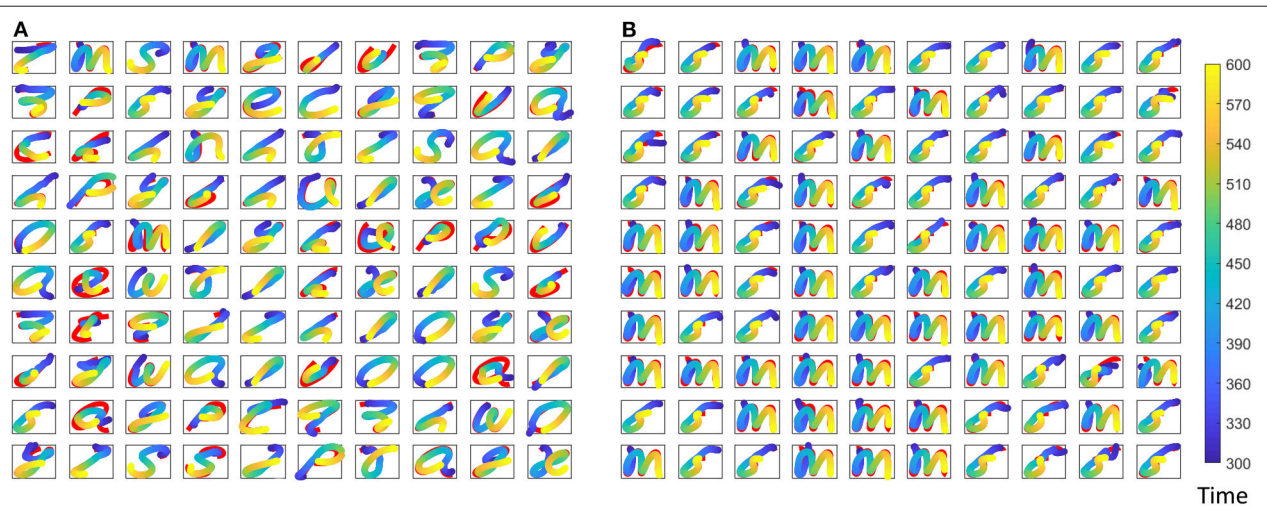


FIGURE 8 | Different motor patterns generated by the ESN for randomly picked 100 input utterances from the TI-alpha test data set. **(A)** The spoken alphabet letter in the input instance. **(B)** The gender of the speaker with "f" for female and "m" for male. Color code shows the time evolution of the signal and shown in red is the expected motor pattern.

CNN was fed with the output motor patterns generated by the ESN (T_{MOTOR}) to observe how "recognizable" they are by a CNN. For generating the T_{TARGET} and T_{MOTOR}, we converted the x, y coordinates of the temporal sequences into a 28×28 image to match the configuration of the instances in T^{Train}_{MNIST}. The CNN network was capable of classifying the 10 training images (T_{TARGET}) with 100% accuracy and MNIST testing data set with 98.9% accuracy. The accuracy on T_{MOTOR} was 98.6%. This accuracy is approximately similar to that reported in Goudar and Buonomano (2018) (also used a CNN for classification). However, it is noteworthy that there are few key differences in the setup involved in Goudar and Buonomano (2018). **Table 2** summarizes these changes along with the performances.

As tabulated in **Table 2**, the work proposed in Goudar and Buonomano (2018) uses a network with 4,000 neurons, and shows an accuracy of 98.7% on 410 examples across five speakers. Therefore, we have achieved similar accuracy to Goudar and Buonomano (2018) with a network of half the size as Goudar and Buonomano (2018), and on $\sim$ 4$\times$ larger number of examples (we used all the 1,594 instances from the TI-10 testing data set), owing to the discrimination based training approach.

To further validate the effect of class attractor separation on accuracy, we used the TI-alpha data set which has more number of classes and examples. We observed an error of 0.0596 in the spoken word generation task and an error of 0.0413 in the letter generation task related to the gender of the speaker. Without the

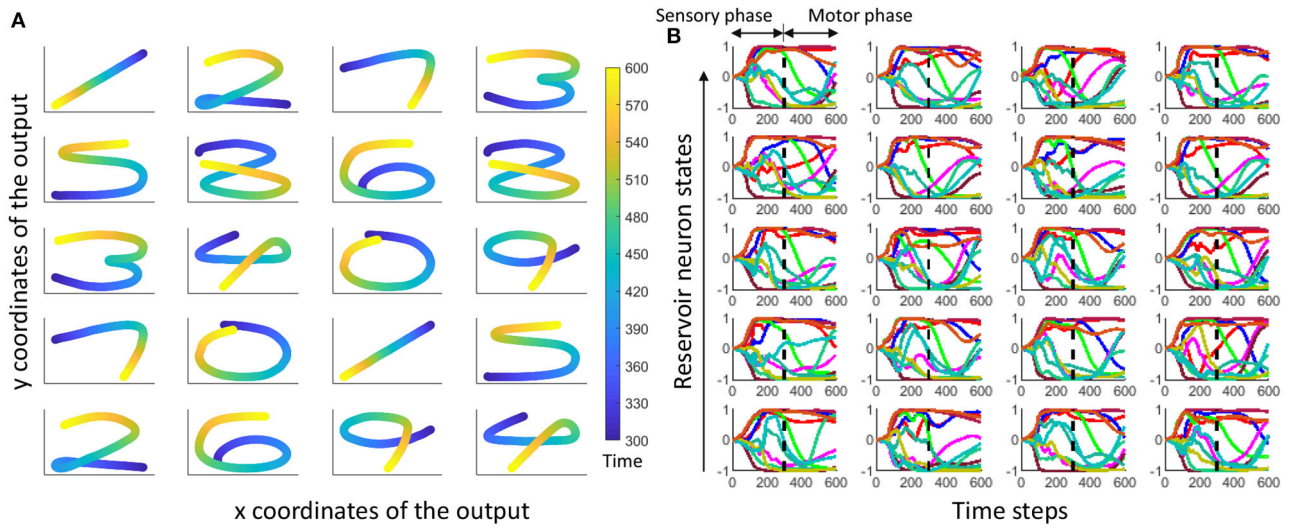


FIGURE 9 | (A) The hand drawn ESN outputs for 20 randomly selected input voice signals from TI-10 testing dataset. Color elaborates the time evolution of the drawn impression. **(B)** Time evolution of reservoir neuron states for randomly picked 10 neurons (out of 400) corresponding to each output in **(A)**.

TABLE 2 | The comparison with reference work.

Network type	Discrimination based training?	RNN type	Number of reservoir neurons	Number of novel testing examples	Number of speakers	Accuracy on 10 hand drawn digits (%)	Ability to classify speaker gender
Goudar and Buonomano (2018)	No	ESN	4,000	410	5	98.7	No
This work	Yes	ESN	2,000	1,594	16	98.6	Yes

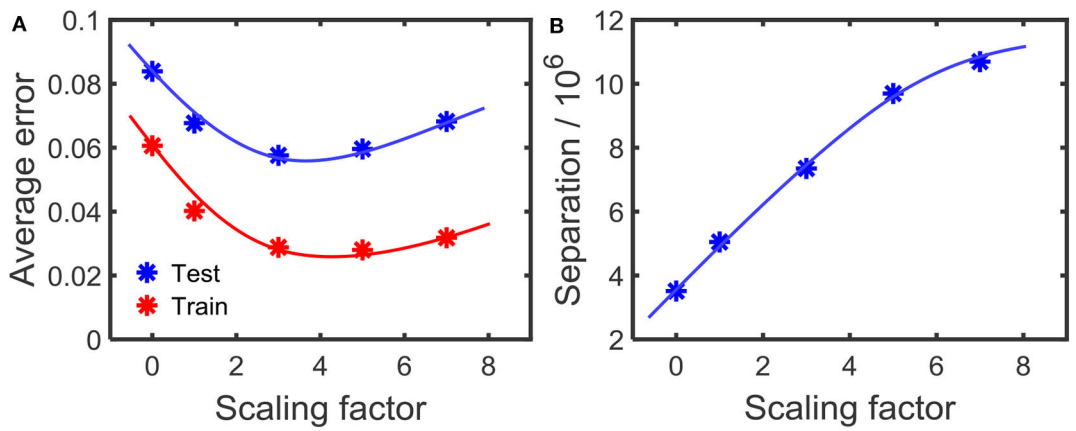


FIGURE 10 | (A) The variation of the error with the amount of separation applied on the attractors. The separation increases with the scaling factor as shown in **(B)**. The results are for the TI-alpha, spoken word recognition application.

separation step, we noticed an error increment of 29% in spoken word generation task.

In order to observe the effect on the error at different amounts of separations, we changed the scaling factor γ in the first step of the learning process (Equation 9). Higher γ will increase the separation between the attractors. **Figures 10A,B** shows how the error changes with the amount of separation. As the figure illustrates, high separation leads to lower error. However, if the separation is too high, then the error increases. We conjecture that this is due to the inability to converge instances in a class

to its corresponding attractor, due to the high separation. With high separation, the network enhances subtle changes in the input to an extent that the approximation step could no longer converge the inputs to their corresponding class attractor. This phenomenon is clearly explained in prior work (Wijesinghe et al., 2019), showing that the increased separation along with insufficient improvement in approximation property leads to reduced classification accuracy in reservoir computing.

Obtaining this optimum point (that gives the highest accuracy) before expensive training is beneficial. Multiple methods of identifying the optimum point based on separation and approximation for reservoir computing are available in literature (Wang et al., 2015; Wijesinghe et al., 2019). As proposed in Wang et al. (2015), the metric for optimum performance point can be obtained by the following equation

$$D = \frac{\sqrt{R_S - R_G}}{R_S} \tag{14}$$

Where R_S is a metric for separation and R_G is a metric for generalization. R_S is the rank of the $N \times m$ matrix M_S, which contains the continuous time states $[x_{u_1}(t_0), ..., x_{u_m}(t_0)]$ of the reservoir as its columns (explained in section 3.1.1). Same aforementioned rank concept is used for measuring R_G, but now on a different matrix M_a. M_a consists of reservoir states $x_{u_{ij}}(t_0)$ as its columns, which are measured by feeding jittered versions of u_i (u_{ij}) to the reservoir. Unlike R_S, lower rank of M_a ($= R_G$) suggests better generalization. The R_S metric for our work is the rank of the matrix that contains the attractors sampled at $t = t_0$, as its columns. Given that the attractors are equal to the number of classes, we need to find the rank of a 2000×20 matrix (for the voice based digit drawing application). As explained in section 3.1.1, the metric R_S is most likely equal to the number of classes (i.e., 20), since $N \gg m$. Therefore, R_S is simply a constant and may not contain any useful information. Hence the metric D may not be applicable for our work. However, the Discriminant ratio (DR) proposed in Wijesinghe et al. (2019) is much general and applicable in finding the optimum point. The metric can be elaborated in the following equations.

$$DR = tr(S_b)tr(S_w)^{-1} \tag{15}$$

$$S_w = \sum_{i=1}^{L} P(\omega_i)\hat{\Sigma}_i \tag{16}$$

where $P(\omega_i)$ is the probability of class ω_i, $\hat{\Sigma}_i$ is the sample covariance matrix (Park and Park, 2018) for class ω_i. $tr(S_b)$ is explained in Equations (5) and (6), as separation property (SP). As shown in Wijesinghe et al. (2019), the point at which this DR is a maximum, is the optimum accuracy point. As shown in **Figure 10A**, for the digit drawing problem, the highest accuracy point lies at scaling factor $\approx$ 3.7. Obtaining this point before the reservoir-output weight training can be done by using the aforementioned DR metric.

3.3. Convergence and Stability of the Network

In this section we are exploring the convergence of the training method and the stability of the trained system. The RLS learning method involved in this work was specifically proposed for recurrent neural networks with chaotic activity (such as the network used in this work). The other algorithms designed for RNNs are computationally demanding and do not converge under chaotic activity (Rumelhart et al., 1986; Abarbanel et al., 2008; Sussillo and Abbott, 2009). The factors that can potentially affect the convergence are the learning rate and the number of parameters available for optimization. With more parameters for optimization, we can attain more convergence (Wijesinghe et al., 2017). However, as explained in Bengio (2012), when the number of hyper parameters of a network is high, it becomes less general i.e., the network can predict the data in its training set with high accuracy, but it will likely fail to perform correctly for previously unseen inputs. Mechanisms have been proposed in literature to avoid such over-training situations including early stopping of training (Doan and Liong, 2004), adding stochastic noise (An, 1996) etc. These methods are still applicable to the training method proposed in this work as well.

As shown in Equation (9), we are using a particular scaling factor during training. This scaling factor could potentially be viewed as the learning rate of the system. In general, high learning rates may hinder the convergence to a required solution. In fact, the output can oscillate between high accuracy and low accuracy states between epochs (Attoh-Okine, 1999). Smaller learning rates on the other hand would take more number of epochs to achieve convergence, and possibly reach a local minimum point rather than the global minimum. In this work, we have shown the effect on the scaling factor to the accuracy in **Figure 10A**. We have used the same number of epochs for the experiment. Therefore, the magnitude of the learning rate decides how much separation we apply between the attractors. High scaling factors lead to higher amounts of oscillations in attractors, while trying to reach higher separation. **Figure 11** shows the attractors corresponding to digit 1 and digit 2 before (**Figure 11A**) and after (**Figure 11B**) training. The training was conducted with a high scaling factor (20). Note that the amount of oscillations has increased now to accommodate the separation between the attractors. Even though a good separation was achieved with the high separation rate, it is now difficult to converge different instances available in the same class to the same attractor (due to the highly detailed nature of the attractors).

We further analyzed the outcome of the trained system by means of principal components. Since the output is in high dimensional space, it is difficult to visualize the converged output. We concatenated temporal information of all the neurons in to a single vector per instance, and obtained the projection of them in to the two dimensional space with respect to the first two principal components. **Figure 12A** shows the data instances of two classes (digit 1 and digit 2) before training, and it is evident that the instances are not well separated and approximated. After training, the data instances are well separated and approximated as illustrated in **Figure 12B**. Note that even the male and female instances of each class are very well-separated.

