Technische Universität München
TUM School of Computation, Information and Technology

Sparse Spiking Neural Networks for Radar Signal Processing

Javier López Randulfe

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Felix Dietrich

Prüfende der Dissertation:

1. Prof. Dr.-Ing. habil. Alois C. Knoll
2. Prof. Stephen B. Furber, Ph.D.

Die Dissertation wurde am 04.06.2024 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 23.01.2025 angenommen.

Acknowledgement

I want to thank all my colleagues at the chair, as well as the administrative staff, Alex, Janine, Amy, and Ute, for their kindness and invaluable help, and Professor Knoll for his supervision. I especially want to thank my KI-ASIC project teammates, Tobi, Robin, Nico, and Negin, for creating a fun and enjoyable work environment all these years. I also greatly appreciate the scientific and endless discussions with Tobi and Nico, which eventually led to this thesis's main contributions. I also want to thank my mentor Leon Larsen for his support at the start of my research, his guidance was extremely valuable for taking my first steps and finding my path through. Thanks also to Katrine, Josué, Nico, and Oliver for the feedback and tips on the dissertation.

Above all else, I want to thank all the friends and family who have supported me all this time. I would need several pages to describe how each of you has helped me, and even then, I would be missing words.

¡Os lo agradezco de todo corazón!

Munich, May 10, 2024 *Javier López Randulfe*

Abstract

Neuromorphic computing is an emerging field that takes inspiration from nature to craft fast and efficient systems. Contrary to traditional computing, where data is processed in frames sent periodically to the algorithms, neuromorphic systems are event-based, meaning that data is represented by events over time that are processed asynchronously. Spiking neural networks are a typical example of neuromorphic algorithms. In these algorithms, computing is distributed among multiple nodes called neurons and data is propagated as binary spikes. These spiking neural networks are accelerated in dedicated neuromorphic chips to achieve the promised low power and latency. Depending on their scope of application, these chips range from tiny edge-computing devices to large data centres. They can be based on digital processors, analogue electronics, or a mix of both. Neuromorphic hardware and spiking neural networks have grabbed the attention of the automotive industry thanks to their efficient operation. While research is already devising neuromorphic systems for sensors like LiDAR or event-based cameras for perception tasks, event-based processing is still unexplored for automotive radar applications. *Frequency-modulated continuous-wave* (FMCW) radar is a low-power sensor crucial for driving tasks like adaptive cruise control or collision avoidance. They provide a direct measurement of the range and velocity of objects and excel under adverse weather conditions. The temporal and sparse nature of FMCW radar data makes this sensor an ideal candidate for being processed using neuromorphic systems.

This work explores, for the first time, the application of neuromorphic algorithms for directly processing raw data from FMCW radar sensors. The main contribution is the design of a complete signal processing pipeline starting from the radar signal to generate the frequency spectrum. An *analogue-to-spike encoder* (ASE) circuit encodes the voltages into spikes without requiring an analogue-to-digital converter, and a spiking neural network replicates the functionality of the Fourier transform, generating the frequency spectrum with the spikes of the ASE. The thesis also includes implementing and testing a spiking neural network that performs the object-detection task on the output of the Fourier transform. High sparsity, i.e., low spike density, is paramount for making neuromorphic applications efficient. Contrary to rate-based neuromorphic algorithms with high spike density, the ASE and the spiking algorithms encode information in the precise timing of spikes. Namely, each data sample is represented by no more than one spike, resulting in highly sparse algorithms. While the ASE was implemented as a prototype circuit, the spiking neural networks were implemented in the digital neuromorphic chips Loihi 2 and SpiNNaker 2. The experiments tested the ASE using synthetic electric signals obtained in a laboratory and the spiking neural networks with simulated and actual sensor data from automotive scenarios. Experiments also included processing the resulting spikes from the ASE with the spiking Fourier transform. Results show that the neuromorphic pipeline maintains the accuracy of well-established methods like analogue-to-digital conversion or the Fourier transform and opens the door to accelerating end-to-end signal processing pipelines employing neuromorphic systems.

Zusammenfassung

Neuromorphic Computing ist ein aufstrebender Bereich, der von der Natur inspiriert ist, um schnelle und effiziente Systeme zu entwickeln. Im Gegensatz zur traditionellen Datenverarbeitung, bei der die Daten getaktet in Form von Blöcken an die Algorithmen gesendet und verarbeitet werden, sind neuromorphe Systeme ereignisbasiert, d. h., die Daten werden durch zeitliche Ereignisse dargestellt, die asynchron verarbeitet werden. Gepulste neuronale Netze sind ein typisches Beispiel für neuromorphe Algorithmen. Bei diesen Algorithmen ist die Datenverarbeitung auf mehrere Knoten, Neuronen genannt, verteilt und die Daten werden als binäre Pulse übertragen. Diese gepulste neuronale Netze werden auf speziellen neuromorphen Chips beschleunigt, um den versprochenen geringen Stromverbrauch und die geringe Latenzzeit zu erreichen. Je nach Anwendungsbereich findet man diese Chips in kleinen Edge-Computing-Geräten bis hin zu großen Datenzentren. Sie können auf digitalen Prozessoren, analoger Elektronik oder einer Mischung aus beidem beruhen. Neuromorphe Hardware und gepulste neuronale Netze haben aufgrund ihrer effizienten Funktionsweise die Aufmerksamkeit der Automobilindustrie auf sich gezogen. Während die Forschung bereits an neuromorphe Systeme für Sensoren wie LiDAR oder ereignisbasierte Bildsensoren für Wahrnehmungsaufgaben arbeitet, ist die ereignisbasierte Verarbeitung für automobile-Radaranwendungen noch weitestgehend unerforscht. Das *frequency-modulated continuous-wave* (FMCW)-Radar ist ein stromsparender Sensor, der für Fahraufgaben wie die Abstandsregeltempomat oder die Kollisionsvermeidung von entscheidender Bedeutung ist. Sie ermöglichen eine direkte Messung der Entfernung und Geschwindigkeit von Objekten und eignen sich hervorragend für den Einsatz unter ungünstigen Wetterbedingungen. Die zeitliche und spärliche Natur von FMCW-Radardaten macht sie zu einem idealen Kandidaten für die Verarbeitung mittels neuromorpher Systeme.

In dieser Arbeit wird zum ersten Mal die Anwendung neuromorphen Algorithmen für die direkte Verarbeitung von Rohdaten von FMCW-Radarsensoren untersucht. Der Hauptbeitrag ist der Entwurf einer vollständigen Signalverarbeitungspipeline von Radarsignale zur Erzeugung des Frequenzspektrum. Eine *analogue-to-spike encoder* (ASE)-Schaltung kodiert die Spannungen in Pulse, ohne dass ein Analog-Digital-Wandler erforderlich ist. Ein gepulstes neuronales Netz bildet die Funktionalität der Fourier-Transformation, die Erzeugung des Frequenzspektrums, mit den Pulsen des ASE ab. Die Arbeit umfasst auch die Implementierung und den Test eines gepulsten neuronalen Netzes für die Objekterkennung im Anschluss an die Fourier-Transformation. Eine hohe Sparsamkeit, d. h. eine niedrige Pulse-Dichte, ist für die Entwicklung effizienter neuromorpher Anwendungen von zentraler Bedeutung. Im Gegensatz zu ratenbasierten neuromorphen Algorithmen mit hoher Pulse-Dichte kodieren der ASE und die gepulste Algorithmen Informationen im präzisen Zeiten der Pulse. Jedes Datenmuster wird durch nicht mehr als einen Puls dargestellt. Jeder Datenpunkt wird durch nicht mehr als einen Puls in der vorgestellten neuromorphen Datenverarbeitungskette dargestellt, was zu sehr spärlichen Algorithmen führt. Während der ASE als Prototyp-Schaltung implementiert wurde, wurden die gepulste neuronalen Netze in den digitalen neuromorphen Chips Loihi 2 und SpiNNaker 2 implementiert. In den Experimenten wurde der ASE mit synthetischen elektrischen Signalen getestet, die in einem Labor erzeugt wurden, und die gepulste neuronale Netze mit simulierten und tatsächlichen Sensordaten aus Automobilszenarien. Darüber hinaus wurde auch die Verarbeitung der aus dem ASE resultierenden Pulse mit der gepulste Fourier Transformation evaluiert. Die Ergebnisse zeigen, dass die neuromorphe Pipeline die Genauigkeit etablierter Methoden wie der Analog-Digital-Wandlung oder der Fourier-Transformation beibehält und die Tür zur Beschleunigung von End-to-End Signalverarbeitungspipelines unter Verwendung neuromorpher Systeme öffnet.

Contents

List of abbreviations

ADAS advanced driver-assistance system

ADC analog-to-digital converter

AI artificial intelligence

ANN artificial neural network

AoA angle of arrival

AP action potential

ASE analogue-to-spike encoder

CA-CFAR cell-averaging CFAR

CFAR constant false alarm rate

DBSCAN density-based spatial clustering of applications with noise

DCNN deep convolutional neural network

DFT discrete Fourier transform

FFT fast Fourier transform

FMCW frequency-modulated continuous-wave

FS few spikes

FT Fourier transform

HH Hodgkin-Huxley

IF intermediate frequency

ISI inter-spike interval

LIF leaky integrate-and-fire

MIMO multiple-input multiple-output

OS-CFAR ordered-statistics CFAR

PE phase encoding

RF resonate-and-fire

RMSE root-mean-square error

ROC rank-order code

S-FT spiking Fourier transform

S-OSCFAR spiking OSCFAR

SNN spiking neural network

STDP synaptic timing-dependent plasticity

STFT short-time Fourier transform

TC temporal contrast

TCBS temporal charge before spike

TTFS time to first spike

1

Introduction

Humanity has sought to understand how the brain works since the early stages of history and has been attempting to apply the knowledge obtained to create intelligent machines. At the end of the XIX century, the pace of research sharply accelerated. In the natural sciences, the discoveries fostered by Camilo Golgi and Santiago Ramon y Cajal marked the start of modern neuroscience and the neuron doctrine. They led to our vast knowledge of how brains acquire, process, and transmit information [Ram04]. In formal sciences, a group of mathematicians led by Alan Turing took the first steps in defining the theory of what today we know as *artificial intelligence* (AI) [Tur50]. The following decades brought a continuous flow of mathematical models and algorithms that increasingly narrowed the gap between the computing capabilities of machines and biology [Sch22]. The development of *artificial neural networks* (ANNs) has particularly shown remarkable growth, from the early steps with the McCulloch-Pitts neuron model, the perceptron model, the backpropagation algorithm or the application of the chain rule on artificial neural networks; to the modern algorithms like the long short-term memory [HS97] or self-attention transformers [Vas+17]. This progress has resulted in the outstanding performance of today's *deep convolutional neural networks* (DCNNs) for the processing of images [KSH17], or large language models (LLMs) for processing natural language on a similar level as humans [Rad+18].

All these advances would not have been possible without the fast and steady growth in the capabilities of computing systems, i.e., the exponential increase in the performance of processing hardware has been a critical factor for the sharp progress in computer science and AI. This partially results from what is known as Moore's law, which predicts a doubling in the amount of transistors on integrated circuits every two years since the 1970s. Moreover, recent breakthroughs in AI are a consequence of the appearance of new computing architectures that use highly parallel structures for processing data, graphics processing units being the most popular of such devices. Despite the considerable improvements in the accuracy and processing speed, AI systems are very limited for edge computing applications due to their high energy and memory requirements. One of the most illustrative examples is the significant limitation in achieving the long-standing dream of fully autonomous vehicles [Lin+18]. This limitation is a showcase of the still enormous gap between technology and biology in terms of resource optimization, as the human brain has an approximate consumption of $20W$, in contrast to supercomputers, which require more than $40,000$ times more power for performing similar computations [Tha+18].

Inspired by the low energy consumption and the remarkable performance of biological neural circuits, neuromorphic engineering has tried replicating them for decades [Mea90]. Researchers in this field have created novel hardware architectures based on the principles of

neural systems and offer an alternative to the Von Neumann paradigm. These architectures are based on an asynchronous grid of independent computing units with a local memory that emulate neurons' role in the brain. Their communication is based on binary events over time called spikes [Mea20]. Although there is an open debate about the requirements a chip must meet to be considered neuromorphic, many neuromorphic chips that follow different motivations and computing paradigms are available today. We can find large-scale chips like the human brain project SpiNNaker, whose initial goal was to simulate brain-sized neural networks while staying flexible and adapting to different properties and specifications [Fur+14]; as well as chips oriented at embedded, niche applications that only require a small number of neurons [Mor+17]; or chips like Intel's Loihi that target industrial applications by improving the latency and energy specifications of current solutions [Dav+18]. Regarding the working principle, neuromorphic systems were initially designed for implementing highly efficient electronic neural circuits by taking advantage of the sub-threshold properties of CMOS transistors [Ind21; DMM95]. Newer generations of neuromorphic systems are more varied and include digital chips that perform arithmetic operations like traditional computers do [Fur+14; Dav+18]. One last group of chips implements a hybrid approach, combining the sub-threshold regime for neuronal computations and digital electronics to communicate between the different neural nodes [Sch+17].

After the initial breakthroughs in neuromorphic hardware architectures and communication protocols, a new research subfield called *neuromorphic computing* emerged to fulfil the need for algorithms that could leverage the benefits of neuromorphic hardware [Ind21]. This is the purpose of a new generation of ANNs named *spiking neural networks* (SNNs) [Maa97]. SNNs are formed by parallel computing nodes that process incoming events asynchronously, store information locally, and react by generating new events or spikes. The main aim of neuromorphic computing research is to reduce the gap between Engineering and biological systems and create fast and efficient bio-inspired algorithms [Tha+18]. On the one hand, it replicates and validates spiking neural models that mimic the behaviour of biological neurons that help understand how the brain works. On the other hand, it adapts those models so they can solve Engineering problems with higher accuracy, lower energy consumption, and faster reaction times than current state-of-the-art algorithms. Research on SNNs includes diverse tasks such as designing network architectures, information encoding schemes in spikes, dynamic models for the neurons, or learning models that adapt the network properties over time to optimize them for specific tasks. Therefore, neuromorphic computing is highly interdisciplinary, influenced by diverse fields like computational neuroscience, cognition, electrical engineering, computer science, and control theory.

Neuromorphic systems hold significant promise in advancing sensor signal processing research. Research is increasingly integrating them into signal processing pipelines spanning diverse sensor types, offering the potential to notably reduce energy consumption in pipelines with tight energy requirements [Sch+22]. Event-based processing and information encoding into asynchronous event streams make neuromorphic systems well-suited for processing time-varying inputs that carry highly time-correlated information. Hence, neuromorphic computing emerges as a promising tool for signal processing in embedded systems, with applications for *advanced driver-assistance systems* (ADASs) standing out due to their formidable challenges. ADASs' operation within dynamically changing environments, resource constraints, dense data streams, real-time processing demands, and the necessity for high accuracy underscores the crucial role of signal processing. With LiDAR and vision cameras, radar sensors are critical in providing data necessary for the perception of the environment to ensure safe and efficient driving. Radar sensors often deal with vast amounts of real-time data and require efficient processing techniques to extract sparse information from noisy backgrounds. Neuromorphic computing offers the potential to address these chal-

lenges by providing energy-efficient, parallel processing capabilities suited for handling the continuous data stream.

The goal of this work was to explore the application of sparse SNNs for radar signal processing and design novel time-coded models for performing traditional radar signal processing tasks starting at the raw analogue data provided by the sensor's circuit. Research mainly dealt with the data encoding techniques and the spiking neuron models, with a minor focus on network architectures and learning approaches for SNNs. The replaced signal processing tasks were (i) the analogue-to-digital conversion of sensor signals, (ii) the frequency-spectrum representation, (iii) object detection, and (iv) object classification, with a special focus on tasks (i) and (ii). The designed SNNs for tasks (ii) and (iii) were implemented on digital neuromorphic hardware, namely the SpiNNaker 2 chip and Intel's Loihi, and the experiments were run on real-world data. The neuron model designed for (i) was implemented on an ad-hoc electric circuit and tested on artificial data generated at a lab. Experiments also included merging tasks (i) to (iii) for creating a spiking signal processing chain for automotive radar data, starting from the analogue signal until obtaining a target point cloud.

1.1 Background and motivation

While neuromorphic computing is still in the early stages of development and faces challenges such as limited hardware and software availability, it holds significant promise for impacting various fields, particularly those that require energy efficiency, fast processing, and adaptability. Over the years, the scope has broadened from the initial research replicating the neuronal structures for sensing the environment [Mea20], to the multiple potential applications of this technology nowadays [Sch+22]. In 2022, AI, together with data centres and cryptocurrencies, consumed around 2% of the global power demand, and the tendency is to double in 2026 [IEA24]. With this motivation, some neuromorphic systems aim to be the basis for new data centres for energy-efficient AI, Data Analytics, and Process Optimization, thanks to their lower energy consumption and sustainable approach [Vog+24]. Neuromorphic computing is also well-suited for cognitive computing research, as it can simulate how human brains process information and validate biological models [Rho+20]. Furthermore, Neuromorphic processors are promising for edge-computing applications with energy constraints by leveraging neuromorphic computers' low-power operation and event-driven nature for real-time processing and play a critical role in developing autonomous systems, including robotics [Bin+18], drones [Sta+20], and self-driving cars [Via+21], where neuromorphic systems collect sensor data and replace traditional signal processing algorithms by event-based SNNs [Zhe+23; Wan+20]. The resulting systems consist of streamlined SNNs that perform tasks with increasing levels of abstraction, from the input data to meaningful symbols that define the objects in the sensor's surroundings. Spike encoding is crucial for making these pipelines efficient, as the number of spikes directly influences the application's footprint. The encoding strategies must be co-designed with the neuron models determining the SNNs dynamics, as the former defines the message the latter interprets. Rather than striving for direct mimicry of the human brain, biological principles inspire the design process, so the resulting models are tailored to meet the computational demands of real-world applications.