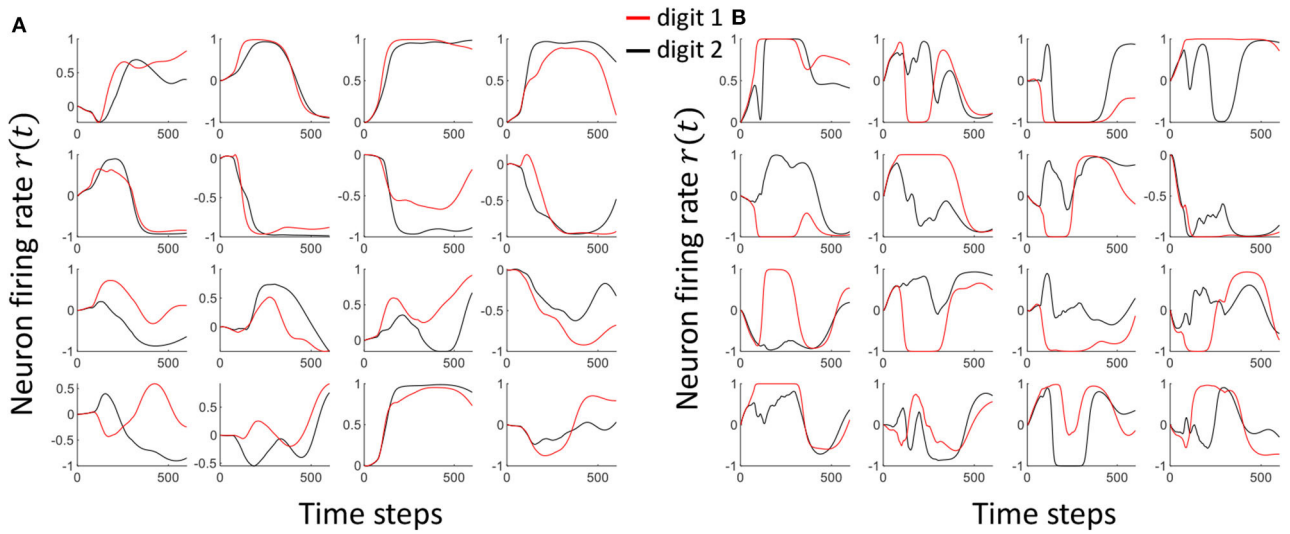


FIGURE 11 | The Attractor dynamics corresponding to digit 1 and digit 2, in 16 randomly picked neurons, **(A)** before training **(B)** after training with a high scaling factor. Notice the increased amount of oscillations leading to much detailed attractors after training with high separation.

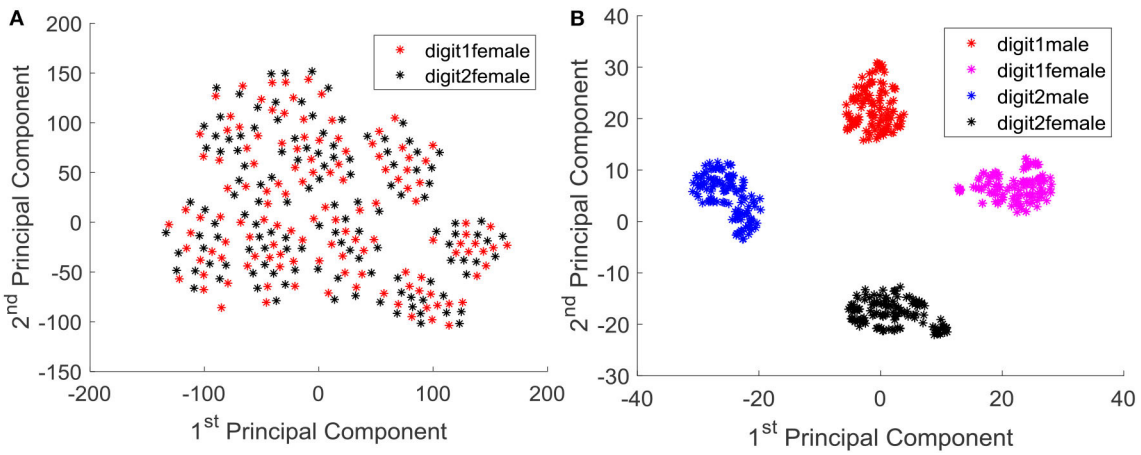


FIGURE 12 | (A) Instances of class digit 1 and digit 2 (female) projected in to the 2D principal component space, before training. **(B)** Instances of class digit 1(male, female) and digit 2 (male, female) projected in to the 2D principal component space, after training.

We further elaborate the effect of training by means of the Eigen-value spectrum of the reservoir weight matrix. As explained in Rajan and Abbott (2006), the eigen values of the weight matrix of a network provides insights on stability of the network. We noticed that the eigen values of the trained weights are more compressed on the right hand side, when compared with the uniformly distributed initial eigenvalue spectrum (**Figure 13**). This elaborates increased stability of the network after the training (Rajan and Abbott, 2006).

4. CONCLUSION

Biological brain; a mystery yet to be solved, is not just a system that can be perceived as a simple cognitive machine. It has the capability to go beyond perception based inference, and is capable of interacting with multiple tasks. Cognitive and motor functions are interlinked in the brain (Leisman et al., 2016). Taking that as an inspiration, this work concatenates multiple tasks into a single network, creating a system that goes beyond perception. The learning algorithm furthermore tries to mimic the properties of the brain that allows massive amount of information to be stored. To enable efficient learning (memorizing), biological brain creates new connections among the existing memory structures. For an incoming input, the brain not only can observe how close it is to an existing memory anchor, but can also detect how different it is from another anchor. The learning rule explained in this work emulates such mechanisms of the brain to store information efficiently utilizing association (approximation) and dissociation (separation) between the data.

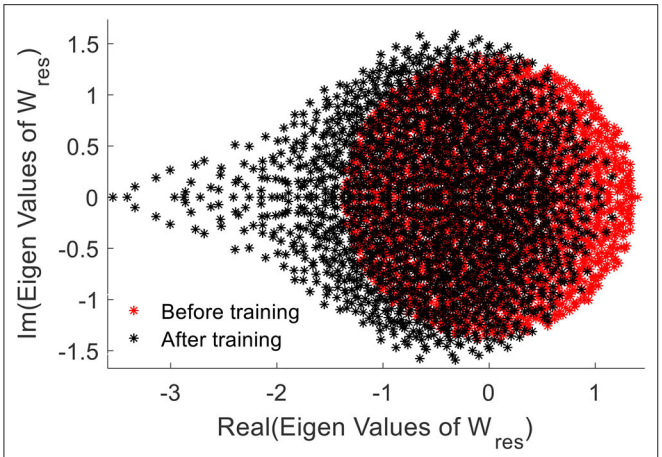


FIGURE 13 | The Eigen value spectrum of the reservoir weight before (red) and after training the network.

Our technique allowed to store twice the number of classes, with a reservoir of half the size (number of neurons), to achieve the same accuracy reported in Goudar and Buonomano (2018), on the entire TI-10 data set [which has $\sim$ 4 times the amount of data than Goudar and Buonomano (2018)]. We further verified the accurate performance on an even bigger data set (TI-alpha) with 52 classes and 6,628 training examples (in contrast to 20 classes and 2,542 training examples).

Biological brain does not store everything in one learning process. Over time it learns new meaning, forgets unwanted information, and gets reshaped by experience. In contrast, our proposed algorithm assumes that all the data are available at the time of training. i.e., it does not learn one instance completely and move to the rest of the data. However, the algorithm can potentially be extended to learn things over time. It will be analogous to increasing the number of attractors over time, rather than starting with a predefined number of attractors. It is as if a baby learns mothers voice first (which is an attractor), and then over time the baby learns different speakers (more class attractors). The class attractors must be adjusted over time using separation, in order to make room for new data and create the dynamic dictionaries the biological brains have.

DATA AVAILABILITY STATEMENT

The datasets generated for this study are available on request to the corresponding author.

AUTHOR CONTRIBUTIONS

PW and CL performed the simulations. All the authors contributed in developing the concepts, generating experiments, and writing the manuscript.

FUNDING

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, by the Semiconductor Research Corporation, the National Science Foundation, Intel Corporation, the DoD Vannevar Bush Fellowship, and by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00772/full#supplementary-material

REFERENCES

Abarbanel, H. D., Creveling, D. R., and Jeanne, J. M. (2008). Estimation of parameters in nonlinear systems using balanced synchronization. *Phys. Rev. E* 77:016208. doi: 10.1103/PhysRevE.77.016208

An, G. (1996). The effects of adding noise during backpropagation training on a generalization performance. *Neural Comput.* 8, 643–674. doi: 10.1162/neco.1996.8.3.643

Attoh-Okine, N. O. (1999). Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance. *Adv. Eng. Softw.* 30, 291–302. doi: 10.1016/S0965-9978(98)00071-4

Bartol, T. M. Jr, Bromer, C., Kinney, J., Chirillo, M. A., Bourne, J. N., Harris, K. M., et al. (2015). Nanoconnectomic upper bound on the variability of synaptic plasticity. *Elife* 4:e10778. doi: 10.7554/eLife.10778

Bengio, Y. (2012). "Practical recommendations for gradient- based training of deep architectures," in *Neural Networks: Tricks of the Trade*, eds G. Montavon, G. B. Orr, and K.-R. Müller (Berlin; Heidelberg: Springer), 437–478. doi: 10.1007/978-3-642-35289-8_26

Byrne, J., and Dafny, N. (1997). *Neuroscience Online: An Electronic Textbook for the Neurosciences*. Department of Neurobiology and Anatomy; The University of Texas Medical School at Houston.

Cruz-Albrecht, J. M., Yung, M. W., and Srinivasa, N. (2012). Energy-efficient neuron, synapse and STDP integrated circuits. *IEEE Trans. Biomed. Circuits Syst.* 6, 246–256. doi: 10.1109/TBCAS.2011.2174152

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

Doan, C. D., and Liong, S. (2004). "Generalization for multilayer neural network Bayesian regularization or early stopping," in *Proceedings of Asia Pacific Association of Hydrology and Water Resources 2nd Conference* (Singapore), 5–8.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Ann. Eugen.* 7, 179–188. doi: 10.1111/j.1469-1809.1936.tb02137.x

Forsythe, C., Liao, H., Trumbo, M. C. S., and Cardona-Rivera, R. E. (2014). *Cognitive Neuroscience of Human Systems: Work and Everyday Life*. Boca Raton, FL: CRC Press. doi: 10.1201/b17445

Fukunaga, K., and Mantock, J. M. (1983). Nonparametric discriminant analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* PAMI-5, 671–678. doi: 10.1109/TPAMI.1983.4767461

Goudar, V., and Buonomano, D. V. (2018). Encoding sensory and motor patterns as time-invariant trajectories in recurrent neural networks. *Elife* 7:e31134. doi: 10.7554/eLife.31134

Haykin, S. (1991). *Adaptive Filter Theory*. Englewood Cliffs, NJ: Prentice-Hall.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-R., Jaitly, N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. *IEEE Signal Process. Mag.* 29, 82–97. doi: 10.1109/MSP.2012.2205597

Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.* 9, 1735–1780. doi: 10.1162/neco.1997.9.8.1735

Hourdakis, E., and Trahanias, P. (2013). Use of the separation property to derive liquid state machines with enhanced classification performance. *Neurocomputing* 107, 40–48. doi: 10.1016/j.neucom.2012.07.032

Ivry, R. B., Keele, S., and Diener, H. (1988). Dissociation of the lateral and medial cerebellum in movement timing and movement execution. *Exp. Brain Res.* 73, 167–180. doi: 10.1007/BF00279670

Jacobson, G. A., Rokni, D., and Yarom, Y. (2008). A model of the olivo-cerebellar system as a temporal pattern generator. *Trends Neurosci.* 31, 617–625. doi: 10.1016/j.tins.2008.09.005

Jaeger, H. (2007). Echo state network. *Scholarpedia* 2:2330. doi: 10.4249/scholarpedia.2330

Jin, Y., and Li, P. (2017). Performance and robustness of bio-inspired digital liquid state machines: a case study of speech recognition. *Neurocomputing* 226, 145–160. doi: 10.1016/j.neucom.2016.11.045

Johansson, F., Hesslow, G., and Medina, J. F. (2016). Mechanisms for motor timing in the cerebellar cortex. *Curr. Opin. Behav. Sci.* 8, 53–59. doi: 10.1016/j.cobeha.2016.01.013

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

Legenstein, R., and Maass, W. (2007). Edge of chaos and prediction of computational performance for neural circuit models. *Neural Netw.* 20, 323–334. doi: 10.1016/j.neunet.2007.04.017

Leisman, G., Moustafa, A. A., and Shafir, T. (2016). Thinking, walking, talking: integratory motor and cognitive brain function. *Front. Public Health* 4:94. doi: 10.3389/fpubh.2016.00094

Liberman, M., Amsler, R., Church, K., Fox, E., Hafner, C., Klavans, J., et al. (1993). "Ti 46-word," in *Linguistic Data Consortium* (Philadelphia, PA: Texas Instruments, Inc.).

Liu, Y., Yenamachintala, S. S., and Li, P. (2019). Energy-efficient fpga spiking neural accelerators with supervised and unsupervised spike-timing-dependent-plasticity. *ACM J. Emerg. Technol. Comput. Syst.* 15, 1–19. doi: 10.1145/3313866

Lyon, R. (1982). "A computational model of filtering, detection, and compression in the cochlea," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82, Vol. 7* (Paris), 1282–1285. doi: 10.1109/ICASSP.1982.1171644

Maass, W., Legenstein, R. A., and Bertschinger, N. (2005). "Methods for estimating the computational power and generalization capability of neural microcircuits," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 865–872.

Medina, J. F., and Mauk, M. D. (2000). Computer simulation of cerebellar information processing. *Nat. Neurosci.* 3, 1205–1211. doi: 10.1038/81486

Nabavi, S., Fox, R., Proulx, C. D., Lin, J. Y., Tsien, R. Y., and Malinow, R. (2014). Engineering a memory with LTD and LTP. *Nature* 511:348. doi: 10.1038/nature13294

Neftci, E. O., Pedroni, B. U., Joshi, S., Al-Shedivat, M., and Cauwenberghs, G. (2016). Stochastic synapses enable efficient brain-inspired learning machines. *Front. Neurosci.* 10:241. doi: 10.3389/fnins.2016.00241

Palaz, D., Collobert, R., et al. (2015). *Analysis of CNN-Based Speech Recognition System Using Raw Speech as Input*. Technical report, Idiap.

Palm, R. B. (2012). *Prediction as a Candidate for Learning Deep Hierarchical Models of Data*. Technical University of Denmark.

Park, K. I., and Park (2018). *Fundamentals of Probability and Stochastic Processes With Applications to Communications*. Cham: Springer. doi: 10.1007/978-3-319-68075-0

Paton, J. J., and Buonomano, D. V. (2018). The neural basis of timing: distributed mechanisms for diverse functions. *Neuron* 98, 687–705. doi: 10.1016/j.neuron.2018.03.045

Rajan, K., and Abbott, L. F. (2006). Eigenvalue spectra of random matrices for neural networks. *Phys. Rev. Lett.* 97:188104. doi: 10.1103/PhysRevLett.97.188104

Reber, P. (2010). Ask the brains. *Sci. Am. Mind* 20:70. doi: 10.1038/scientificamericanmind0409-70

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323, 533–536. doi: 10.1038/323533a0

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cogn. Model.* 5:1.

Skowronski, M. D., and Harris, J. G. (2007). Automatic speech recognition using a predictive echo state network classifier. *Neural Netw.* 20, 414–423. doi: 10.1016/j.neunet.2007.04.006

Slaney, M. (1998). *Auditory Toolbox*. Technical Report, Interval Research Corporation.