1.1.1 Efficient spike representation

Spike-based neuromorphic applications require processing and extracting meaningful information from spatio-temporal data represented as a series of spike trains. Neuromorphic solutions encode and process information efficiently using spikes, mimicking biological neural networks. As neuromorphic hardware architectures become more present, there is a need for efficient algorithms that can exploit the benefits of these chips. One of the most important factors for determining the performance of SNNs when they are implemented on neuromorphic hardware is the sparsity, which refers to the number of spikes used by the network, i.e., the lower spike activity in the network, the less energy it will consume. In the most popular digital neuromorphic chips, spike sparsity is also inversely proportional to the total latency of the system [Dav+18; Rho+18; DF21]. For a given population of spiking neurons, we can think of *temporal sparsity* and *spatial sparsity*. The former refers to the number of spikes that each neuron needs to emit to correctly transmit information, and the latter refers to how many neurons in the population are active for each input sequence. The sparsity of an SNN depends on design decisions like the model of the neuron dynamics, the architecture of the network, and the encoding technique used for mapping information into spikes. Sparsity inherently depends on the task the SNN is solving: Low-level tasks closer to sensor data generally take as input very dense data that contains information spread over large data frames together with redundant data and noise. In contrast, high-level tasks employ pre-processed data where essential information is concentrated and unimportant data and noise are filtered out. This is analogous to the brain: senses collect a high density of information processed sequentially by multiple neuron layers. Initial layers are very active, and subsequent layers are more sparse and react to precise patterns, e.g., many spikes from thalamic neurons, where sensory input is collected, are required to fire single neurons in the layers of the neocortex they are connected to [BP12; Rho+20].

Literature typically classifies spike encoding techniques into *rate encoding* or *time encoding* [Aug+21b], depending on whether the information is encoded in the rate of spikes or the precise timing of the spikes. Some sources also include a third type called *population encoding*, where information is stored in the number of spikes across a population of neurons, i.e., along the spatial dimension [DA01]. However, *population encoding* could also be understood as a subset of *rate encoding* [Aug+21b]. Even though there is an open debate in biology about which technique is used by neurons [Bre15], the supremacy of time encoding from a computational perspective is evident when analyzing parameters like the system latency and energy efficiency [Mea20; DF21]. Time-coded SNNs are more sparse; thus, less energy is required to transmit information across the network. Moreover, since communication requires fewer spikes, the throughput in the interneuron connections will be higher, and the latency for conveying information will be lower. The first steps of neuromorphic computing research mainly focused on rate-based algorithms, as their translation into floating-point operations is more straightforward to understand [Rue+17]. Their goal was to prove the feasibility of performing complex tasks with SNNs. However, research in the last years has shifted its focus towards time encoding methods [Chr+22].

1.1.2 Neuromorphic computing for sensor signal processing

Sensor is a term that refers to any device that responds in a predictive manner to a signal or stimulus. In the case of natural systems, sensors react to a signal by producing a specific flux of ions, whereas artificial sensors typically react by producing a flux of electrons. In [FF10], an *artificial sensor* is defined as *a device that receives a stimulus and responds with an electrical signal*. Based on this definition, the output of a sensor takes the form of voltage,

current, or electric charge. Besides, the output signal can carry information in properties like its polarity, frequency, and phase. Literature traditionally classifies sensors from several perspectives: Depending on the type of excitation (active or passive); the employed reference system (absolute or relative); the kind of stimulus they measure (acoustic, electric, chemical, magnetic, optical ...); their conversion phenomena (photoelectric, thermoelectric, magnetoelectric ...); or their field of application (agriculture, space, construction, domestic, automotive ...). The study of sensors is vast, and a comprehensive overview is out of the scope of this thesis. [FF10] offers a detailed overview of the topic for the interested reader.

Sensor signal processing is a field associated with Electrical Engineering and Computer Science disciplines. It encompasses all techniques used to modify the signals obtained from sensors to generate meaningful data. Sensor data processing typically involves long pipelines of chained processing stages in multiple computing platforms. The first stages occur in high-performance analogue circuits that efficiently process the data in real-time at high speeds. These components produce a signal compatible with the digital interface of the sensor through operations like amplification, filtering, level shifting, and impedance matching [FF10]. For sensors formed by more than one channel, a multiplexer usually follows the conditioning circuit to switch between channels periodically. Alternatively, when sensors are formed by an array of multiple antennas, the signals from the different channels contain redundant information that can be combined in a *beamforming* process. Finally, the output signal is digitised in an *analog-to-digital converter* (ADC) and sent to a digital processor that performs the following processing pipeline stages. Semantics become more relevant there, so the output data is more meaningful for human interpretation or for creating a final application.

One of the main challenges for modern embedded systems is implementing sophisticated solutions to solve complex problems while keeping a low power consumption. Hence, a growing amount of research is working toward using neuromorphic hardware for fast and efficient signal processing pipelines [Aim+22], and it is expected to gain momentum within the industry in the following years. A market analysis predicted that neuromorphic computing could represent up to 20% of the AI market share by 2035 [Int21]. The main features driving this interest are the sparsity, event-based processing, and suitability for processing data that changes over time. The use of SNNs is especially appealing for applications that are dominated by some of these features:

- Applications where data is parallel and sparse, like peak clustering, tracking, or inference of DCNNs.
- Applications highly dominated by temporal dynamics and event-based data, like the processing of event-based sensors.
- Applications that can operate directly on the raw voltage provided by the sensor, thus removing the necessity of an ADC.

In some cases, applying SNNs is beneficial only for specific processing stages. For this reason, some chip designs follow hybrid approaches where the data processing swaps between neuromorphic and traditional computing approaches throughout the processing flow [Yan+21]. The suitability of neuromorphic systems for signal processing also depends on the type of sensor at hand: Whereas some sensors produce dense, frame-based data that is hard to accelerate on neuromorphic hardware, others provide sparse events or time series that can be efficiently computed with SNNs. For this reason, research on neuromorphic computing for signal processing targets niche applications where the nature of the sensor and the demand for fast and efficient computation justifies its implementation. The perception task for mobile systems like mobile robots, drones, and ADASs is paradigmatic, as they possess all the aforementioned characteristics, i.e., the sensors onboard produce data that is sparse, redundant,

and highly correlated over time. Moreover, safety requirements demand fast reaction times and high accuracy for calculating the location of obstacles and other vehicles, and the limited energy available in the batteries requires computing approaches with constrained consumption, as this determines the maximum navigation time. In the car industry, the increasing push for more intelligent vehicles is bringing the sensing and computing systems onboard, composed of GPUs, ASICs, and FPGAs, to drain up to 10% of the available energy [Lin+18], reducing the available mileage. Perception typically takes place through LiDAR, radar, and vision sensors. Ultrasound sensors are used for tasks like parking assistance and obstacle detection at low speeds, where close-range detection is essential, so their processing with neuromorphic hardware is less attractive. A new type of sensing called event-based vision is getting more attention for replacing or complementing traditional vision cameras due to their robustness and fast response times, and researchers anticipate a wide use of this sensor for autonomous navigation [Che+20]. Table 1.1 compares the aforementioned sensors qualitatively.

Event-based vision sensors operate differently from traditional cameras by outputting asynchronous brightness changes (events) instead of fixed-size frames at a fixed rate. Event-based cameras provide advantages like low latency (microseconds vs. milliseconds), high dynamic range (140 dB vs. 60 dB), and no motion blur. Event-based sensors produce a sequence of events defined by the pixel location and the exact timing of the event. Data is transmitted using an address-event representation protocol to facilitate efficient communication to the element processing the events [IH11]. The first event-based sensor was introduced by Mead and Mahowald in 1988 [MM88], and since then, research with this sensing technology has been fruitful. Tobi Delbruck's team significantly contributed by developing the dynamic-vision sensor [LPD08]. Because they generate asynchronous, event-based data, they are highly compatible with neuromorphic hardware and algorithms. [Che+20] describes this sensor's working principle and processing alternatives in detail and reviews the most prominent algorithms and applications.

LiDAR, which stands for Light Detection and Ranging, is a sensing technology that has gained momentum in recent years thanks to its fast and accurate sensing of the surrounding environment [Beh+17]. The working principle of these sensors consists of the emission of coherent light waves with precise knowledge of their direction and timing and the recording of the arrival of the reflections generated when the light beams hit a target. The round-trip delay indicates the distance to the detected target. A beam-steering technique is necessary to orient the light towards all desired directions to scan the whole area and obtain a 3D representation of the environment. Alternatively, flash LiDAR sensors scan the whole scene simultaneously at the cost of reduced light intensity. Recent advances in neuromorphic computing have shown the potential of bio-inspired methods for processing LiDAR data [DA23]. Neuromorphic approaches can be applied to different stages of LiDAR processing chains, from the raw data formed by the photon arrival times to computing already processed point

Table 1.1: Qualitative comparison of some characteristics of the most relevant sensors for current and future automotive applications. Robustness refers to a combination of accuracy and lack of clutter.

Sensor	Price	Data density*	Robustness	Maturity
Vision	Low	High	Low	High
LiDAR	High	Mid	High	Mid
Radar	Low	Low	Mid	Mid
Event-based vision	High	Low	High	Low

* For LiDAR and radar, data density refers to the point cloud generated after processing the raw signal.

clouds. In [Wan+20], the authors process the temporal pulses obtained from LiDAR sensors with spiking neural networks to solve an object detection task. The authors in [Sha+20] introduce the design of a memristive architecture for processing LiDAR signals following an event-based fashion. The work in [Afs+20] directly encodes the photon time of flight for 3D imaging applications into spikes using an address-event representation, similar to event-based vision sensors. The resulting event-based method drastically reduces the data rate and improves the accuracy compared to frame-based alternatives.

Similar to LiDAR, radar technology is based on the emission of electromagnetic waves and detecting the reflections produced by obstacles in the vicinity [Pat+17]. The core difference resides in the wavelength of the emitted waves, i.e., whereas LiDAR sensors work in the infrared spectrum, radars lie in the radio spectrum. Most automotive radar sensors operate in the 76–81 GHz band and ongoing efforts aim to increase this value to over 100 GHz [WHM21]. Compared to LiDAR sensors, radars can detect obstacles hidden behind other objects; achieve long ranges, up to 250m; are more reliable under adverse weather conditions, like rain, fog, and darkness; and are more affordable. On the downside, they are generally less robust regarding detection accuracy and range resolution. Radar sensing and processing is a mature field, as it is more than 100 years old, even though its usage onboard vehicles started recently thanks to the improvement of the mm-wave technology [Pat+17]. We distinguish pulsed radar, frequency-modulated radar, or *frequency-modulated continuous-wave* (FMCW) radar depending on how the radar wave is modulated over time. The latter is the strategy used for automotive applications, as it allows for the simultaneous calculation of the position and speed of targets and is, thus, the primary sensor considered in this thesis. chapter 2 details the fundamentals of this type of radar sensor and the most common operations applied to its output. The most prominent usage to date of neuromorphic computing for FMCW radar has been gesture recognition. This is motivated by the ease of classifying hand gestures compared to automotive tasks and the availability of public datasets for this type of task. In [Ger+22], the authors proposed an SNN that learns gestures from binned radar range-Doppler and range-angle maps. They used the *leaky integrate-and-fire* (LIF) neuron model and learned the weights using surrogate gradients and backpropagation through time. The authors in [Stu+21] presented a digital, event-driven chip able to classify gestures draining only 60μW of power. To classify the gestures, they initially trained an ANN and converted the learnt weights to an SNN based on the LIF neuron model. The work in [Tsa+21] uses a liquid state machine in combination with a spike representation of input range-Doppler maps for classifying input hand gestures. All these works show the feasibility of using neuromorphic computing for processing signals from FMCW radars and motivate the design of SNNs that can perform more complex tasks such as frequency-spectrum representation, multi-object detection, or clustering, which could be accelerated in neuromorphic hardware for automotive applications.

Although audio sensors are unsuitable for perception tasks in mobility applications, they are also attractive for neuromorphic research in this area for several reasons. Firstly, they produce data with temporal patterns similar to radar and LiDAR, so transferring the designed neuromorphic algorithms is possible. Secondly, they pose a more straightforward challenge than the other sensors, as they work on slower time scales and their applications require lower precision. These sensors have already been the target of a research competition organized by Intel to push the development of neuromorphic signal processing solutions [Tim+23]. The winners proposed a method based on gated spiking neurons that decompose the audio spectrogram into frequency sub-bands, which they then filtered individually. In [Aug+21a], the authors use resonate-and-fire neurons as a first layer to filter the different frequency components in audio signals for later classification of speech commands using surrogate gradients to differentiate the neuron function. A specially interesting feature of

the neuron model they used is the fact that neurons take sensor voltages as input, so the SNN could be accelerated in hardware without the need of an ADC component. Other works directly classified speech commands using SNNs trained with supervised [Dom+18] and unsupervised [DHX18] learning methods.

1.1.3 Radar signal processing and compatibility with neuromorphic HW

Starting from the raw analogue signal produced by the sensor, a radar processing pipeline contains various operations computed sequentially. Regardless of the final application, radar pipelines typically consist of the following initial stages:

- *Analogue-to-digital-conversion*: This is a fundamental operation consisting of translating analogue values in the continuous domain to discrete, digital values. ADCs are the components carrying out this operation.
- *Frequency spectrum representation*: As radar waves carry information in the frequency components, mapping the signal to its frequency spectrum is crucial. Thanks to its robustness and computational efficiency, the *Fourier transform* (FT) is the most extended algorithm for performing the task.
- *Angle of arrival* (AoA) estimation: Sensors formed by an array of multiple antennas allow us to obtain the direction of a target by comparing the relative distance to each of the antennas. The techniques for performing this task range from analytic geometry processes based on triangulation to beamforming or computing the FT in the angular dimension.
- *Object detection*: This stage determines which radar reflections correspond to actual objects and discards all data considered background or noise. The output is a point cloud.
- *Clustering*: After obtaining a point cloud, a clustering algorithm groups generated points into blobs that belong to the same object.

After processing the raw signal, higher-level stages compute the output and provide meaningful information such as the objects' identity, location, and path-tracking.

The temporal nature of radar signals and the increasing demand for efficient and accurate systems in automotive applications make SNNs an attractive choice for processing the data from these sensors. However, research to date on applying event-based algorithms to radar data is limited to gesture recognition. Moreover, none of the SNNs designed for solving this task take as input the raw signals from the sensors, i.e. they learn the features and classify gestures using already processed data comprised of the frequency spectrum given by the FT or object point clouds. Previous work with other sensors shows that SNNs can directly process sensor raw data, e.g., EEG signals [Sha+21a] or audio signals [Aug+21a]. Such approaches promise energy gains while yielding high accuracy, inspiring the crafting of similar concepts for other sensors like, in this case, FMCW radar.

1.2 Goal and contribution

This work aimed to design neuromorphic computing strategies for processing FMCW radar signals, with the ultimate goal of obtaining end-to-end neuromorphic signal processing

pipelines that can operate in real-time and leverage the benefits of event-based computing. The design approach was modular, so the replacement of each processing stage was handled independently, following the task distribution shown in section 1.1.3. Nevertheless, the design of the individual algorithms had a global vision, and the implementation experiments analysed the interoperability of the algorithms. The compatibility between the processing elements required special care when choosing spike encoding schemes, i.e., an efficient spiking pipeline must comprise SNNs with matching neural codes.

The main body of this thesis consists of the following two contributions:

- The design of an SNN for replicating the function of the FT. The idea was initially explored in [Lóp+21] using a rate-based neuron model and was later improved in [Lóp+22] employing a novel time-based spiking neuron model called *temporal charge before spike* (TCBS). The weights of the *spiking Fourier transform* (S-FT) are mathematically derived from the FT equations, and the SNN is time encoded, i.e., one spike is enough to represent each data point. The algorithm was implemented and tested on digital neuromorphic hardware, namely the chips from the Loihi family and the SpiNNaker 2 chip.

- The replacement of ADCs using an analogue circuit that directly converts the sensor signals to spikes. The designed *analogue-to-spike encoder* (ASE) provides time-coded spikes compatible with the S-FT. The ASE was introduced in [LRK23], where a circuit prototype was implemented and tested on electric signals produced in a lab. The experiments included the assessment of the S-FT on the spike trains produced by the circuit.

This work also includes the following minor contributions:

- The collaboration in the design of a time-based SNN for object detection, called *spiking OSCFAR* (S-OSCFAR), presented in [Lóp+21]. Further work reported in [Vog+22] included a more comprehensive analysis of the S-OSCFAR parameters and its validation on automotive data.

- The application in [LRK22] of the TCBS neuron model for converting DCNNs to SNNs. The work tested the neuron model for an image classification task.

- The analysis and review in [Vog+22] of the potential of neuromorphic computing for its application to FMCW radar sensors.

- The collaboration in [Kai+22] for applying complex-valued neural networks to compute the localization of targets from FMCW radar data.

- The collaboration in [Ree+25] for optimising resonate-and-fire neurons for computing the precise frequency spectrum of multi-dimensional data.

This work studied the application of SNNs for radar signal processing, prioritising the development of networks with optimal efficiency, particularly emphasizing sparsity and time-coding techniques. The research was an integral approach that started with the sensor analogue signal. The proposed methodologies facilitate substituting ADCs and traditional frequency-spectrum representations with event-based models. The S-FT method was employed for frequency spectrum computation, and an ASE was designed to compute a continuous signal into spikes compatible with the S-FT. Follow-up work delved into using resonate-and-fire neurons. Despite their higher computational demands, these neurons present a

promising alternative for directly computing frequency spectra from analogue signals without needing prior conversion into spikes or digital values. The code needed for implementing the models shown in this thesis is open-source and is available on a public repository[1].

This thesis was the result of a collaboration between the *Technical University of Munich (TUM)* and industrial and academic partners in the context of the KI-ASIC project[2], funded by the Federal Ministry of Research and Education of Germany (*Bundesministerium für Bildung und Forschung*). The KI-ASIC project explored the application of neuromorphic solutions for automotive radar, including developing neuromorphic hardware architectures, neuromorphic computing algorithms, and their implementation in real-world driving scenarios. The role of TUM in the project was to develop neuromorphic algorithms for processing radar signals and create quantitative benchmarks that show the performance of neuromorphic solutions and allow their comparison with traditional signal processing chains.

The work summarized here is the first application of neuromorphic computing to the raw data of radar sensors. The resulting implementation is a proof-of-concept that opens the door to end-to-end neuromorphic radar pipelines where the sensor signal is directly converted to events and no ADC is necessary. The algorithms presented here are also the first applications of SNNs with radar data in open-world scenarios.