Süli, E., and Mayers, D. F. (2003). *An Introduction to Numerical Analysis*. Cambridge, UK: Cambridge University Press. doi: 10.1017/CBO9780511801181

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

Suzuki, W. A. (2007). Making new memories: the role of the hippocampus in new associative learning. *Ann. N. Y. Acad. Sci.* 1097, 1–11. doi: 10.1196/annals.1379.007

Swietojanski, P., Ghoshal, A., and Renals, S. (2014). Convolutional neural networks for distant speech recognition. *IEEE Signal Process. Lett.* 21, 1120–1124. doi: 10.1109/LSP.2014.2325781

Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., et al. (2019). Recent advances in physical reservoir computing: a review. *Neural Netw.* 115, 100–123. doi: 10.1016/j.neunet.2019.03.005

Timmann, D., Citron, R., Watts, S., and Hore, J. (2001). Increased variability in finger position occurs throughout overarm throws made by cerebellar and unskilled subjects. *J. Neurophysiol.* 86, 2690–2702. doi: 10.1152/jn.2001.86.6.2690

Tran, D., and Wagner, M. (1999). "Fuzzy hidden Markov models for speech and speaker recognition," in *18th International Conference of the North American Fuzzy Information Processing Society-NAFIPS (Cat. No. 99TH8397)* (New York, NY: IEEE), 426–430.

Wang, Q., Jin, Y., and Li, P. (2015). "General-purpose LSM learning processor architecture and theoretically guided design space exploration," in *2015 IEEE Biomedical Circuits and Systems Conference (BioCAS)* (Atlanta, GA), 1–4. doi: 10.1109/BioCAS.2015.7348397

Wang, Q., and Li, P. (2016). "D-LSM: Deep liquid state machine with unsupervised recurrent reservoir tuning," in *2016 23rd International Conference on Pattern Recognition (ICPR)* (Cancun), 2652–2657. doi: 10.1109/ICPR.2016.7900035

Wijesinghe, P., Ankit, A., Sengupta, A., and Roy, K. (2018). An all-memristor deep spiking neural computing system: a step toward realizing the low-power stochastic brain. *IEEE Trans. Emerg. Top. Comput. Intell.* 2, 345–358. doi: 10.1109/TETCI.2018.2829924

Wijesinghe, P., Liyanagedera, C. M., and Roy, K. (2017). "Fast, low power evaluation of elementary functions using radial basis function networks," in *Proceedings of the Conference on Design, Automation & Test in Europe* (Lausanne: European Design and Automation Association), 208–213. doi: 10.23919/DATE.2017.7926984

Wijesinghe, P., Srinivasan, G., Panda, P., and Roy, K. (2019). Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines. *Front. Neurosci.* 13:504. doi: 10.3389/fnins.2019.00504

Zhang, W., and Li, P. (2019). Information-theoretic intrinsic plasticity for online unsupervised learning in spiking neural networks. *Front. Neurosci.* 13:31. doi: 10.3389/fnins.2019.00031

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 Neuroscience

ORIGINAL RESEARCH
published: 26 October 2020
doi: 10.3389/fnins.2020.00907

HFNet: A CNN Architecture Co-designed for Neuromorphic Hardware With a Crossbar Array of Synapses

Roshan Gopalakrishnan[1], Yansong Chua[1], Pengfei Sun[1], Ashish Jith Sreejith Kumar[1,2] and Arindam Basu[2]*

[1] Institute for Infocomm Research, Agency for Science, Technology and Research (ASTAR), Singapore, Singapore, [2] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, Singapore

The hardware-software co-optimization of neural network architectures is a field of research that emerged with the advent of commercial neuromorphic chips, such as the IBM TrueNorth and Intel Loihi. Development of simulation and automated mapping software tools in tandem with the design of neuromorphic hardware, whilst taking into consideration the hardware constraints, will play an increasingly significant role in deployment of system-level applications. This paper illustrates the importance and benefits of co-design of convolutional neural networks (CNN) that are to be mapped onto neuromorphic hardware with a crossbar array of synapses. Toward this end, we first study which convolution techniques are more hardware friendly and propose different mapping techniques for different convolutions. We show that, for a seven-layered CNN, our proposed mapping technique can reduce the number of cores used by 4.9–13.8 times for crossbar sizes ranging from 128 × 256 to 1,024 × 1,024, and this can be compared to the toeplitz method of mapping. We next develop an iterative co-design process for the systematic design of more hardware-friendly CNNs whilst considering hardware constraints, such as core sizes. A python wrapper, developed for the mapping process, is also useful for validating hardware design and studies on traffic volume and energy consumption. Finally, a new neural network dubbed HFNet is proposed using the above co-design process; it achieves a classification accuracy of 71.3% on the IMAGENET dataset (comparable to the VGG-16) but uses 11 times less cores for neuromorphic hardware with core size of 1,024 × 1,024. We also modified the HFNet to fit onto different core sizes and report on the corresponding classification accuracies. Various aspects of the paper are patent pending.

Keywords: neuromorphic computing, neuromorphic chip, hardware constraints, deep learning, neural network, crossbar array, convolution, convolutional neural network

OPEN ACCESS

Edited by:
Chiara Bartolozzi,
Italian Institute of Technology (IIT), Italy

Reviewed by:
Melika Payvand,
ETH Zurich, Switzerland
Guoqi Li,
Tsinghua University, China
Qingjiang Li,
National University of Defense Technology, China

***Correspondence:**
Yansong Chua
james4424@gmail.com

Specialty section:
This article was submitted to Neuromorphic Engineering, a section of the journal Frontiers in Neuroscience

Received: *06 November 2019*
Accepted: *04 August 2020*
Published: *26 October 2020*

Citation:
Gopalakrishnan R, Chua Y, Sun P, Sreejith Kumar AJ and Basu A (2020) HFNet: A CNN Architecture Co-designed for Neuromorphic Hardware With a Crossbar Array of Synapses. Front. Neurosci. 14:907. doi: 10.3389/fnins.2020.00907

1. INTRODUCTION

Over the past decade, GPUs have emerged as a major hardware resource for deep learning tasks. However, fields, such as the internet of things (IoT) and edge computing are constantly in need of more efficient neural-network-specific hardware (Basu et al., 2018; Deng et al., 2018; Alyamkin et al., 2019; Roy et al., 2019). This encourages competition among companies, such as Intel,

IBM, and others to propose new hardware alternatives, leading to the emergence of commercially available deep learning accelerators (Barry et al., 2015; Jouppi et al., 2017) and neuromorphic chips (Esser et al., 2016; Davies et al., 2018; Pei et al., 2019). Deep learning accelerators are application specific integrated circuits (ASICs) tailored for artificial neural networks (ANN), whereas, neuromorphic chips can fall in two categories (Bose et al., 2019): (1) ASICs with biologically inspired spiking neural networks (SNN), which contain networks of neurons and synapses for computation and communication, or (2) ASICs with analog computing by exploiting dense non-volatile memory based crossbars to accelerate matrix-vector multiplications. Our paper is not concerned with any specific hardware but any neuromorphic architecture relying on analog crossbars for matrix-vector multiplications.

A schematic of a generic crossbar-based neuromorphic chip is shown in **Figure 1**. The chip has "N" number of neuromorphic cores. Network on chip (NoC) or router interfaces are not shown for illustration purposes. Each neuromorphic core contains a crossbar array of synapses as shown in the first inset of the figure. The rows and columns of the crossbar correspond to input axons and output neurons, respectively. These axons and neurons are interconnected to each other and represented in the form of blue dots at these intersections. Within each intersection of the crossbar between the word line and the bit line is a synaptic device that has memory and can perform in-memory computation (as shown in the second inset). The crossbar structure is well-suited for performing matrix-vector multiplication (MVM) (Hu et al., 2016) along each column in a crossbar architecture. For instance, a neuromorphic core with a core size of 256 $\times$ 256, input voltages from the respective 256 axons are fed through each word line (red horizontal lines). The bit line (yellow vertical lines) collects all of the weighted current at each synaptic node (256 $\times$ 256) and delivers to the respective output neurons for integration. The neuromorphic core size refers to the number of axons (axon size, AS) $\times$ the number of neurons (neuron size, NS) in a single neuromorphic core. The weighted current depends on the memory element at each intersection of the word line and bit line. In analog devices, using Kirchoff's current law, the total current flowing into each neuron from the respective bit lines is the sum of currents flowing through each intersection in every column. This corresponds well with how inputs in a neural network is the weighted sum of input voltages ($\sum(Input \times Weight)$). Considering such a neuromorphic chip, there are several hardware constraints: firstly, at the single device, we may have low bit precision of synaptic weights and output activations (Ji et al., 2018; Deng et al., 2020), synaptic noise and variability (Ambrogio et al., 2014a,b). Secondly, in the chip architecture, we have a limited number of neuromorphic cores and a limitation in the core size of each neuromorphic core and the fan-in/fan-out degree of each neuron (Ji et al., 2018; Gopalakrishnan et al., 2019b).

The neuromorphic chip considered in this paper is based on a crossbar architecture (Prezioso et al., 2015) of non-volatile memory synapses. Crossbar architecture fits well for fully connected neural networks, such as the multi-layered perceptron (MLP). Given that one of the popularly used layers in SNNs are fully connected ones (Diehl and Cook, 2015), crossbar architectures are a natural fit. However, with recent advancement in research related to conversion of ANNs to SNNs (Rueckauer et al., 2016) and training of convolutional SNNs (Wu et al., 2019), one of the main challenges is to efficiently map the neurons in a CNN onto the neuromorphic chip while fulfilling hardware constraints, such as core size, number of cores and fan-in/fan-out (Ji et al., 2016). Given existing CNNs and neuromorphic hardwares, how can we best map the CNN onto the hardware using the least number of cores? This requires understanding of the computation at each crossbar array and how best to map each convolutional layer onto the core. We may then ask what convolution layers are best suited for mapping onto neuromorphic hardwares. This is the first major contribution of this paper and these questions are addressed under section Mapping. Existing neuromorphic chips have a mapping framework which is hardware specific. IBM TrueNorth (Akopyan et al., 2015) uses Matlab based Corelet (Amir et al., 2013) which is specific to their hardware. Within Corelet, a mapping technique is implemented as a minimization problem (Akopyan et al., 2015). SpiNNaker and BrainScaleS use a simulator-independent Python wrapper, PyNN (Andrew et al., 2009). Sequential mapping is used in SpiNNaker while neural engineering framework (NEF) is used for Neurogrid (Voelker et al., 2017). Neutrams (Ji et al., 2016) implements an optimized mapping technique based on the graph partition problem: Kernighan-Lin (KL) partitioning strategy for network on chip (NoC). For mapping CNNs onto neuromorphic hardwares, an iterative process is implemented using a Python wrapper, which is also discussed in section 2.3.2.

While developing deep neural networks that are to be mapped onto a neuromorphic chip, one need not in principle be aware of the underlying hardware architecture. The mapping above assumes that the CNNs and hardware constraints are given. We may however further ask how software and hardware co-design can give us both CNNs and neuromorphic hardwares that are mapped using fewer cores while achieving similar classification accuracies. Specifically, given a neuromophic hardware with square crossbar array, we would like to design a CNN that utilizes fewer cores (section Co-design). In this regard, one may take two approaches, either design the network from scratch so as to satisfy the hardware constraints (Esser et al., 2016) or modify an existing CNN, such as reducing the number of features (feature maps) in each convolution layer without having to split the convolution matrix among different cores ("core matrix splitting") whereby axons and weight matrix of a particular layer are split onto multiple cores (detailed in subsection 2.2.1). This is the second major contribution of this paper, and the proposed novel hardware-friendly CNN, HFNet, is obtained by iteratively modifying the layers of existing CNNs (VGG, MobileNet, NIN, and Squeezenet; this is discussed in section 2) and the number of features (feature maps) in some layers are altered so as to fit onto cores of different sizes. This is done to avoid core matrix splitting. Finally, the different versions of HFNet are trained and their classification accuracies on the IMAGENET dataset (Deng et al., 2009) tested so as to study the trade-off between accuracies and core sizes (section 3.2). This work is mainly focused on

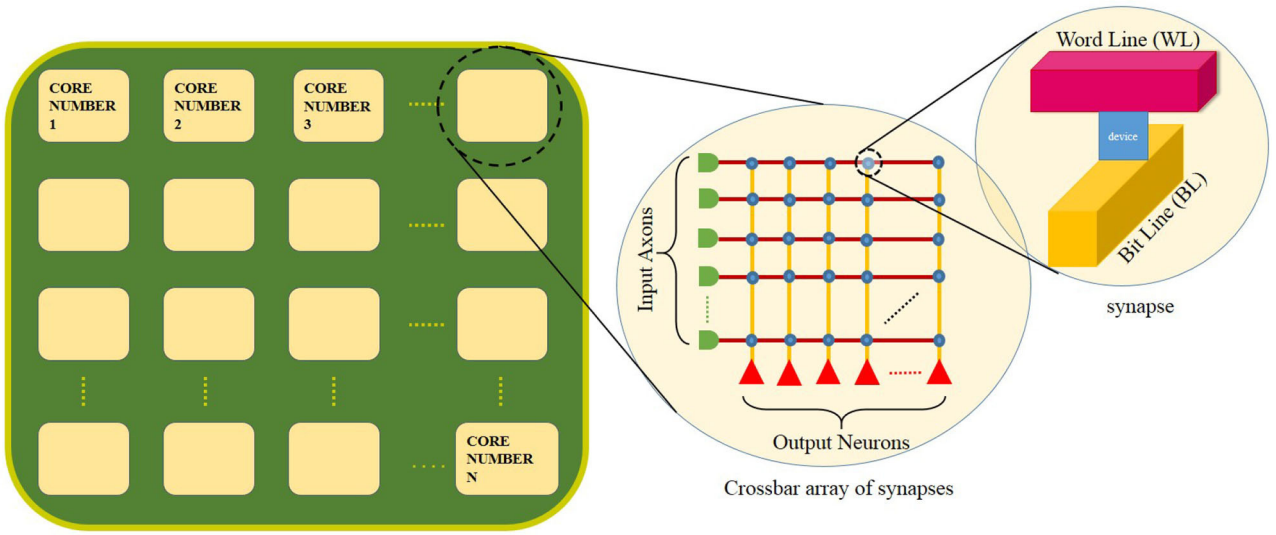


FIGURE 1 | A schematic of a neuromorphic chip with N number of neuromorphic cores. The first inset shows the crossbar array of synapses within each core. A memory device is used to implement each synapse at the crossbar intersection (as shown in the second inset).

mapping of feedforward deep neural networks (DNN). During the investigation of mapping techniques, we understood that designing and mapping of a CNN must be performed in close relation to each other for better hardware utilization. Hence, as a beginning, we have limited the design space to just feedforward networks instead of skipped connections. We have considered the better-performing feedforward CNN MobileNet as an initial candidate for mapping and later modified to HFNet based on mapping and traditional deep learning techniques.

The trained CNNs have full precision weights and activation values for fair comparison with existing CNNs. Hardware limitations on synapses, such as low precision weights and variability issues are not within the scope of our work.

The paper is organized as follows. Section 2 mainly contains two subsections, one on mapping and another on co-design. Mapping describes the computation and mapping in a crossbar array. Co-design is illustrated with the issue of core matrix splitting and then the motivation and design flow of the proposed hardware-friendly neural network, HFNet. Section 3 provides an experimental framework for the two subsections, mapping and co-design in section 2. The classification accuracy of different versions of HFNet on the IMAGENET dataset is included here. The paper is concluded in section 4 with a discussion of future works.