1.2.1 Publications

The work introduced in this thesis has been published in the following peer-reviewed contributions:

Journal articles

- Lopez-Randulfe, J., Duswald, T., Bing, Z. and Knoll, A., 2021. Spiking neural network for fourier transform and object detection for automotive radar. Frontiers in Neurorobotics, 15, p.688344. [Lóp+21].
- Lopez-Randulfe, J., Reeb, N., Karimi, N., Liu, C., Gonzalez, H.A., Dietrich, R., Vogginger, B., Mayr, C. and Knoll, A., 2022. Time-coded spiking fourier transform in neuromorphic hardware. IEEE Transactions on Computers, 71(11), pp.2792-2802. [Lóp+22].
- Vogginger, B., Kreutz, F., Lopez-Randulfe, J., Liu, C., Dietrich, R., Gonzalez, H.A., Scholz, D., Reeb, N., Auge, D., Hille, J. and Arsalan, M., 2022. Automotive radar processing with spiking neural networks: Concepts and challenges. Frontiers in Neuroscience, 16, p.851774. [Vog+22].
- Lopez-Randulfe, J., Reeb, N. and Knoll, A., 2023. Integrate-and-fire circuit for converting analog signals to spikes using phase encoding. Neuromorphic Computing and Engineering, 3(4), p.044002. [LRK23].
- Reeb, N., Lopez-Randulfe, J., Dietrich R. and Knoll, A., 2025. Range and Angle Estimation with Spiking Neural Resonators for FMCW Radar arXiv preprint arXiv:2503.00898 [Ree+25]

[1] https://github.com/KI-ASIC-TUM
[2] https://www.elektronikforschung.de/projekte/ki-asic

Conference proceedings

- Kaiser, K., Daugalas, J., Lopez-Randulfe, J., Knoll, A., Weigel, R. and Lurz, F., 2022, September. Complex-Valued Neural Networks for Millimeter Wave FMCW-Radar Angle Estimations. In 2022 19th European Radar Conference (EuRAD) (pp. 145-148). IEEE. [Kai+22].
- Lopez-Randulfe, J., Reeb, N. and Knoll, A., 2022. Conversion of ConvNets to Spiking Neural Networks With Less Than One Spike per Neuron. In 2022 Conference on Cognitive Computational Neuroscience (pp. 553-555). [LRK22].

1.2.2 Acknowledgment of Research Collaborators and Funding

I acknowledge the valuable contributions of the collaborators and co-authors to this work, whose expertise significantly enriched the development of the methods described in this thesis. Nico Reeb, Tobias Duswald, Robin Dietrich, Negin Karimi, Susanne Junghans, Leon Larsen, and Alexander Lenz provided essential insights for the theoretical modelling and testing of the introduced solutions. This work also benefited from the insights provided by colleagues at the Technical University of Dresden and Infineon AG. Additionally, the contributions of students who collaborated on this project, particularly those working on their master's theses in collaboration with this research, were instrumental. The work presented in this thesis was carried out as part of KI-ASIC, a research project funded by the Federal Ministry of Education and Research of Germany under project number 16ES0995.

1.2.3 Thesis structure

This thesis is structured as follows. chapter 2 describes the working principles of FMCW radar and the main processing steps in a traditional radar pipeline for detecting targets' relative position, angle, and velocity. chapter 3 describes the fundamentals of neuromorphic engineering and computing, and the current state-of-the-art. chapter 4 introduces the TCBS neuron model and its application as a replacement for the FT and as a tool for converting DCNNs. chapter 5 introduces an ASE, which was designed for converting analogue signals to spikes for later use by the TCBS. chapter 6 describes the implementation of the TCBS neuron model, the ASE, and the S-OSCFAR for processing analogue signals. The implementations are followed by experiments that validate the different elements for processing radar sensors. Depending on the experiment, the source of the data was i) obtained from a radar software simulator, ii) obtained from an electric function generator, or iii) from real radar sensors. Finally, the work is closed with the concluding remarks in chapter 7.

2

Radar signal processing

Sensor signal processing is a field with a history spanning over a century that deals with the acquisition, transformation, and representation of sensor data. Sensors are a vital component of almost every technological system nowadays, and the study and development of sensor processing pipelines is as old as the electrical engineering discipline itself. Signal processing has been the subject of steady improvement in terms of accuracy, reliability, and efficiency. One of the main drivers of this constant improvement has been the sharp evolution of electronic systems over the last 50 years, thanks to the increasing degree of miniaturization and computing capability per silicon area, as described by Moore's law [Lei+20].

Radars are sensors for locating objects in space based on the reflection of electromagnetic waves. The term is an acronym for *radio detection and ranging*. Radar technology has been under development for more than 80 years. The first real-world applications came in the middle of the XX century and for several decades it was limited to the military field. With the advent of modern electronics and the growth of interest in embedded systems in recent decades, the usage of radar sensors for civil applications has been steadily increasing [Win+14; Has+12]. Autonomous driving is one of the fields with the greatest potential for developing new techniques and architectures for radar sensors, especially for *frequency-modulated continuous-wave* (FMCW) radar. Their low price and robustness against bad weather and lighting conditions make them a great companion for other sensors like LIDAR and vision cameras [Win+14; Pat+17].

This chapter introduces the working principle of FMCW radar sensors in section 2.1. The following sections describe the primary operations of radar signal processing pipelines, including raw data acquisition, frequency spectrum analysis, target detection, and clustering.

2.1 Sensing principles

FMCW radar sensors emit sequences of electromagnetic waves of increasing frequency, called chirps, and sense the echoes generated when they hit an object (see fig. 2.1). Chirps are characterized by their base frequency f_0 and the frequency and time differences since the start and end of the chirp, which we denote as Δf and Δt, respectively. The frequency difference $\Delta f = f_{\max} - f_{\min}$ is also referred to as the bandwidth of the radar chirp. In general, we will assume $f_{\min} = f_0$, and we define the ramp of a chirp S as

$$S = \frac{\Delta f}{\Delta t}\,. \tag{2.1}$$

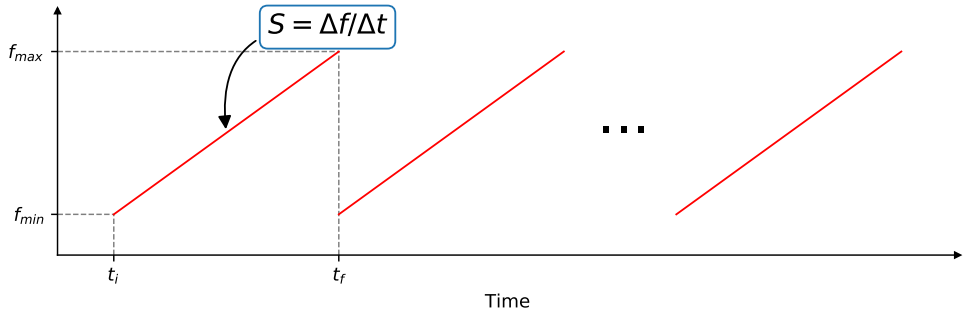


Figure 2.1: Time evolution of the frequency of the emitted signal over several chirps of an FMCW radar.

As angular frequencies are often more convenient, we also define the ramp of the chirp's angular frequency

$$S_\omega = 2\pi S\,. \tag{2.2}$$

Figure 2.2 shows the schematic of a typical FMCW radar sensor. By comparing the waveform of the received echoes with the original signal, we can obtain information about the range and velocity of the objects that provoked the reflections [Jan18; Pat+17]. After combining the transmit and receive waves in an operation called mixing, we obtain a sinusoidal wave called *intermediate frequency* (IF), which has the form

$$s_{\text{IF}}(t) = A_{\text{IF}} e^{-j2\pi f_b t}\,, \tag{2.3}$$

where A_{IF} is the IF's amplitude, f_b is its frequency, called beat frequency, and j is the complex unit. From (2.3), we can estimate the range of a detected target as

$$R = \frac{f_b c}{2S}\,, \tag{2.4}$$

where c is the speed of light. We can also estimate the radial velocity of the target as

$$v = \frac{\lambda \omega_D}{4\pi f_c}\,, \tag{2.5}$$

where ω_D is the Doppler shift frequency over consecutive chirps, and λ is the base wavelength of the emitted signal. Therefore, the data is packed in frames formed by several chirps.

Derivation. *We define the transmitting wave as*

$$s_t(t) = A\cos\left(\omega_t(t)t + \phi_t\right) = A\cos\left(\omega_0 t + S_\omega t^2 + \phi_t\right)\,, \tag{2.6}$$

where A, ω_0, and ϕ_t are the wave's amplitude, base angular frequency, and initial phase, respectively. We can analogously define the receiving wave,

$$s_r(t) = A\cos\left(\omega_r(t)t + \phi_r\right) = \alpha A\cos\left[(\omega_0 + \omega_D)t + S_\omega(t - t_d)t + \phi_r\right]\,, \tag{2.7}$$

where α and t_d are the wave attenuation and the time delay between the transmit and receive waves; and ω_D and ϕ_r are the frequency and initial phase of the wave. An electronic component called mixer combines the emitted and received waves. The output signal of the mixer, $s_m(t)$, is the multiplication of both input signals,

$$s_m(t) = \frac{\alpha A^2}{2}\left[cos((\omega_t + \omega_r)t + (\phi_t + \phi_r)) + cos((\omega_t - \omega_r)t + (\phi_t - \phi_r))\right]\,. \tag{2.8}$$

A low-pass filter after the mixer gets rid of the high-frequency component (i.e., the first cosine term) and reduces eq. (2.8) *to a sinusoidal signal called IF. Assuming the Doppler frequency shift is negligible,* $\omega_D \ll \omega_0$*,*

$$s_{IF}(t) = \frac{\alpha A^2}{2} cos\left[(S_\omega t_d)t + \phi_{IF}\right], \tag{2.9}$$

where $\phi_{IF} = \phi_t - \phi_r$*. We can observe that the IF described by* (2.9) *is a wave with a constant frequency called beat frequency,* f_b*, defined as*

$$f_b = S t_d . \tag{2.10}$$

Thus, the beat frequency f_b *is proportional to the time delay* t_d *between the transmitted and emitted waves [Win+14], which is proportional to the distance R to the object that caused the reflection,*

$$t_d = \frac{2R}{c}, \tag{2.11}$$

where c is the speed of light, and the 2 *in the numerator stands for the round trip of the wave back to the receive antenna. From* (2.10) *and* (2.11) *we can infer R as*

$$R = \frac{f_b c}{2S}. \tag{2.12}$$

If there is a relative velocity v between the radar sensor and the object, the relative phase ϕ_{IF} *between the transmitted and received waves shifts for consecutive chirps changes with an angular frequency*

$$\omega_D = \frac{d\phi_{IF}}{dt}, \tag{2.13}$$

where dt *is the difference in the round-trip time for the electromagnetic waves.* (2.13) *is known as the Doppler effect or Doppler shift. For solving the relative velocity of the target, let us express* (2.13) *in discrete form,*

$$\Delta\phi_{IF} = \omega_D \Delta t . \tag{2.14}$$

We calculate the phase shift $\Delta\phi_{IF}$ *as a function of the base frequency* ω_0 *of the transmitted electromagnetic wave*

$$\Delta\phi_{IF} = \omega_0 \Delta t . \tag{2.15}$$

Combining (2.15) *with* (2.11)*, we obtain*

$$\Delta\phi_{IF} = 2\pi f_0 \frac{2\Delta R}{c}. \tag{2.16}$$

As the change in R for two consecutive chirps is $\Delta R = v\Delta t$*, and applying the equality* $c = \lambda f_0$ *for electromagnetic waves, we can rewrite* (2.16) *as*

$$\Delta\phi_{IF} = \frac{4\pi v \Delta t}{\lambda}. \tag{2.17}$$

Applying the equality for the Doppler frequency in (2.14)*, we rewrite* (2.17) *as*

$$\omega_D = \frac{4\pi v}{\lambda}, \tag{2.18}$$

where the dimensions of ω_D *are* $[rad/chirp]$*. If we use the chirp frequency* f_c *for expressing it in* $[rad/s]$*, we can solve the target radial velocity as*

$$v = \frac{\omega_D \lambda}{4\pi f_c}. \tag{2.19}$$

□

Measurement limitations

The resolution and maximum detectable values for the range and velocity of targets in front of an FMCW radar are extracted from the equations that define the working principle of the sensor. The range resolution ΔR of a radar is the minimum distance between two objects that allows us to distinguish both objects. Assuming that the IF signal is processed with a *Fourier transform* (FT), two frequency components f_1 and f_2 are distinguishable in the frequency spectrum if

$$|f_1 - f_2| > \frac{1}{\Delta t}. \tag{2.20}$$

Δt, which defines the chirp duration, corresponds as well with the time window used for generating the frequency spectrum. Combining (2.20) with (2.10), we obtain the range resolution

$$\Delta R = \frac{c}{2\Delta f}. \tag{2.21}$$

Equation (2.21) implies that the range resolution of an FMCW radar only depends on the sensor bandwidth Δf. We obtain the maximum range R_{max} that the sensor can detect from (2.4) by replacing f_b by the maximum achievable frequency,

$$R_{\text{max}} = \frac{f_{\text{samp}} c}{2S}, \tag{2.22}$$

where f_{samp} is the sampling frequency of the *analog-to-digital converter* (ADC) that converts the IF signal. The properties of the velocity estimation depend on the chirp layout of the sensor. Same as for the range, the velocity resolution Δv depends on the time window used for measuring, in this case, in the chirp dimension. If a frame has N chirps, the angular resolution of the velocity is $\Delta\omega_v = 2\pi/N$. We then use (2.19) to obtain the radial velocity resolution,

$$\Delta v = \frac{\lambda}{2T_f}, \tag{2.23}$$

where $T_f = N\Delta t$ is the total frame time. Finally, the maximum velocity that the sensor can measure is

$$v_{\text{max}} = \frac{\lambda}{4\Delta t}. \tag{2.24}$$

In other words, we can increase the velocity range by reducing the chirp time.

Reflection power

When using radar sensors, the power captured at the receiving antenna P_{rx} indicates the intensity of the signal reflected by an object in the sensor's field of view. Electromagnetic waves propagate homogeneously in the 3D space. If the emitting antenna sends a wave with power P_{tx}, the power per unit area of the emitted signal at a distance R is

$$P'_{\text{tx}} = \frac{P_{\text{tx}}}{4\pi R^2}. \tag{2.25}$$

Likewise, (2.25) is applied again when the wave is scattered after hitting a target. Assuming the distance between the transmit and receive antenna is negligible compared to the distance to the target, we calculate the power per unit area of the wave when it reaches the receiver antenna as

$$P'_{\text{rx}} = \frac{P_{\text{tx}}}{4\pi R^2} \frac{1}{4\pi R^2}. \tag{2.26}$$

The power effectively absorbed by the receive antenna depends on the effective antenna aperture A_p, which is the area of the antenna in the direction of the arrival wave. The A_p of a lossless antenna is calculated as

$$A_p = \frac{\lambda^2}{4\pi}, \tag{2.27}$$

where λ is the wavelength of the received electromagnetic signal. Assuming the transmit and receive antennas have gains G_{tx} and G_{rx}, and the target has a radar cross-section σ, we obtain the *general radar equation*

$$P_{\text{rx}} = \frac{P_{\text{tx}} G_{\text{tx}} G_{\text{rx}} \sigma \lambda^2}{R^4 (4\pi)^3}. \tag{2.28}$$

This equation calculates a target reflection's power for specific antenna parameters and a specific target set-up.

The radar cross-section σ reflects how easily detecting an object by a radar signal is and depends on the object's material, shape, and size. According to the real-world experiments described in [KPP17], typical σ values for conventional vehicles oscillate between 8 and 25 dBsm, The authors also compared the results with the σ of pedestrians (-5 to $0dBsm$) and bicycles ($\sim 10dBsm$). The work in [Dee+20] includes a thorough description of pedestrians RCS. They introduce a mathematical procedure for simulating 3D pedestrians. Typically, co-polarization components range from -10 to 5 dBsm, and cross-polarization components tend to be 10 dBsm smaller.

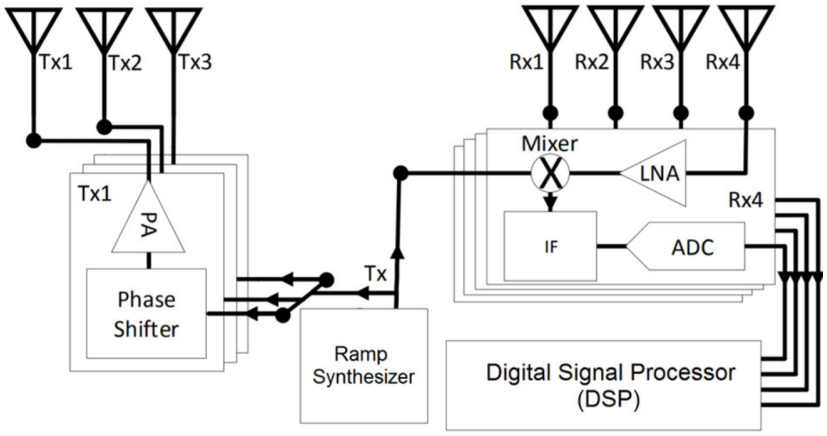


Figure 2.2: Schematic picture of a typical *multiple-input multiple-output* (MIMO) radar, formed by three transmit and four receive antennas. The signals sent to the transmit antennas are combined in a mixer with the signals received from the receive antennas, which results in the IF wave. After digitization, the IF is processed in a digital-signal processor [Vog+22].

2.2 Analogue to digital Conversion

An ADC is the electric component that converts an analogue voltage signal into a digital time series. The ADC is the bridge between the analogue and the digital realms, and it is a crucial stage of sensor signal processing from the energy consumption perspective. ADC design is a decades-long field that lies in the field of electrical engineering and is sometimes ignored by computer scientists when designing signal processing algorithms. Energy consumption is a critical parameter for ADCs, as the operation of these components is not trivial. They typically drain a large percentage of the energy consumed by low-power sensor applications [RG19].

Theoretical studies have found a fundamental lower boundary for the energy consumption of ADCs per processed sample based on the *Neuman-Landauer* principle [Mur13; Mur15]. The energy boundary is expressed as

$$E_{\min} = 8K \cdot T \cdot SNR, \tag{2.29}$$

where K is the Boltzmann constant, T is the temperature, and SNR is the signal-to-noise ratio of the ADC. Even though today's ADCs typically lie two orders of magnitude above this boundary, authors in [Mur13] argue that this gap will hardly improve due to design limitations.

The majority of ADC designs nowadays belong to the successive-approximation register (SAR), delta-sigma ($\Delta - \Sigma$), pipelined, or flash architectures [Mur15]:

- SAR architectures are typically used for low sampling frequencies. Their working principle is based on comparing the input voltage with the output of a digital-to-analogue converter that sequentially approximates the actual value of the input. The input voltage is kept stable with a sample and hold (S/H) circuit during the comparison process. A popular approach for improving the sampling rate f_s of SAR ADCs is to implement a time-interleaved circuit, replicating it M times in parallel with equidistant phase shifts. Thus, the effective sampling time becomes M/f_s [Rey+19].

- $\Delta - \Sigma$ architectures are well-known for the high sampling rates they can achieve. A first circuit (delta Δ) subtracts the previously stored sample from the input voltage and converts the result into a pulse frequency. A second circuit (sigma Σ) counts the produced pulses and stores the count in a digital register.