2. MATERIALS AND METHODS

2.1. Mapping

2.1.1. Computation in a Crossbar Architecture

The crossbar array of synapses in a neuromorphic chip can be used to perform convolutions. Mathematically, convolution is the sum of dot product of feature and input matrices (**Figure 2**). In CNNs, the input matrix will be the activations from the prior layer while the filter matrix is the convolution filter kernel, saved as weights, W after training. Since these weights can be either positive or negative, one way of implementing convolution on a crossbar array is to split the weights into positive and negative matrices along with two copies of input matrices in positive and negative values. The details of the matrix generation is shown in **Figure 2**, which incorporate the convolution operation in crossbar arrays as described in (Yakopcic et al., 2016). A single column crossbar gives the output of an element of the convolution operation, which is provided to the corresponding neuron. Convolution operation is extended across multiple columns of synapses to be computed in parallel. This requires the weights and inputs to be represented in a toeplitz matrix, as shown in **Figure 2** (Yakopcic et al., 2017), **Figure 3** illustrated such an implementation. This implementation doubles the hardware resources required, which is also the case in IBM Truenorth (Esser et al., 2016), where two synapses are required to implement the ternary weights −1, 0, +1. IBM Truenorth also uses toeplitz structure or structured kernels for mapping (Appuswamy et al., 2016). In order to mitigate the aforementioned doubling of hardware requirement in a neuromorphic hardware, one can implement two memory devices at each synapse to represent both the positive and negative weights by subtraction. This implementation does not need two copies of weights; generating a single weight toeplitz matrix is sufficient.

2.1.2. Mapping on a Crossbar Architecture

Crossbar architecture is efficient for implementing a fully connected neural network and its mapping is straightforward. However, mapping a convolution layer in a CNN onto a crossbar architecture is not and requires further consideration for efficient mapping. An example of convolution operation in between layers

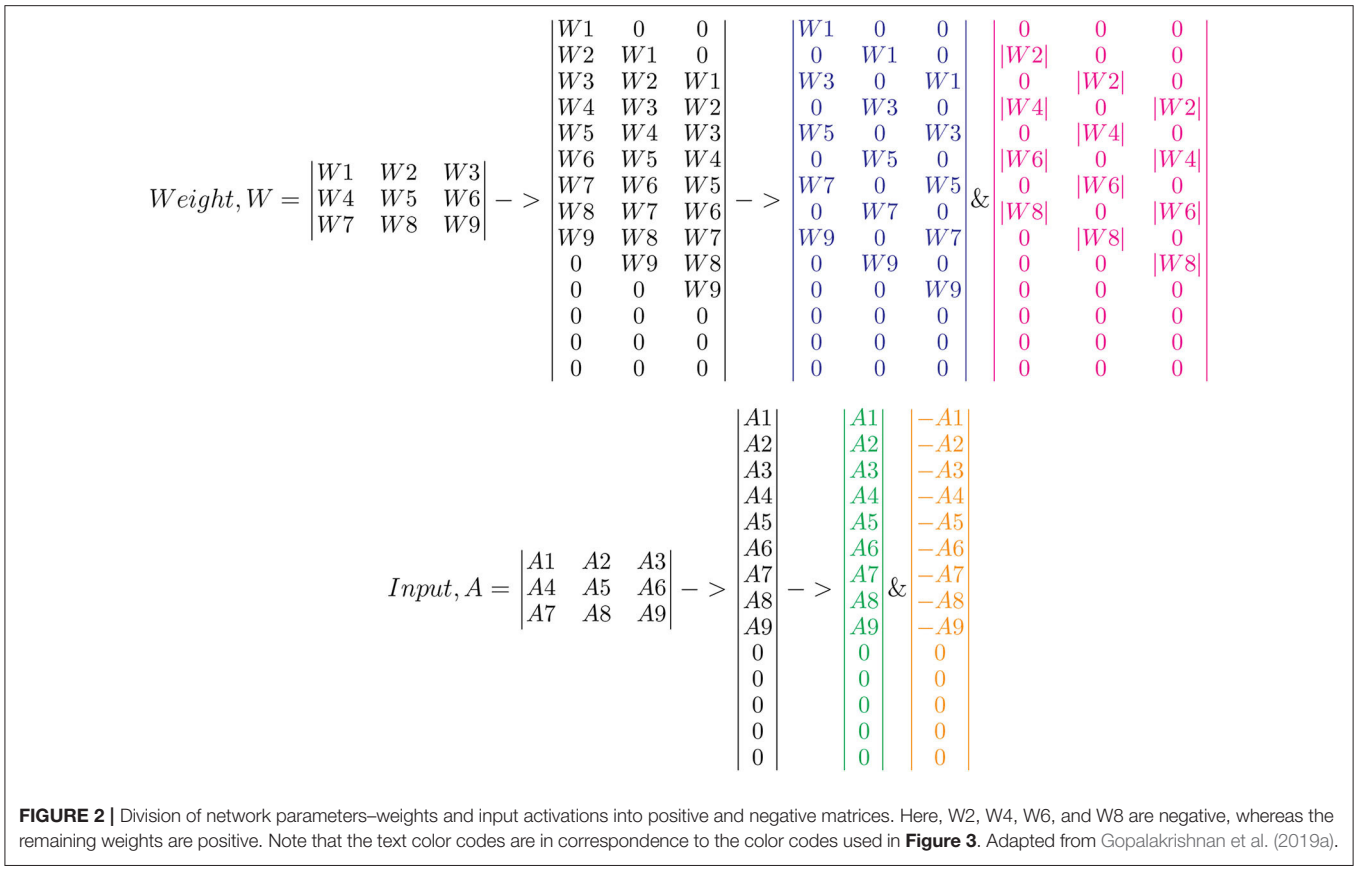


FIGURE 2 | Division of network parameters–weights and input activations into positive and negative matrices. Here, W2, W4, W6, and W8 are negative, whereas the remaining weights are positive. Note that the text color codes are in correspondence to the color codes used in **Figure 3**. Adapted from Gopalakrishnan et al. (2019a).

of a CNN is illustrated in **Figure 4A**. We consider a 4 × 4 input layer. Convolution of this input layer with two weight filters of size 2 × 2 and stride of 1 will result in an output layer of size 3 × 3 × 2. If this same network was considered fully connected between input layer of 16 neurons and output layer of 18 neurons, then, for such a network of full connection, a core size of 16 × 18 is required, which will have 100% synaptic utilization in the core. The mapping of CNNs onto a crossbar array will, however, require a layer-wise core size of 4 × 18 in the above example, while the input will have to be fed in over many time steps, in order to have 100% synaptic utilization (with duplication of synaptic weights). Hence, CNN mapping onto crossbar architecture will always lead to some weights not being used and we want to explore what mapping techniques can reduce this wastage. Mapping convolution onto a crossbar architecture can be constructed by any of the methods as shown in **Figure 4**. Note the numbering of neurons in each layer along with the color of each weight for better illustration of the different methods of mapping.

- Block method: as shown in **Figure 4B**, one could see that mapping is done in a straightforward manner without optimization. Here, the input sequence is repeated in the rows of a crossbar array; the neurons across the feature maps in a layer are arranged in the columns of the crossbar array and the weights are laid down according to the connections of the input axons and output neurons. This kind of implementation results in the weight matrices being mapped onto the crossbar array in blockwise manner, hence the name. Each block of weight matrix in **Figure 4B** are repeated from the same layers with the weight matrix being flattened into a row with size of kernel width × kernel height × number of feature maps (2 × 2 × 2) in the layer. Throughout the crossbar array these weight blocks (2 × 2 × 2) are repeated diagonally. In this method, one can find that the neurons across feature maps (N11 and N21 are the first and second neurons, respectively) are selected for mapping in the crossbar array and hence early layers of convolution which contain less feature maps maybe implemented using this method for better hardware utilization.
- Toeplitz method: the weight matrices are arranged in the toeplitz matrix or circular matrix format as shown in **Figure 4C**. This optimized method of mapping is commonly used in a neuromophic core with crossbar array of synapses. Here, the neurons are selected from a single feature map of a particular layer instead of selecting neurons across feature maps (note that the neurons are chosen from first channel of output layer in cyan color). The corresponding axons are selected from the previous layer and is arranged sequentially without any repetitions. The weight matrix per column of a crossbar array is the flattened structure of weight matrix in a particular layer of a neural network architecture. This weight matrix per column repeats along each columns in a circular

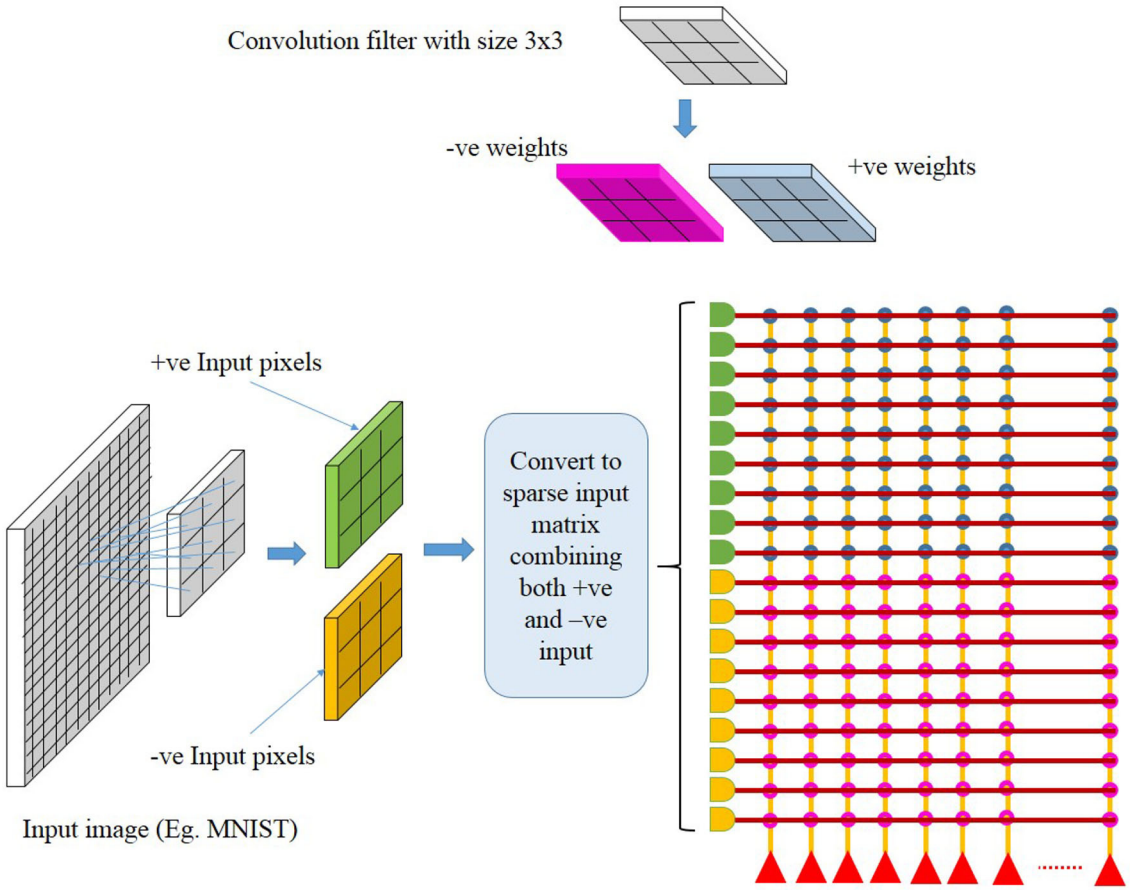


FIGURE 3 | Computation in a crossbar architecture within a single neuromorphic core. Follow the color codes representing the partitioning of input and weight matrices and its corresponding mapping onto the crossbar array. Taken from Gopalakrishnan et al. (2019a).

shift with respect to the strides of convolution in the particular layer as shown in **Figure 4C**. This method maps each feature map of a layer in crossbar arrays rather than mapping neurons across the feature maps as in block method. The toeplitz method of mapping is therefore suitable for certain type of convolutional layers, such as the depthwise convolutional layer, in which convolutions are separately performed in each feature map, independent of other feature maps within a layer (Howard et al., 2017; Gopalakrishnan et al., 2019a) or suitable for layer wise computations, such as pooling.

- Hybrid method: Considering the two aforementioned methods, one may combine the block and toeplitz methods of mapping in two different ways, as shown in **Figures 4D,E**. In **Figure 4D**, we select the neurons within a feature map (N11, N12, N13, etc. in cyan color from the same feature map of output layer) and lay down the weights in the toeplitz matrix manner. This toeplitz method is then repeated in a blockwise manner throughout the crossbar array, mapping a set of neurons across the feature maps of a particular layer in the neural network architecture. This can be viewed as implementing the toeplitz method in a blockwise manner. We can also implement block method in the toeplitz manner as shown in **Figure 4E**, where the neurons across a feature map (N11 in cyan and N21 in magenta as group of neurons from different feature maps of output layer) are selected for mapping using the block method, though without any repeated axons, while the entire block is repeated in a toeplitz manner throughout the crossbar array, mapping a set of neurons across the feature maps of a convolutional layer.