- Pipelined ADCs are typically used for mid-range sampling frequencies, lying between the SAR and $\Delta - \Sigma$ architectures. This architecture consists of a sequential encoding of the input voltage using low-bit resolution flash ADCs to generate the most significant bits at a time. The remaining bits are then passed to the next stage until the desired bit resolution is achieved.

- Flash ADCs are used for the fastest sampling frequencies, i.e., frequencies higher than those of the $\Delta - \Sigma$ architecture. Their working principle is the simplest: they create a voltage ladder from the input voltage and compare the obtained voltages with a series of reference voltages. The output from these comparators is combined using simple logic gates that generate the final bit distribution. This simple approach leads to their fast conversion speeds but also makes them very expensive and limits the output bit resolution. Adding an S/H circuit at the input is unnecessary for this architecture.

2.3 Frequency domain Representation

Most natural signals are formed by waves with patterns repeated over time. Such waves carry information in properties like the frequency, amplitude, and relative phaseof those patterns. Measuring these properties and understanding their meaning is paramount for many engineering applications. Frequency domain representation consists of the mapping of periodic waves from a time-based system into a frequency-based system. This allows for further analysis and processing of the data. When working with digital data, this processing stage decomposes a time series into the frequencies that dominate the signal.

An essential characteristic for classifying frequency-based applications is whether they analyse specific frequencies or the whole spectrum. Applications that analyse a single frequency or a small group of frequencies tend to be more precise and efficient, as more computational resources are usually available for assessing each frequency. These approaches are typically based on filters tuned for the desired frequencies.

The most popular technique for mapping signals to their whole frequency spectrum is the *Fourier transform* (FT), an integral transform introduced by Joseph Fourier in 1822. The general expression of the FT algorithm calculates the frequency spectrum of time-varying signals according to

$$y(\nu) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi\nu x} dx\,, \tag{2.30}$$

where $y(\nu)$ is the Fourier transform of the integrable and continuous function $f(x)$ at the frequency ν. The FT is a lossless transformation, as the original signal can be recovered by applying the Fourier inversion formula

$$f(x) = \int_{-\infty}^{\infty} y(\nu) e^{j2\pi\nu x} d\nu\,. \tag{2.31}$$

Researchers have provided different versions of the FT over the last decades, aiming to optimise its computational efficiency and adaptability to dynamically changing data. Alternative algorithms like the wavelet transform address some of the caveats of the FT.

2.3.1 Discrete Fourier Transform

The *discrete Fourier transform* (DFT) is one of the central processing operations applied to radar data in the digital domain. It is obtained by applying the general FT algorithm (2.30) to a digitised signal. In automotive radar applications, it is typically applied to at least two different dimensions, i.e., the range and velocity. Whereas the range-frequency spectrum is obtained by directly applying the FT to the magnitude of an input chirp, the velocity is extracted from the phase change in the range frequency components over consecutive chirps. This phase shift is caused by the Doppler effect, described in more detail in section 2.1.

A discrete version of (2.30) is needed to process the time series obtained after digitising a sensor signal. Thus, the DFT is the result of applying the general expression (2.30) to discrete samples,

$$Y_k = \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{-j2\pi\frac{kn}{N}}\,, \tag{2.32}$$

where Y_k is the kth bin of the Fourier transform of the discrete-time series X of length N, and X_n is the nth element of the time series. The division by N normalises the transform with the number of samples. (2.32) can also be expressed in the trigonometric form

$$Y_k = \frac{1}{N} \sum_{n=0}^{N-1} X_n \left[cos\left(\frac{2\pi}{N} kn\right) - j \cdot sin\left(\frac{2\pi}{N} kn\right) \right]. \tag{2.33}$$

To compute the whole spectrum, we need to evaluate (2.32) N times ($k \in [0, N-1]$), which requires N^2 multiplication operations and $N(N-1)$ addition operations. Thus, the computational complexity of the DFT is $\mathcal{O}(N^2)$.

The FT generates a complex-valued spectrum with N bins, where $(N/2)-1$ bins represent positive frequency values, $(N/2)-1$ bins represent negative ones, and the remaining two bins

represent the DC component. The amplitude of the FT bins is proportional to the energy of the wave at their corresponding frequency,

$$Y_k = \frac{A_k^2}{4}, \tag{2.34}$$

where A_k is the amplitude of the input wave for the kth frequency. The result of the FT is symmetric for a real input, meaning that the positive and negative parts of the spectrum are identical and half of the wave's energy is displayed on each side, i.e., we can discard the last $N/2$ bins and reduce (2.34) to

$$Y_k = \frac{A_k^2}{2}. \tag{2.35}$$

The half-spectrum is typically arranged so the first bin represents the offset or zero-mode of the transform, Y_0. The lowest frequency that a DFT can detect is the one that translates into an entire cycle within the sampling window. This frequency defines the frequency resolution Δf of the DFT. If the input signal has N equally spaced values sampled with a frequency f_s over a sampling time window of length ΔT, we define Δf as

$$\Delta f = \frac{1}{\Delta T} = \frac{f_s}{N}, \tag{2.36}$$

which is analogous to (2.21) in section 2.1. The resolution Δf relates to the precision in the estimation of a frequency component. However, for distinguishing two consecutive frequency components, f_1 and f_2, they need to be at a distance of at least twice the resolution, $|f_1 - f_2| > 2\Delta f$. Smaller distances would lead to a homogeneous peak, making it impossible to distinguish the two peaks. The DFT has $(N/2)-1$ frequency bins, and the last bin Y_N bin represents the frequency

$$f_N = \frac{f_s}{N}\left(\frac{N}{2} - 1\right) = \frac{f_s}{2} - \frac{f_s}{N}. \tag{2.37}$$

By combining (2.36) and (2.37), we obtain the maximum frequency that the DFT can map into the spectrum,

$$f_{\max} = f_N + \Delta f = \frac{f_s}{2}, \tag{2.38}$$

which corresponds as well to the *Nyquist frequency* of the sampled signal, i.e., it is impossible to recover information at frequencies higher than $f_{\max}$.

The possibility of inverting the FT mapping makes it an ideal choice for generating the frequency spectrum, processing the signal in the frequency domain, and inverting the result back to the time domain. The FT algorithm is costly, so its usage should be limited to applications that provide an actual improvement compared to alternative algorithms in the time domain.

2.3.2 Fast Fourier transform

The *fast Fourier transform* (FFT) is a computationally optimized version of the DFT, and probably the most popular approach for implementing the frequency spectrum analysis. Cooley and Tukey proposed the first FFT algorithm in 1965 [CT65], and was later considered one of the most important algorithms of the XX century [DS00]. The FFT is based on a recursive split of the DFT algorithm into smaller transforms to exploit the symmetry in calculating the different output bins. This is known as the Danielson-Lanczos lemma, which states that a

DFT of size N equals the sum of two DFTs of size $N/2$, one formed by the even points and the second formed by the odd points,

$$Y_k = Y_k^e + t_k Y_k^o \,, \tag{2.39}$$

where Y_k^e and Y_k^o are the DFTs of the even and odd input points, respectively, and t_k is a twiddle factor. The output of the DFT can thus be replicated with linear combinations of smaller FTs. The intermediate DFTs are connected in pairs with crossed connections to the following blocks, leading to what is known as the butterfly connectivity structure, which is represented in fig. 2.3. The connectivity pattern limits the number of input samples N: for a radix-2 architecture, one of the simplest and most popular, the butterfly blocks are connected in groups of two inputs. This imposes the input dimensionality as a power of two, $N = 2^m$. Generalizing, a radix-R FFT requires the input to contain R^m samples.

The simplification of the computation of the FT using the FFT architecture leads to a computational complexity of

$$\mathcal{O}(N \cdot \log N) \,, \tag{2.40}$$

which contrasts with the complexity of the original DFT of $\mathcal{O}(N^2)$.

Derivation. *We first calculate from* (2.32) *the value of the* $(k+N)$*th bin as*

$$\begin{aligned} Y_{k+N} &= \sum_{n=0}^{N-1} X_n e^{-j2\pi \frac{(k+N)n}{N}} \\ &= \sum_{n=0}^{N-1} X_n e^{-j2\pi \frac{kn}{N}} e^{-i2\pi n} \,. \end{aligned} \tag{2.41}$$

Analogously, the value of the $(k+N/2)$*th bin is*

$$Y_{k+N/2} = \sum_{n=0}^{N-1} X_n e^{-j2\pi \frac{kn}{N}} e^{-i2\pi \frac{n}{2}} \,. \tag{2.42}$$

As $e^{-j2\pi n} = 1$ *we obtain the properties*

$$Y_{k+N} = Y_k \tag{2.43}$$

and

$$Y_{k+N/2} = -Y_k \,. \tag{2.44}$$

We now rephrase (2.32) *as*

$$\begin{aligned} Y_k &= \sum_{n=0}^{\frac{N}{2}-1} X_{2n} e^{-j2\pi \frac{k2n}{N}} + \sum_{n=0}^{\frac{N}{2}-1} X_{2n+1} e^{-j2\pi \frac{k(2n+1)}{N}} \\ &= \sum_{n=0}^{\frac{N}{2}-1} X_{2n} e^{-j2\pi \frac{kn}{N/2}} + e^{-j2\pi \frac{k}{N}} \sum_{n=0}^{\frac{N}{2}-1} X_{2n+1} e^{-j2\pi \frac{kn}{N/2}} \,. \end{aligned} \tag{2.45}$$