2.2. Co-design

2.2.1. Core Matrix Splitting

For the toeplitz and hybrid method of mapping techniques, as shown in **Figure 4**, the axons of the neuromorphic core are arranged sequentially without any repetition as compared to the block method of mapping. Such sequential input of axons is possible only with the circular or toeplitz arrangement of weight filters in a crossbar architecture. In fact, the mapping of CNN layers onto crossbar architecture involves conversion of two-dimensional arrays into one-dimensional arrays. The two-dimensional convolutional operations in a CNN is converted to a one-dimensional convolutional operation along each columns of a crossbar architecture. The neurons in the output layer

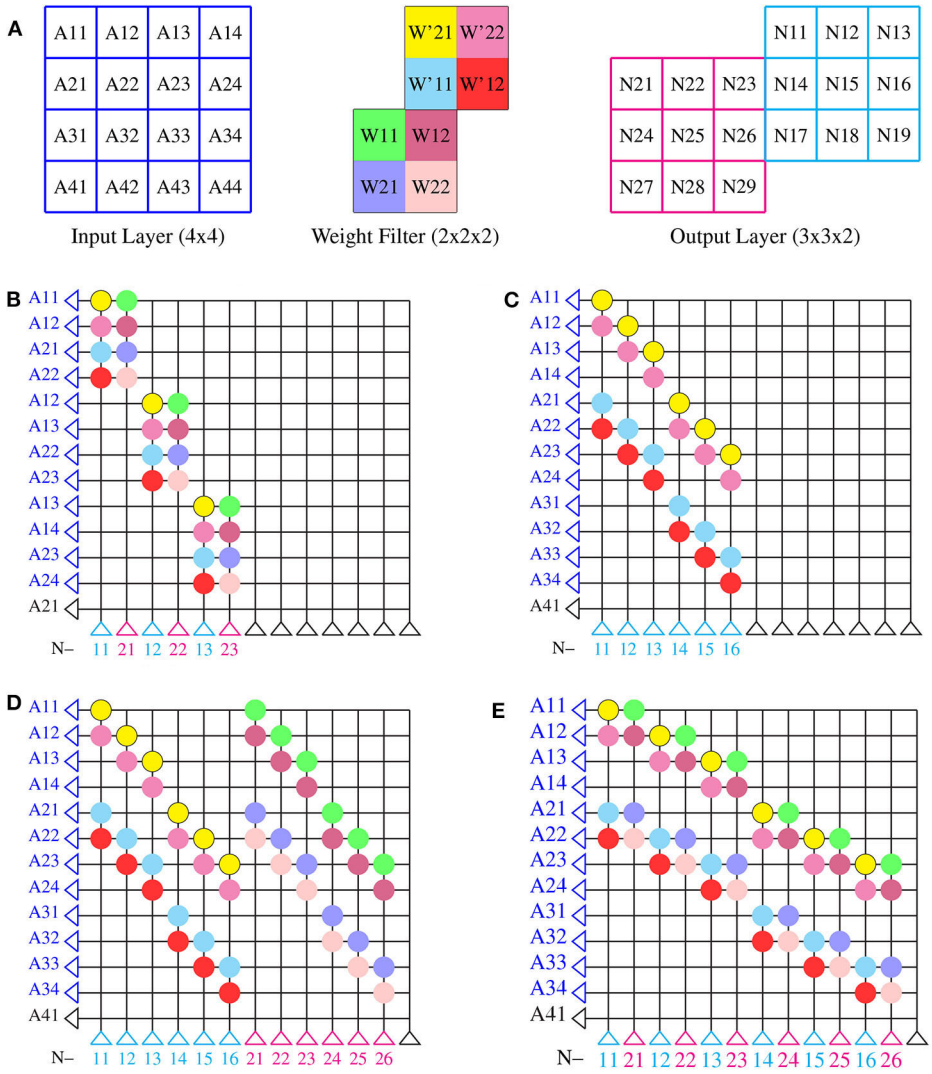


FIGURE 4 | Methods of mapping onto a single neuromorphic core: **(A)** an example of convolution operation between layers for illustrating different mapping techniques. Input of size 4 × 4 and filter weights of size 2 × 2 × 2 is considered for convolution to obtain an output of size 3 × 3 × 2. **(B)** Block method, **(C)** toeplitz method and hybrid method, **(D)** toeplitz method in blockwise manner, and **(E)** block method in toeplitz manner.

is mapped onto the crossbar array with its corresponding one-dimensional array of input axons in a sequence and extending the one-dimensional weight matrix along each column of crossbar by circularly shifting the one-dimensional weight matrix with respect to corresponding convolutional strides (as shown in **Figure 4C**). Two adjacent convolutional operation shares a portion of input section with respect to the strides. This shared portion of input while convolution is reflected as weight connections along the rows of the core. The number of weight connections along each row implies the fan-out of that particular axon in the core, whereas the weight connections along the column implies the fan-in of that particular neuron in the core. For the toeplitz method of mapping, a section of any CNN layer with a rectangular dimension of $neuron_{row}$ and $neuron_{col}$ to a crossbar array, the number of axons to be selected for such a section of CNN layer can be expressed mathematically:

$$N_{axons} = K \times K + K \times S \times (Neuron_{col} - 1) + S \times S \times (Neuron_{col} - 1) \times (Neuron_{row} - 1) + K \times S \times (Neuron_{row} - 1) \quad (1)$$

where,

N_{axons} = total number of axons to be selected from layer N-1.

K = convolution filter size. Here, we have only considered same width and height for kernel filter size.

S = stride

$Neuron_{row}$ = number of neurons across row to be selected from layer N.

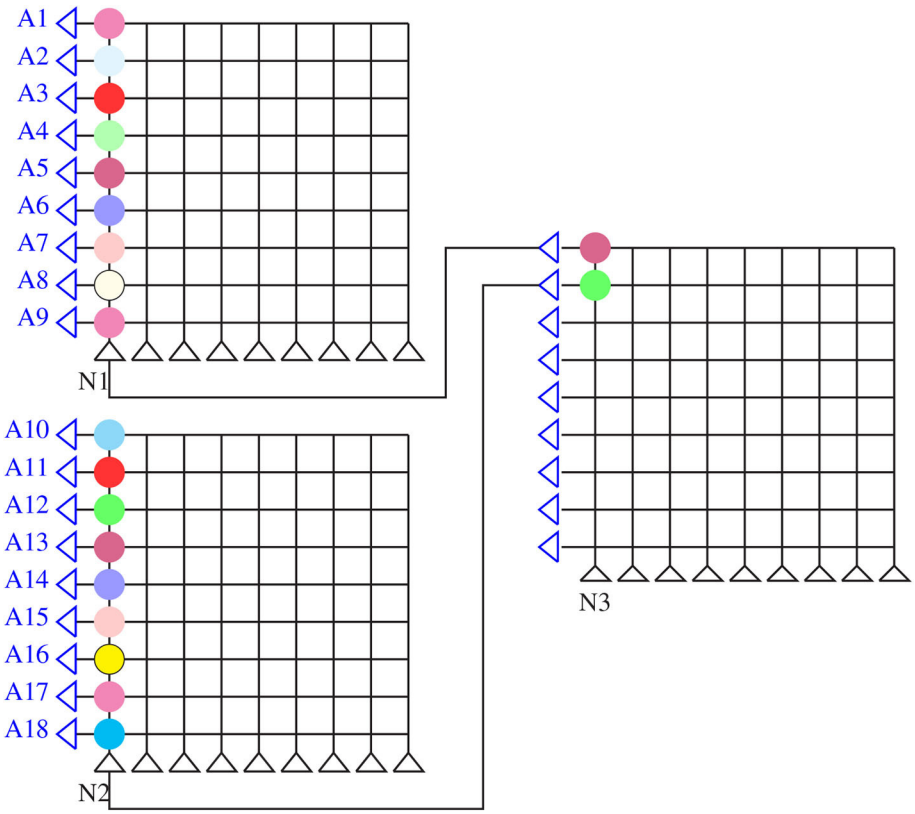


FIGURE 5 | Core matrix splitting: splitting of weighted sum of inputs into two cores and summing up of intermediate activations in a third neuromorphic core.

$Neuron_{col}$ = number of neurons across column to be selected from layer N.

To decide on core size, a major restriction comes from the below inequality:

$$N_{axons} < Axon\ \ size, AS \tag{2}$$

where Axon size and AS the number of physical axons per core. In general, we would like to minimize AS to have smaller cores, and we thus need to minimize N_{axons}. In this case, the selection of neurons, $Neuron_{row}$ and $Neuron_{col}$, in a layer becomes primary step in mapping. To perform the optimization, we further consider that output size (O_{size}) is constant, i.e.,

$$O_{size} = Neuron_{row} \times Neuron_{col} = A \tag{3}$$

where A ∈ R is a constant. The optimization problem can be now framed as choosing $Neuron_{row}$ and $Neuron_{col}$ to minimize Equation (1) subject to the constraint in Equation (2). Do note that Equation (1) considers only a single feature map; this can be easily extended to multiple feature maps by multiplying right hand side of Equation (1) with the respective number of feature maps in each layer.

Now, denoting $Neuron_{row}$ and $Neuron_{col}$ by x and y, respectively, for brevity of notation, Equation (1) can be reduced as follows:

$$\begin{aligned} N_{axons} &= K \times K + K \times S \times (Neuron_{col} - 1) + S \times S \times \\ &\quad (Neuron_{col} - 1) \times (Neuron_{row} - 1) + \\ &\quad K \times S \times (Neuron_{row} - 1) \\ &= K \times K + K \times S \times (y-1) + S \times S \times (y-1) \times (x-1) \\ &\quad + K \times S \times (x-1) \\ &= K^2 + KS(x+y-2) + S^2(xy - y - x + 1) \\ &= (K^2 - 2KS + AS^2 + S^2) + (x+y)(KS - S^2) \end{aligned} \tag{4}$$

where we have used xy = A from Equation (3). Since (KS $-S^2$) > 0, minimizing Equation (4) is equivalent to minimizing (x+y). We show in the following theorem that (x+y) is minimized for x = y.

Theorem: For any given a ∈ R, then x = y, for argmin (x+y), such that x × y = A, x,y ∈ R.

Proof:

$$\begin{aligned} &Let\ \ x + y = Z \\ &then\ \ x + \tfrac{A}{x} = Z \\ &\tfrac{dZ}{dx} = 1 - \tfrac{A}{x^2} \\ &at\ \ minima\ \ \tfrac{dZ}{dx} = 0, \\ &\therefore\ \ 1 - \tfrac{A}{x^2} = 0 \end{aligned} \tag{5}$$

$$x = \pm\sqrt{A}$$
$$If \;\; x, y > 0, \;\; then \;\; Z \;\; is \;\; minimum \;\; \therefore x = y = \sqrt{A}$$

From the theorem and its proof, we can see that for optimized mapping, a square shaped selection of neurons is always better. This implies that, for optimized mapping, $Neuron_{row} = Neuron_{col}$ in Equation (1). Hence, substituting $Neuron_{row} = Neuron_{col} =$ "$N_{neurons}$" in Equation (1), we get,

$$N_{neurons} = (\sqrt{N_{axons}} - K)/S + 1 \tag{6}$$

If we consider input channels in Equation (1), then the above equation becomes

$$N_{neurons} = (\sqrt{N_{axons}}/C - K)/S + 1 \tag{7}$$

where, C = number of input channels for each layer.

This equation can be used to determine the design co-mapping of the convolutional neural network onto a neuromorphic chip with crossbar array of synapses, where the hardware constraint is given by the axon size and the convolution layer design is with respect to K, S, and C for each layer. This suggests that mapping and designing of a CNN must co-exist for the better utilization of a neuromorphic hardware.

It can also be seen from the optimized mapping that the time delay of the hardware design is less. The design space exploration of core size, [axon size (AS) × neuron size (NS)] w.r.t N_{axons}, $Neuron_{row}$, and $Neuron_{col}$ becomes $[N_{axons} \times (Neuron_{row} \times Neuron_{col})] = [N_{axons} \times N^2_{neurons}]$. This implies that the search space for neuron size is reduced to only square numbers ($N^2_{neurons}$) instead of all the factors ($Neuron_{row} \times Neuron_{col}$) of the neurons in a core (NS = 256, 512, etc). The fact that a unique HFNet is trained for each topology saves on many days of training.

In the event that the fan-in degree of a single neuron in a layer exceeds the maximum number of axons in a neuromorphic core [$N_{axons} >$ AS in Equation (1) when $Neuron_{row} = 1$ and $Neuron_{col} = 1$], mapping of that particular neuron has to be split among multiple cores, as shown in **Figure 5**. In general, if the output of each neuron undergoes a non-linear activation, the final output would deviate from the intended output:

$$\begin{aligned} &given, \quad W = (W_1, W_2), \\ &f(f(\sum W_1 A) + f(\sum W_2 A) \neq f(\sum WA)) \end{aligned} \tag{8}$$

This method of splitting is termed as core matrix splitting (**Figure 5**). Additional hardware considerations have to be made to communicate only intermediate results without the activation function. Additional cores are also required for mapping. In order to avoid core matrix splitting, a hardware-friendly approach is considered by grouping neurons across feature maps while training in the IBM TrueNorth chip (Esser et al., 2016). In this work, we adopt a different approach by modifying existing CNN architectures and training them. We consider other forms of convolution operations, such as depthwise and pointwise convolutions [network-in-network (Lin et al., 2013) and MobileNet (Howard et al., 2017)] while co-designing the CNN with the core size in mind so as to avoid core matrix splitting.

2.2.2. Overview of Convolution Layers in a CNN

Standard convolution can be computationally intensive and also hard to map onto square neuromorphic cores of limited sizes. To map them, we would have to consider cores of different shapes. Given these square cores, we considered other computationally less intensive convolution techniques, namely, depthwise, pointwise, and group convolution.

2.2.2.1. Depthwise convolution

In depthwise convolution, the convolution operation is independently applied to each input channel so as to obtain its corresponding output feature map (Howard et al., 2017). In general, the number of output channels from a depthwise convolution is the same as the number of input channels, although this may be changed by outputting multiple channels per input channel (depth multiplier parameter). The depthwise convolution is typically followed by pointwise convolution, which is discussed in section 2.2.3. For depthwise convolution, weights from within rather across feature maps are mapped first. Hence, the toeplitz method is better suited for mapping depthwise convolutions.

As shown in **Figure 6**, the input matrix is convolved with F^{in}_{maps} different filters, each of size K × K. The output of each depthwise convolution involving a filter and a single input channel is O × O × 1, and F^{in}_{maps} such filters compute an output of dimensions O × O × F^{in}_{maps}. The depth multiplier is set to one here. The computational cost, C, of depthwise convolution is given below:

$$C = O^2 \times K^2 \times F^{in}_{maps} \times D \tag{9}$$

where,
O = output size after convolution
K = filter size
F^{in}_{maps} = number of input channels
D = depth multiplier.

2.2.2.2. Pointwise convolution

Pointwise convolution is a special case of the standard convolution operation whereby the filter size per channel is 1 × 1 (Lin et al., 2013). The entire filter therefore has a shape of 1 × 1 × F^{in}_{maps} × F^{out}_{maps}, where F^{in}_{maps} is the number of input channels and F^{out}_{maps}, the number of output channels. Since the filter size is reduced, the computational complexity is also reduced by an order of the square of the filter size. Its computational cost, C, is given below:

$$C = O^2 \times F^{in}_{maps} \times F^{out}_{maps} \tag{10}$$

where,
O = output size after convolution
F^{in}_{maps} = number of input channels
F^{out}_{maps} = number of output channels.

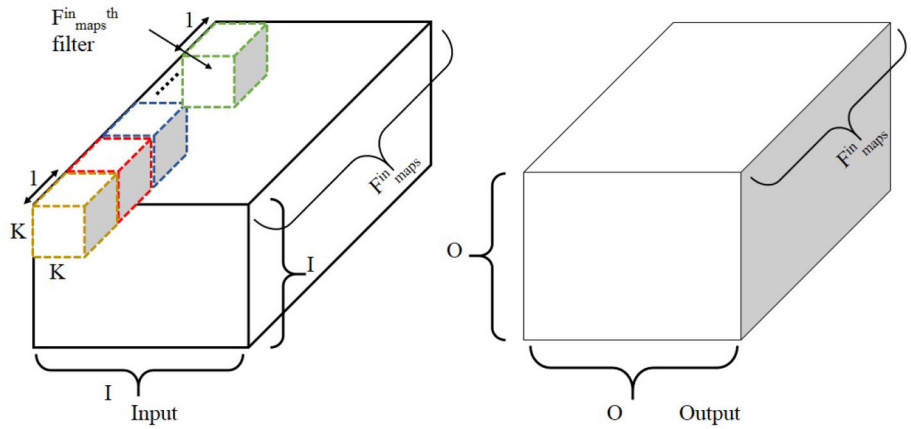


FIGURE 6 | Illustration of depthwise convolution. Note that depth multiplier is set to one here. The filter size is K × K × 1 with F^{in}_{maps} such filters to obtain an output size of O × O×F^{in}_{maps}. Adapted from Gopalakrishnan et al. (2019a).

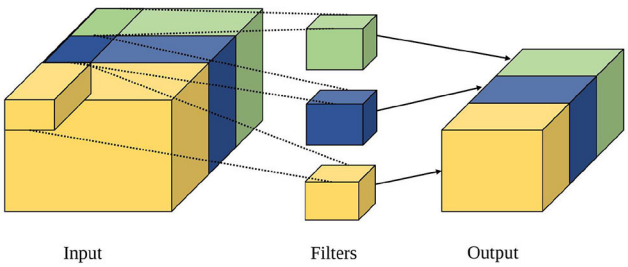


FIGURE 7 | Illustration of group convolution adapted from Gopalakrishnan et al. (2019a).

While we can use either the toeplitz or hybrid method of mapping for pointwise convolution layers depending on the CNN architecture and the core sizes, the block method in toeplitz is preferred if AS < NS; otherwise the toeplitz method in block is preferred (AS > NS).

2.2.2.3. Grouped convolution

Grouped convolution is a convolution technique whereby the standard convolution is applied separately to an input matrix diced into equal parts along the channel axis. As shown in **Figure 7**, the input is divided into equal parts along the channel axis, and group convolution is then applied separately. The individual outputs are then combined into a final output, with variations, such as stacked convolution, dependent stacked convolution and shuffled group convolution (Zhang and Sun, 2018). Computational complexity of grouped convolution is calculated as per standard convolution. It is therefore more hardware friendly as each neuron has a lower fan-in/fan-out degree when mapped. Either the toeplitz or hybrid method of mapping may be used for grouped convolutions.

2.2.3. Insights From Different CNNs

2.2.3.1. MobileNet

Depthwise and pointwise convolutions and depthwise separable convolutions are introduced in MobileNet (Howard et al., 2017). For pointwise convolution, one may think of it as duplicates of full connections between inputs and outputs in the same location channel-wise. This is ideal for efficient mapping onto crossbar array of synapses with good core utilization. The application of depthwise and then pointwise convolution (depthwise separable convolution) has a much lower fan-in degree per neuron compared to the standard convolution, which helps to avoid core matrix splitting. As such, depthwise separable convolution is the preferred method in our co-design of CNN architectures.

2.2.3.2. VGGNet and NIN

VGGNet (Simonyan and Zisserman, 2014) gradually shrinks the size of feature maps by applying max pooling after two convolutions in the shallow layers and every three layers afterwards. Intuitively, this approach improves classification accuracy, which we also validate as shown in results in section 3. Using fewer feature maps when each map is large effectively reduces the fan-in degree of the neurons and avoid core matrix splitting.

2.2.3.3. Other insights

Global average pooling (GAP) as used in Network in network (NIN) (Lin et al., 2013) or SqueezeNet (Iandola et al., 2016) helps to reduce fan-in degree of neurons. Instead of using fully connected layers in the deeper layers of the CNN, which have high fan-in degree, one may use GAP for a more hardware-friendly design. Maxpooling is also not hardware friendly. The toeplitz mapping method is required for maxpooling, resulting in poor core utilization. We would therefore avoid maxpooling when co-designing the hardware-friendly CNN. The reduction in feature map size achieved by maxpooling may also be achieved by increasing the stride size of prior convolution layer with no significant loss in accuracy, even if functionally, they are different

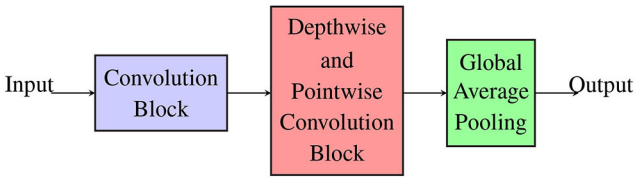


FIGURE 8 | Hardware-friendly CNN in modular form: standard convolution, depthwise separable convolution, and global average pooling.

(Springenberg et al., 2014). Hence, the above insights (listed below) help to guide our co-design of the hardware-friendly CNN (section 2.2):

- Combined usage of standard convolution and depthwise separable convolutions,
- Excluding pooling between standard convolutions,
- And replacement of fully connected layers with GAP at end of CNN.

2.2.4. Co-design Methodology

2.2.4.1. Hardware-friendly CNN

We adopt the modular form of the CNN as shown in **Figure 8** for a hardware-friendly architecture. The iterative process of the co-design is described in **Figure 9**. We first initiate certain design parameters, such as "N_C" number of convolutional layers and "N_{DP}" number of depthwise separable convolution layers. Setting N_C and N_{DP} depends on the input size and the stride size of each convolution, which affects the classification accuracy of the CNN. The singular GAP layer is added to end of the CNN.

The complete step by step co-design process is shown in the flowchart (**Figure 9**):

- After setting the number of layers in each module, we next decide on the number of feature maps per layer, with the constraint of avoiding core matrix splitting (**Figure 9**). $F^i_{maps\_HF}$ (HF denotes hardware friendly) for each layer i depends on the number of axons in each core (axon size, AS) and constraining the fan-in degree of a neuron in i (fan-in $\leq$ AS) to avoid core matrix splitting. For the standard convolution layer, the number of feature maps denoted by F_{maps} can be calculated as below:

$$F_{maps} \leq \frac{AS}{K_{width} \times K_{height}} \tag{11}$$

where,

K_{width} and K_{height} are respectively the width and height of the convolution kernel.

AS = axon size of crossbar array in a neuromorphic core.

Equation (11) is the maximum possible number of feature maps, F_{maps}, when in Equation (1) we set $Neuron_{row} = 1$ and $Neuron_{col} = 1$. By so doing, we set the reference number of feature maps in i, $F^i_{maps\_ref}$ to be less than the fan-in (maximum possible axon connections) degree of a neuron. In such a case, toeplitz mapping may also be used.

- For pointwise convolution, we need to ensure that $F_{maps} \leq$ axon size. Core matrix splitting is not an issue for depthwise convolutions with small kernel size. Hence, for depthwise separable layers, fan-in degree of pointwise convolution is the key delimiting factor. The co-designed CNN is then mapped to obtain number of cores used, and it is then trained and tested for classification accuracy (section 2.3). If it is not satisfactory, the process is repeated with different initial parameters, N_C and N_{DP}.

2.3. CNN Training and Mapping

2.3.1. CNN Training

The co-designed CNNs are trained using Tensorpack (Wu, 2016). We use "Momentum" Optimizer (momentum of 0.9) with batch size of 48, and weight regularization with decay of 0.0005. We initialized the weights using "He normal" initializer. The learning rate is adjusted using a heuristic whereby it is reduced by 10 every 30 epochs. The learning rate was initialized at 0.01 and reduced three times prior to termination. We trained the network for 100 epochs on the IMAGENET dataset. Accuracy is chosen from the best testing accuracy across five trials.

2.3.2. CNN Mapping

The calculation for the number of cores is obtained using the python wrapper, mapping, and debugging (MaD) framework (Gopalakrishnan et al., 2019b), which also map the CNN onto cores. The mapping function outputs the weight matrices for the crossbar array in each core, a connectivity list between cores and an estimate of total number of cores needed for mapping. MaD also allows us to carry out inference in Python on the core level, which is useful for validating the correctness of the mapping done, and also study communication across cores, such as traffic volume and energy consumption estimation.

The methods and techniques mentioned in this paper is not restricted to any kind of neural network like ANN or SNN. The mapping and designing methods are useful for both ANN and SNN, especially with convolutional layers in their architecture. In fact, the simulations are all done for ANN models and not SNN. All the codes used for generating results in this manuscript is publicly available at https://github.com/roshan-gopalakrishnan/NeuromorphicComputing.git.

3. EXPERIMENTAL FRAMEWORK AND RESULTS

This section is also divided into two subsections: mapping and a proposed CNN, the HFNet. In section 3.1, we benchmark the different mapping techniques based on cores used. In 3.2, we propose a hardware-friendly CNN, the HFNet, and report on (1) the cores required for mapping, (2) classification accuracy and cores required with and without maxpooling and full connections, (3) classification accuracy and cores required for different core sizes, (4) comparison of the MobileNet and HFNet, and (5) the results when grouped convolution replaces depthwise separable convolution.

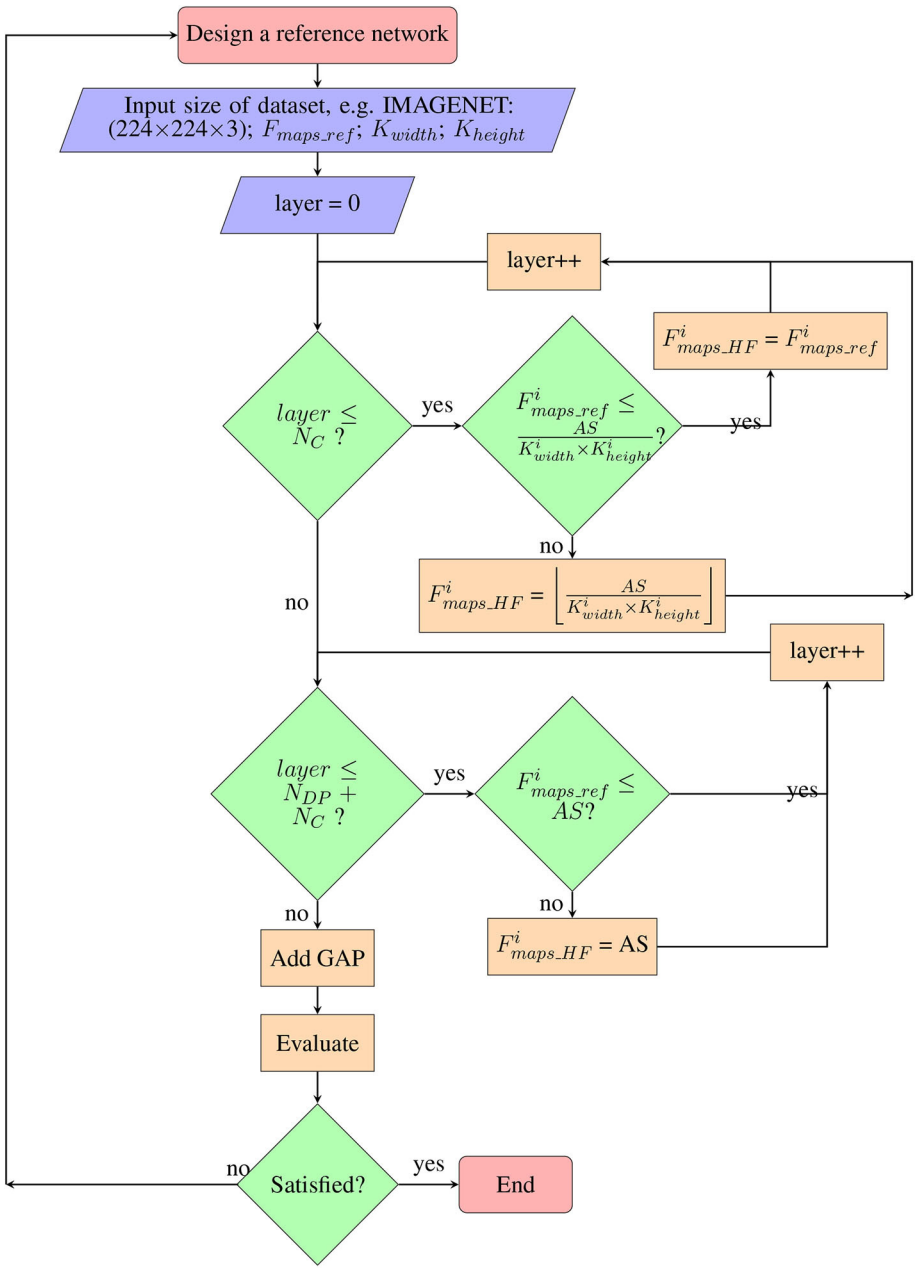


FIGURE 9 | Co-design flowchart: step by step methodology for co-design of hardware-friendly CNN.

3.1. Mapping Results

We first study the advantage of using the hybrid method over the toeplitz method of mapping a CNN for classifying CIFAR-10 (Krizhevsky, 2009) (**Figure 10**). As mapping of pooling layers is done with the toeplitz method and mapping fully connected layer requires core matrix splitting, we consider only the convolutional layers for illustrative purposes. The CNN is designed in the following manner: 32 × 32 × 3 (stride 2) – 16 × 16 × 4 – 14 × 14 × 8 – 12 × 12 × 12 – 10 × 10 × 16 – 8 × 8 × 20 – 6 × 6 × 24 – 4 × 4 × 28. All 7 convolutional layers have kernel filter size 3 × 3 and stride of 1, unless otherwise stated. We consider four different core sizes for mapping: 128 × 256 (similar to IBM TrueNorth), 256 × 256 (NC chip-V1), 512 × 512 (NC chip-V2), and 1,024 × 1,024 (NC chip-V3). The benchmarking metric is the number of cores needed to map the CNN. From the bar graph in **Figure 10**, it is observed that hybrid method always utilize less number of cores compared to the toeplitz method and number of cores required decrease with core sizes, irrespective of mapping methods. Quantitatively, 13.88, 15, 16.98, and 4.94 times fewer cores are utilized in the case of hybrid mapping compared to

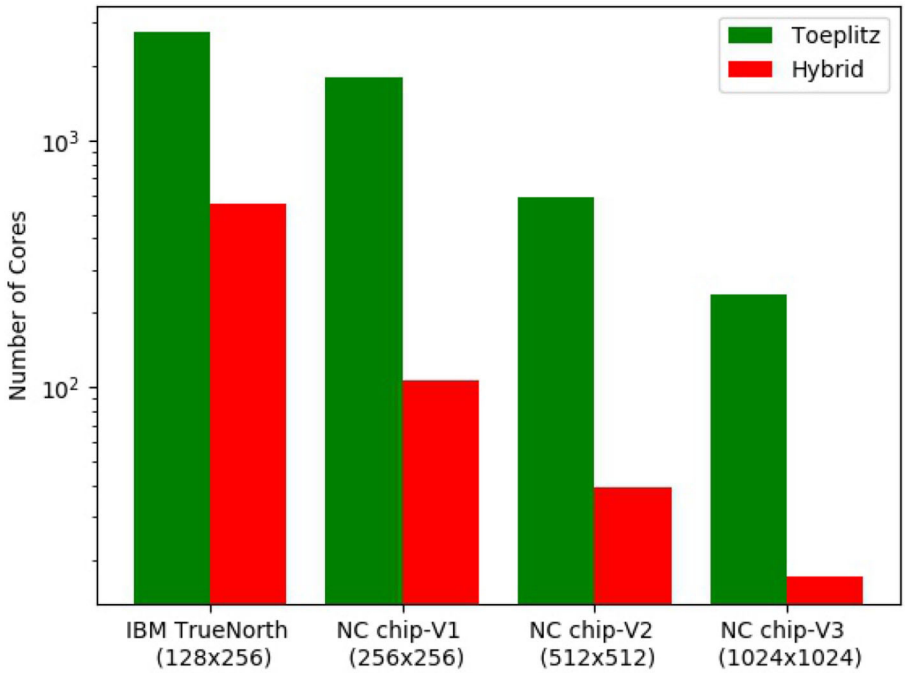


FIGURE 10 | Benchmarking of mapping methods. The bar graph shows the number of cores utilized for both hybrid and toeplitz mapping against different core sizes. We consider only the convolution layers in the CNN.

toeplitz mapping on neuromorphic chip with core sizes 128 × 256, 256 × 256, 512 × 512, and 1,024 × 1,024, respectively.

3.2. Hardware-Friendly CNN: The HFNet

Using the co-design process as described in section 2.2, we propose the HFNet. It is a hybrid architecture, based on our insights (**Figure 8**), that borrows from the VGGNet, MobileNet, and NIN so as to improve mapping on neuromorphic cores. As shown in **Figure 8**, the shallow layers are standard convolutional layers (VGGNet), followed by depthwise separable convolutions (MobileNet), and fully connected layers with large fan-in degrees are replaced with GAP (NIN).

The detailed input, output size of each layer in the HFNets for different core sizes are given in **Table 1**. We have named different versions of the HFNets for different core sizes (256 × 256, 512 × 512, and 1,024 × 1,024, respectively) as HFNet-V1, HFNet-V2, and HFNet-V3. Core size determines the number of axons (AS) × number of neurons (NS) in a single core. The second convolution layer in HFNet-V2 (1) has $F_{maps} = 56$, which is the maximum it can have (Equation 11) to avoid core matrix splitting. A kernel size of 3 × 3 is used for standard and depthwise convolution.

3.2.1. Maxpooling and Fully Connected Layers

This experiment investigates the performance of the proposed architecture with and without maxpooling and fully connected layers (**Table 1**). HFNet-V3 with maxpooling replaces all convolution layers with stride of two with convolution layer with stride of 1 and an additional maxpooling layer. For HFNet-V3 with full connections, a fully connected layer (1,024 × 1,000) is added on top of the average pooling layer in HFNet-V3 while changing the last pointwise convolution layer to 7 × 7 × 1,024 instead of 7 × 7 × 1,000. From **Table 2**, it can be seen that there is only very slight improvement in classification accuracies when maxpooling or full connection is used. This further validates our design criteria for HFNet. The number of cores required for HFNet-V3 with pooling layer is huge, as mapping of pooling layers is done using the toeplitz method. The number of parameters is higher for HFNet-V3 with fully connected layer. All CNNs considered in this experiment is illustrated in **Supplementary Material**.

3.2.2. Number of Cores and Classification Accuracy

Here we study both the number of cores required for mapping and the classification accuracy (IMAGENET) for the HFNets (**Figure 11**) and three other popular CNN architectures, namely VGGNet (VGG-16), MobileNet, and REMODEL [a modification of VGG-16 for mapping the final fully connected layers onto IBM TrueNorth (Shukla et al., 2019)]. We consider two core sizes: the minimum 128 × 256 and the maximum size 1,024 × 1,024. Note that VGG-16 and REMODEL require core matrix splitting for mapping onto a 1,024 × 1,024 core. Mapping of all CNNs onto 128 × 256 cores requires core matrix splitting. As expected, the number of cores used by MobileNet and HFNet are ∼10 times fewer compared to VGG-16 and REMODEL. It can be seen that HFNet-V1 uses the least number of cores among all models. The table (**Figure 11**) shows the classification accuracy of the CNNs on IMAGENET. The HFNet-V3 is as accurate

TABLE 1 | Neural network architecture (NN archi.) for different core sizes.

NN	HFNet model (core size)					
archi.	HFNet-V1 (256 × 256)		HFNet-V2 (512 × 512)		HFNet-V3 (1,024 × 1,024)	
Layers	Input size	Output size	Input size	Output size	Input size	Output size
C[a]	226 × 226 × 3	112 × 112 × 16	226 × 226 × 3	112 × 112 × 32	226 × 226 × 3	112 × 112 × 32
C	114 × 114 × 16	56 × 56 × 28	114 × 114 × 32	56 × 56 × 56	114 × 114 × 32	56 × 56 × 64
C	58 × 58 × 28	28 × 28 × 64	58 × 58 × 56	28 × 28 × 256	58 × 58 × 64	28 × 28 × 256
D[b]	30 × 30 × 64	28 × 28 × 64	30 × 30 × 256	28 × 28 × 256	30 × 30 × 256	28 × 28 × 256
P[c]	28 × 28 × 64	28 × 28 × 256	28 × 28 × 256	28 × 28 × 256	28 × 28 × 256	28 × 28 × 256
D	30 × 30 × 256	14 × 14 × 256	30 × 30 × 256	14 × 14 × 256	30 × 30 × 256	14 × 14 × 256
P	14 × 14 × 256	14 × 14 × 256	14 × 14 × 256	14 × 14 × 512	14 × 14 × 256	14 × 14 × 512
D	16 × 16 × 256	14 × 14 × 256	16 × 16 × 512	14 × 14 × 512	16 × 16 × 512	14 × 14 × 512
P	14 × 14 × 256	14 × 14 × 256	14 × 14 × 512	14 × 14 × 512	14 × 14 × 512	14 × 14 × 512
D	16 × 16 × 256	14 × 14 × 256	16 × 16 × 512	14 × 14 × 512	16 × 16 × 512	14 × 14 × 512
P	14 × 14 × 256	14 × 14 × 256	14 × 14 × 512	14 × 14 × 512	14 × 14 × 512	14 × 14 × 1,024
D	16 × 16 × 256	14 × 14 × 256	16 × 16 × 512	14 × 14 × 512	16 × 16 × 1,024	14 × 14 × 1,024
P	14 × 14 × 256	14 × 14 × 256	14 × 14 × 512	14 × 14 × 512	14 × 14 × 1,024	14 × 14 × 1,024
D	16 × 16 × 256	14 × 14 × 256	16 × 16 × 512	14 × 14 × 512	16 × 16 × 1,024	14 × 14 × 1,024
P	14 × 14 × 256	14 × 14 × 256	14 × 14 × 512	14 × 14 × 512	14 × 14 × 1,024	14 × 14 × 1,024
D	16 × 16 × 256	14 × 14 × 256	16 × 16 × 512	14 × 14 × 512	16 × 16 × 1,024	14 × 14 × 1,024
P	14 × 14 × 256	14 × 14 × 256	14 × 14 × 512	14 × 14 × 512	14 × 14 × 1,024	14 × 14 × 1,024
D	16 × 16 × 256	7 × 7 × 256	16 × 16 × 512	7 × 7 × 512	16 × 16 × 1,024	7 × 7 × 1,024
P	7 × 7 × 256	7 × 7 × 1,000	7 × 7 × 512	7 × 7 × 1,000	7 × 7 × 1,024	7 × 7 × 1,024
D	9 × 9 × 1,000	7 × 7 × 1,000	9 × 9 × 1,000	7 × 7 × 1,000	9 × 9 × 1,024	7 × 7 × 1,024
P	7 × 7 × 1,000	7 × 7 × 1,000	7 × 7 × 1,000	7 × 7 × 1,000	7 × 7 × 1,024	7 × 7 × 1,000
GAP[d]	7 × 7 × 1,000	1 × 1 × 1,000	7 × 7 × 1,000	1 × 1 × 1,000	7 × 7 × 1,000	1 × 1 × 1,000

[a] *Convolution layer.*
[b] *Depthwise convolution layer.*
[c] *Pointwise convolution layer.*
[d] *Global average pooling.*

TABLE 2 | With and without pooling and fully connected layers.

	HFNet-V3	With pooling layer	With FC* layer
Classification accuracy (%)	71.3	71.6	71.5
Number of parameters (M)	6.46	6.46	7.51
Storage for parameters (MB)	24.63	24.63	28.63
Number of cores (1,024 × 1,024)	4,720	10,940	4,721

**Fully connected.*

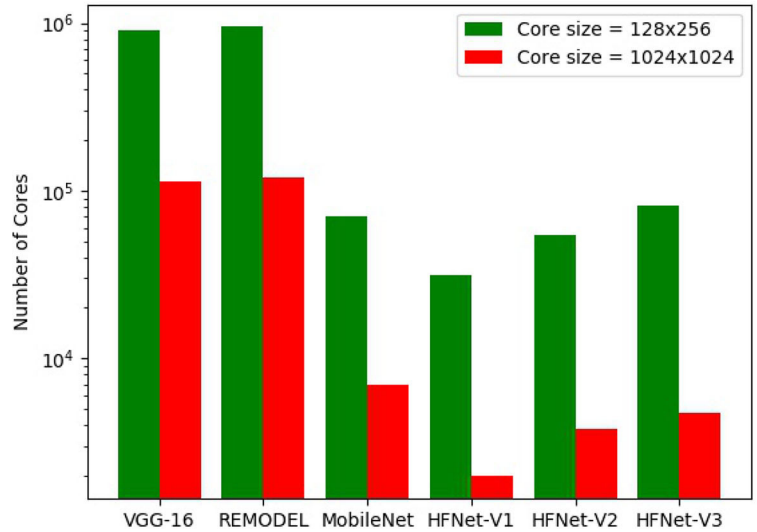
as VGG-16 while 2.2% more accurate than the MobileNet. It also utilizes less number of cores than the MobileNet. All HFNet models considered in this experiment is illustrated in **Supplementary Material**.

3.2.3. Augmenting the HFNet

This experiment adds more hardware-friendly layers to HFNet-V3 to investigate the increase in classification accuracies. We report the accuracy results for the adding depthwise separable convolution layer to end of HFNet-V3 (HFNet-V3-M0), one or two standard convolution layers to front of HFNet-V3 (HFNet-V3-M1 and HFNet-V3-M3, respectively) and both these two layers (HFNet-V3-M2). **Figure 12** illustrates these additions. **Table 3** shows the corresponding results. As expected, the additional layers lead to improved accuracies, with a standard convolution layer having a larger impact than depthwise separable convolution, while having less total number of parameters. This further validates the insight from VGGNet: the retaining of larger feature maps at shallow layers improves accuracy. It can be seen that adding depthwise separable convolution layers has a larger increase in number of parameters compared to standard convolution in shallow layers but has a smaller increase in number of cores. Adding two standard convolutions does not lead to better accuracy compared to one standard and one depthwise separable convolution. Best accuracy is obtained for HFNet-V3-M2 among all variants of the HFNet. All CNNs considered are illustrated in **Supplementary Material**.

3.2.4. Comparison With Modified MobileNets

In this experiment, we compare the HFNets with modified MobileNets that are more hardware friendly. The MobilNet is chosen as it also uses the hardware-friendly depthwise separable



	VGG-16	*REMODEL*	*MobileNet*	*HFNet-V1*	*HFNet-V2*	*HFNet-V3*
Classification Accuracy (%)	71.3	-	69.1	62.5	67.8	71.3
Number of parameters (M)	138.4	138.4	4.24	1.73	3.23	6.46
Storage for parameters (MB)	528	528	16.14	6.58	12.31	24.63
Number of cores (1024×1024)	113968	119172	6964	1978	3814	4720

Note:Parameters of VGG16 is taken from https://keras.io/applications/
MobileNet accuracy is 70.4% in Keras.

FIGURE 11 | Comparing different CNNs: the bar graph shows the number of cores required for mapping different CNNs. The table shows the classification accuracy on IMAGENET for the CNNs. Classification accuracy of HFNet-V3 is close to VGG-16 and MobileNet, while requiring a smaller number of cores.

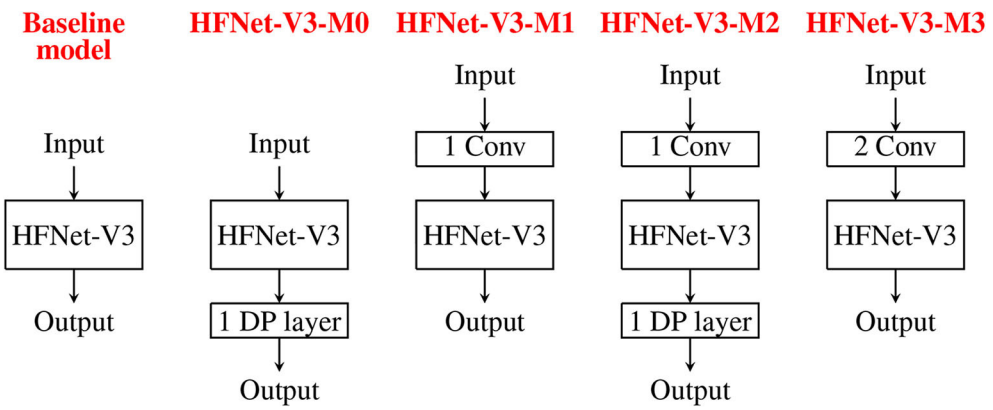


FIGURE 12 | Illustration of adding different convolutions to the baseline model, HFNet-V3: HFNet-V3-M0 has one more depthwise separable convolution. HFNet-V3-M1 has one more standard convolution. HFNet-V3-M2 has one more standard and one more depthwise separable convolution. HFNet-V3-M3 has two more standard convolutions.

convolution which also uses less parameters. The modified MobileNets are HF-MobileNet-V1 and HF-MobileNet-V2, such that the number of parameters are close to HFNet-V3-M1 and HFNet-V3, respectively. We also compare MobileNet with modified version of HFNet-V2, HFNet-V2-M0, by increasing parameters of HFNet-V2 to be close to those of MobileNet. From **Table 4**, the performance of HF-MobileNet-V2 (71.4%) compared to HFNet-V3 (71.3%) is better by only 1% while

TABLE 3 | Addition of layers to HFNet-V3.

	1 DP* layer HFNet-V3-M0	1 Conv layer HFNet-V3-M1	1 DP + 1 Conv layer HFNet-V3-M2	2 Conv layer HFNet-V3-M3
Classification accuracy (%)	71.5	72.2	72.7	72.6
Number of parameters (M)	7.52	6.64	7.7	6.75
Storage for parameters (MB)	28.68	25.3	29.35	25.73
Number of cores (1,024 × 1,024)	4855	6960	6948	11664

**Depthwise separable convolution.*

TABLE 4 | Comparison of HFNets with hardware-friendly MobileNets.

	HFNet-V3-M1	HF-MobileNet-V1	HFNet-V3	HF-MobileNet-V2	HFNet-V2-M0	MobileNet
Classification accuracy (%)	72.2	71.9	71.3	71.4	70	69.1
Number of parameters (M)	6.64	6.62	6.46	6.42	4.21	4.24
Storage for parameters (MB)	25.3	25.22	24.63	24.46	16.05	16.14
Number of cores (1,024 × 1,024)	6,960	7,642	4,720	7,529	3,949	6,964

MobileNet accuracy is 70.4% in Keras.

MobileNet (69.1%) compared to HFNet-V2-M0 (70%) is worse by 0.9% and HF-MobileNet-V1 (71.9%) compared to HFNet-V3-M1 (72.2%) is worse by 0.3%. Here, all the classification accuracies are trained using Tensorpack and Tensorflow framework. Note that MobileNet accuracy in Keras is 70.4%, 1.3% higher compared to Tensorpack result, but for a fair comparison we report accuracy results for CNN's trained using Tensorpack. Comparing MobileNets and HFNets with similar accuracy and parameter size, HFNet-V3 utilizes 2,809 less cores than MobileNet-V2, HFNet-V2-M0 utilizes 3,015 less cores than MobileNet and HFNet-V3-M1 utilizes 682 fewer cores than MobileNet-V1. The results show that the accuracies are close while there is a big difference in the number of cores utilized to make HFNet variants more hardware friendly. HFNets are therefore more neuromorphic hardware friendly than the hardware-friendly versions of existing deep learning architectures like MobileNet. The difference in number of cores for mapping is compared in **Supplementary Material** with respect to core utilization in each layer for HFNet-V3 and MobileNet.

3.2.5. Grouped Convolution

Here we replace depthwise separable convolution with grouped convolution (Esser et al., 2016): HFNet-GC. HFNet-GC is compared with HFNet-V3-M2, as it is modified from HFNet-V3-M2 by using approximately the same parameter size in each layer. To do so, we set the group convolutions to eight groups in each HFNet-GC layer. From **Table 5**, classification accuracy for HFNet-GC is around 13% less than the HFNet-V3-M2, while number of cores required almost doubled. HFNets with depthwise separable convolutions are

TABLE 5 | Comparison of HFNet with grouped convolutions (GC).

	HFNet-V3-M2	HFNet-GC
Classification accuracy (%)	72.7	59.8
Number of parameters (M)	7.7	6.91
Storage for parameters (MB)	29.35	26.38
Number of cores (1,024 × 1,024)	6,948	11,424

TABLE 6 | Number of cores for different core shapes.

Core shape	Number of cores
64 × 4,096	111,642
128 × 2,048	58,098
256 × 1,024	26,424
512 × 512	14,263
1,024 × 256	14,069
2,048 × 128	28,240
4,096 × 64	56,480

hence more hardware friendly than HFNets with grouped convolutions. The architecture of HFNet-GC is illustrated in the **Supplementary Material**.

4. DISCUSSION AND CONCLUSION

In our work, we first study what convolutions are more hardware friendly and how to best map them onto a neuromorphic hardware. We next identify deep learning techniques to avoid which result in poor core utilization or even result in core matrix splitting. We then propose a framework for the design of more hardware-friendly CNNs, and implement it using a Python wrapper (MaD). As a result of the above, the HFNet is proposed. Different versions of the HFNet are also proposed using the framework, that have better classification accuracy with more cores used when mapped. The framework thus allows us to propose different CNNs in a more principled manner by changing design parameters. We then evaluate it by comparison with other CNNs in terms of classification accuracy on IMAGENET, number of parameters, and cores required for mapping. It is able to achieve very comparable accuracy, using about the same number of parameters as per other more hardware-friendly CNNs but with substantially fewer mapped cores. Here, we have shown results on one of the biggest and most popular datasets in visual classification, IMAGENET, our results are quite generally applicable for visual tasks which is already covering a very big application space. Second, it has been shown that networks pre-trained on IMAGENET can be used as feature extractors for spectrograms for audio analysis (Acharya and Basu, 2020); our method can thus potentially generalize to other types of datasets as well. We have also proposed an optimized mapping technique by considering a square shaped selection of neurons. As shown in **Table 6**, we further explore how different core shapes ($64 \times 4{,}096$, $128 \times 2{,}048$, $256 \times 1{,}024$, 512×512, $1{,}024 \times 256$, $2{,}048 \times 128$, $4{,}096 \times 64$) affect the number of cores used to map the HFNet-V3. Overall, the trend in cores used agree with the intuition provided in the proof (section 2.2.1), which is based on real numbers; while core sizes are based on integers which may lead to some discrepancy. We further note that cores required is not symmetric about 512×512, with larger input dimensions using less cores (e.g., $4{,}096 \times 64$ against $64 \times 4{,}096$). This is due to avoiding core matrix split while mapping.

The HFNet is a hardware-friendly CNN that is designed using an iterative process that takes into account how best it can be mapped on a neuromorphic hardware with crossbar array of synapses while achieving good accuracy. One hard constraint while mapping is avoiding core matrix splitting. As shown in section 3, a typical HFNet requires thousands of cores for mapping. This is still larger than most neuromorphic chips. For future work, we will consider how one may map the HFNet onto a neuromorphic chip with limited number of cores. While we try to optimize synaptic resources and reduce "wastage" by minimizing unused synapses, it might be possible to reuse these synapses to provide a degree of fault tolerance in the hardware by providing redundancy. Current explorations of fault tolerance mostly show reduction in performance degradation after retraining a neural network with faults (Lee et al., 2014; Feinberg et al., 2018); however, there might be scope to optimize the fault tolerance by providing some extra synapses. We feel this is an important avenue of future work. We would also consider quantized CNNs, both weight and activations, in future work, which is beyond the scope of current work. Other hardware constraints, such as synaptic noise in novel devices, will be considered in future work.

Considering chip area, it maybe that only one physical neuron is implemented per core. This neuron is utilized in a time multiplexed manner to emulate all neurons within the core. If, however, there is more than one physical neuron per core, one can speed up computation by pipelining the time multiplexed neurons. Further speed-up can be achieved if the number of fan-in neurons per convolution operation is considered while increasing the physical neurons.

In our current study, we only studied CNNs without skip connections. Residual networks (He et al., 2016) have skip connections that increase the fan-in/fan-out degree of neurons. Not only that, to map them, we would either have to save intermediate results of skip-connections at routers or in buffers at the axons. This would be interesting to consider in future work.

DATA AVAILABILITY STATEMENT

All datasets generated for this study are included in the article/**Supplementary Material**.

AUTHOR CONTRIBUTIONS

RG and YC designed the manuscript outline and experimental framework. RG wrote the manuscript, designed and implemented the MaD python wrapper, and conducted the experiments. YC edited the manuscript and conducted the experiments. PS also conducted the experiments. AS contributed to the implementation of the MaD python wrapper and contributed to the figures in the manuscript. AB was involved in the discussion, experiment design, and editing of the manuscript. All authors contributed to the article and approved the submitted version.

FUNDING

This research was supported by Programmatic grant no. A1687b0033 from the Singapore Governments Research, Innovation and Enterprise 2020 plan (Advanced Manufacturing and Engineering domain).

ACKNOWLEDGMENTS

Short versions of this manuscript has been released as a Pre-print at Gopalakrishnan et al. (2019a,b).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnins.2020.00907/full#supplementary-material

REFERENCES

Acharya, J., and Basu, A. (2020). Deep neural network for respiratory sound classification in wearable devices enabled by transfer learning. *IEEE Trans. Biomed. Circuits Syst.* 14, 535–544. doi: 10.1109/TBCAS.2020.2981172

Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., et al. (2015). Truenorth: design and tool flow of a 65 MW 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34, 1537–1557. doi: 10.1109/TCAD.2015.2474396

Alyamkin, S., Ardi, M., Brighton, A., Berg, A. C., Chen, B., Chen, Y., et al. (2019). Low-power computer vision: status, challenges, opportunities. *IEEE J. Emerg. Select. Top. Circuits Syst.* 9, 411–421. doi: 10.1109/JETCAS.2019.2911899

Ambrogio, S., Balatti, S., Cubeta, A., Calderoni, A., Ramaswamy, N., and Ielmini, D. (2014a). Statistical fluctuations in HfOx resistive-switching memory: part I–set/reset variability. *IEEE Trans. Electron Devices* 61, 2912–2919. doi: 10.1109/TED.2014.2330200

Ambrogio, S., Balatti, S., Cubeta, A., Calderoni, A., Ramaswamy, N., and Ielmini, D. (2014b). Statistical fluctuations in HfOx resistive-switching memory: part II–random telegraph noise. *IEEE Trans. Electron Devices* 61, 2920–2927. doi: 10.1109/TED.2014.2330202

Amir, A., Datta, P., Risk, W. P., Cassidy, A. S., Kusnitz, J. A., Esser, S. K., et al. (2013). "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Dallas, TX), 1–10. doi: 10.1109/IJCNN.2013.6707078

Andrew, P. D., Daniel, B., Jochen, E., Jens, K., Eilif, M., Dejan, P., et al. (2009). PyNN: a common interface for neuronal network simulators. *Front. Neuroinformatics* 2:11. doi: 10.3389/conf.neuro.11.2008.01.046

Appuswamy, R., Nayak, T. K., Arthur, J. V., Esser, S. K., Merolla, P., McKinstry, J. L., et al. (2016). Structured convolution matrices for energy-efficient deep learning. *arXiv* 1606.02407.

Barry, B., Brick, C., Connor, F., Donohoe, D., Moloney, D., Richmond, R., et al. (2015). Always-on vision processing unit for mobile applications. *IEEE Micro.* 35, 56–66. doi: 10.1109/MM.2015.10

Basu, A., Acharya, J., Karnik, T., Liu, H., Li, H., Seo, J. S., et al. (2018). Low-power, adaptive neuromorphic systems: recent progress and future directions. *IEEE J. Emerg. Select. Top. Circuits Syst.* 8, 6–27. doi: 10.1109/JETCAS.2018.2816339

Bose, S., Acharya, J., and Basu, A. (2019). "Is my neural network neuromorphic? Taxonomy, recent trends and future directions in neuromorphic engineering," in *ASILOMAR Conference on Signals and Systems* (Pacific Grove, CA). doi: 10.1109/IEEECONF44664.2019.9048891

Davies, M., Srinivasa, N., Lin, T., 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

Deng, B. L., Li, G., Han, S., Shi, L., and Xie, Y. (2020). Model compression and hardware acceleration for neural networks: a comprehensive survey. *Proc. IEEE* 108, 485–532. doi: 10.1109/JPROC.2020.2976475

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). "ImageNet: a large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition* (Miami, FL), 248–255. doi: 10.1109/CVPR.2009.5206848

Deng, L., Zou, Z., Ma, X., Liang, L., Wang, G., Hu, X., et al. (2018). Fast object tracking on a many-core neural network chip. *Front. Neurosci.* 12:841. doi: 10.3389/fnins.2018.00841

Diehl, P., and Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Front. Comput. Neurosci.* 9:99. doi: 10.3389/fncom.2015.00099

Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R., Andreopoulos, A., et al. (2016). Convolutional networks for fast, energy-efficient neuromorphic computing. *Proc. Natl. Acad. Sci. U.S.A.* 113, 11441–11446. doi: 10.1073/pnas.1604850113

Feinberg, B., Wang, S., and Ipek, E. (2018). "Making memristive neural network accelerators reliable," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (Vienna), 52–65. doi: 10.1109/HPCA.2018.00015

Gopalakrishnan, R., Chua, Y., and Kumar, A. J. S. (2019a). Hardware-friendly neural network architecture for neuromorphic computing. *arXiv* arXiv:1906.08853.

Gopalakrishnan, R., Kumar, A. J. S., and Chua, Y. (2019b). MaD: mapping and debugging framework for implementing deep neural network onto a neuromorphic chip with crossbar array of synapses. *arXiv* arXiv:1901.00128.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas, NV), 770–778. doi: 10.1109/CVPR.2016.90

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv* 1704.04861.

Hu, M., Strachan, J. P., Li, Z., Grafals, E. M., Davila, N., Graves, C., et al. (2016). "Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)* (Austin, TX), 1–6. doi: 10.1145/2897937.2898010

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50× fewer parameters and <0.5 MB model size. *arXiv* 1602.07360.

Ji, Y., Zhang, Y., Chen, W., and Xie, Y. (2018). "Bridge the gap between neural networks and neuromorphic hardware with a neural network compiler," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems* (Williamsburg, VA: ACM), 448–460. doi: 10.1145/3173162.3173205

Ji, Y., Zhang, Y., Li, S., Chi, P., Jiang, C., Qu, P., et al. (2016). "Neutrams: neural network transformation and co-design under neuromorphic hardware constraints," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (Taipei), 1–13. doi: 10.1109/MICRO.2016.7783724

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., et al. (2017). "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 17* (New York, NY: Association for Computing Machinery), 1–12. doi: 10.1145/3079856.3080246

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images.*

Lee, M., Hwang, K., and Sung, W. (2014). "Fault tolerance analysis of digital feed-forward deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Florence), 5031–5035. doi: 10.1109/ICASSP.2014.6854560

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR* abs/1312.4400.

Pei, J., Deng, L., Song, S., Zhao, M., Zhang, Y., Wu, S., et al. (2019). Towards artificial general intelligence with hybrid tianjic chip architecture. *Nature* 572, 106–111. doi: 10.1038/s41586-019-1424-8

Prezioso, M., Merrikh-Bayat, F., Hoskins, B. D., Adam, G. C., Likharev, K. K., and Strukov, D. B. (2015). Training and operation of an integrated neuromorphic network based on metal-oxide memristors. *Nature* 521, 61–64. doi: 10.1038/nature14441

Roy, K., Jaiswal, A., and Panda, P. (2019). Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 607–617. doi: 10.1038/s41586-019-1677-2

Rueckauer, B., Lungu, I.-A., Hu, Y., and Pfeiffer, M. (2016). "Theory and tools for the conversion of analog to spiking convolutional neural networks," in *Workshop "Computing with Spikes", 29th Conference on Neural Information Processing Systems (NIPS 2016)* (Barcelona).

Shukla, R., Lipasti, M., Van Essen, B., Moody, A., and Maruyama, N. (2019). Remodel: rethinking deep cnn models to detect and count on a neurosynaptic system. *Front. Neurosci.* 13:4. doi: 10.3389/fnins.2019.00004

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv* 1409.1556.

Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for simplicity: the all convolutional net. *CoRR* abs/1412.6806.

Voelker, A. R., Benjamin, B. V., Stewart, T. C., Boahen, K., and Eliasmith, C. (2017). "Extending the neural engineering framework for nonideal silicon synapses," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)* (Baltimore, MD), 1–4. doi: 10.1109/ISCAS.2017.8050810

Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., and Tan, K. C. (2019). A tandem learning rule for efficient and rapid inference on deep spiking neural networks. *arXiv* 1907.01167.

Wu, Y. (2016). *Tensorpack*. Available online at: https://github.com/tensorpack/

Yakopcic, C., Alom, M. Z., and Taha, T. M. (2016). "Memristor crossbar deep network implementation based on a convolutional

neural network," in *2016 International Joint Conference on Neural Networks (IJCNN)* (Vancouver, BC), 963–970. doi: 10.1109/IJCNN.2016.7727302

Yakopcic, C., Alom, M. Z., and Taha, T. M. (2017). "Extremely parallel memristor crossbar architecture for convolutional neural network implementation," in *2017 International Joint Conference on Neural Networks (IJCNN)* (Anchorage, AK), 1696–1703. doi: 10.1109/IJCNN.2017.7966055

Zhang, X., Zhou, M. L., and Sun, J. (2018). "Shufflenet: an extremely efficient convolutional neural network for mobile devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT), 6848–6856. doi: 10.1109/CVPR.2018.00716

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