The previous expression is the Danielson-Lanczos lemma. It has the form $Y_k = Y_k^e + t_k Y_k^o$*, where* Y_k^e *and* Y_k^o *are the smaller DFTs for even and odd indexed samples, respectively, and* $t_k = e^{-j2\pi k/N}$ *is called the twiddle factor. We know that the terms* Y_k^e *and* Y_k^o *for* k *and* $k + N/2$ *are identical thanks to the property in* (2.43)*, and the twiddle factor is negated,* $t_{k+N/2} = -t_k$ *due to* (2.44)*.*

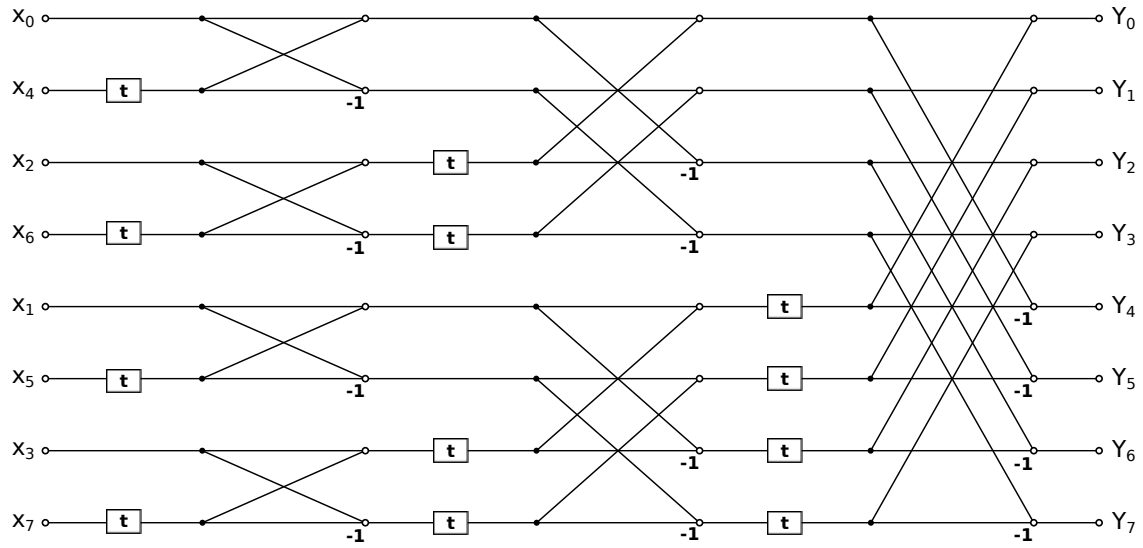


Figure 2.3: Structure of a radix-2 FFT algorithm for an input with 8 points. The white connection nodes represent additions, where inputs with a -1 next to them are negated. The white boxes represent the multiplication with the twiddle factors.

Therefore, we only need to compute the Fourier coefficients for half of the output points. Applying this simplification, the calculation for the values of (2.45) *for* $k > N/2$ *yields*

$$Y_{k+N/2} = E_k Y_k^e - t_k Y_k^o \,. \tag{2.46}$$

We obtain the FFT by applying (2.45) m *times recursively on the resulting DFTs, where* $N = 2^m$*. In the last stage,* $N/2$ *DFTs is formed by two multiplications, leading to a total of* N *computations (*N *additions and* N *multiplications). In the previous stage, we have* $N/4$ *DFTs formed by four multiplications. Generalizing, the computation of a given stage* $m1$ *contains* $N/m1$ *DFTs with* $m1$ *multiplications and* $m1$ *additions each. We obtain the FT iterating back and reconstructing all* m *stages, yielding a computational complexity of*

$$\mathcal{O}(N \cdot m) = \mathcal{O}(N \cdot \log N) \,. \tag{2.47}$$

The iteration rule in (2.45) *and the structure depicted in fig. 2.3 correspond to a radix-2 butterfly structure, as each DFT is split into two smaller DFTs. If we instead split each DFT into 4, 8, or more DFTs, we obtain butterflies with more extensive connectivity, i.e. radix-4 butterfly, radix-8 butterfly, and so on.*

□

Table 2.1 summarizes the main parameters and performance indicators for FFT accelerators published recently. The works in [Che+18] and [Guo+14] consist of variable size FFT hardware accelerator for digital signal processing applications. The accelerator in [Che+18] is based on matrix transposition and uses a hybrid approach for the twiddle factors, i.e., factors used several times are stored in look-up tables, and factors used only once are directly calculated with an algorithm. The chip described in [McK10] computes the FT using a radix-2 FFT architecture, meant for low-power digital signal processing applications. The twiddle factors are stored in a look-up table. The work in [Gon+21] is specially designed for radar processing. It implements a dual-radix FFT architecture, employing radix-4 butterflies for computing the range FT and radix-2 butterflies for the Doppler and angle FTs.

Table 2.1: Specifications and performance indicators for state-of-the-art FFT accelerators.

	[Guo+14]	[McK10]	[Che+18]	[Gon+21]
Process (nm)	65	90	45	22FDX
Voltage	1	1.3	0.9	0.6
Stream Number	2	2	2	1
Area (mm^2)	4.6	-	2.4	0.024
Power (mW)	172.38	38.56	91.3	6.40
Frequency (MHz)	500	100	1000	150
Exec Time (us)	2.81	73.15	1.38	8.8

2.3.3 Sparse Fourier Transform

The motivation of a sparse FT is to drastically reduce the computational complexity of the FFT in data with few dominant frequencies. For achieving this goal, the sparse FT analyses only the k most dominant frequency components from the N samples of the input signal, as opposed to the N bins that are analysed in an FFT [Gil+14]. The most straightforward case is solving a single-frequency problem. A high-speed approach is to recover the single frequency by comparing it to consecutive samples. This method has a complexity $\mathcal{O}(1)$ but is highly susceptible to noise. Alternatively, the sparse FT performs a binary search: The N samples are iteratively divided into several subgroups, and the algorithm searches for the subgroup to which the frequency belongs, based on angle proximity.

Depending on the value of k, we distinguish two different types of sparse FTs:

- Exactly k-sparse case, when k is the actual number of meaningful objects in the input
- Approximately k-sparse case, when k is just an approximation of the number of objects in the input

Whereas the computational complexity of the FFT can not beat the limit $\mathcal{O}(N \log N)$, the sparse FT promises computational complexities of $\mathcal{O}(k \log N \log(N/k))$, where k is the number of frequencies that are to be found. The complexity is reduced to $\mathcal{O}(k \log N)$ for the exactly k-sparse case, i.e., when the number of targets is known beforehand. The sparse FT provides big computational gains when $k \ll N$. Experiments in [Gil+14] show computational gains for sparsities lower than 0.1%, i.e., less than one dominating frequency per 1000 input samples. Sparse FT is an efficient alternative to the FFT for highly sparse data, like GPS synchronization or big data. This is, however, not the case of FMCW radars, as they generally record scenes with several targets and sample sizes $N \leq 2^{10}$. Moreover, the number of dominant frequencies is generally unknown in perception tasks.

2.3.4 Short-Time Fourier Transform

The *short-time Fourier transform* (STFT) is a modification of the FT intended to localize the frequency components in time, i.e., the algorithm generates sequential frequency spectra for different times to reflect changes in the input over time. To achieve this, we add a window function $w(t-\tau)$ to (2.30) that yields zero outside the local region of interest,

$$y(\nu, \tau) = \int_{-\infty}^{\infty} f(x) w(t-\tau) e^{-j2\pi\nu x} dx\,. \tag{2.48}$$

This way, we split the input data over time and compute the FT over the different temporal sectors. The output of (2.48) has one more dimension than the FT for representing time and can find temporal variations in the input signal structure. The downside of this approach is a loss in the frequency resolution, as the input signal will now contain fewer samples. Some authors also call this method windowed-FT [Gra95].

2.3.5 Wavelet Transform

The wavelet transform was conceived as an alternative to the FT for applications involving non-stationarydata, where features often appear in the form of short transients. For example, speech data is characterized by irregular, short patterns representing the different consonants and vowels that we produce. The FT falls short in these cases, as its analysis does not use a sense of locality [SIA09; Gra95]. Instead, the FT integrates the input signal with periodic sine waves and detects regular, periodic patterns in the input. In other words, the FT cannot locate the region in time when an oscillation occurs. The STFT partially solves this issue by splitting the frequency analysis into different temporal sections and narrowing the spectrum generation to the selected windows. This comes at the cost of a lower frequency resolution. Analogous to the FT, the wavelet transform integrates a data series with a continuous function ψ. However, the mapping of a signal x with the wavelet function ψ occurs over different scales and locations,

$$w(s,\tau_w) = \int_{-\infty}^{\infty} x(t)\psi^*\left(\frac{t-\tau_w}{s}\right)dt\,, \tag{2.49}$$

where s and τ_w are the scaling factor and time shift, respectively, and $(\cdot)^*$ is the complex conjugate operator. Whereas the FT uses periodic sine and cosine functions, ψ is a brief and irregular oscillation over time. Moreover, it is not limited to a specific function like in the FT, but to an arbitrarily large set of functions. The choice of the function is application-specific. Some popular functions are the *Morlet* wavelet, the *Daubechies* wavelet, or the *Paul* wavelet [Gra95; TC98] Coarse and fine scales are used to find low-and high-frequency patterns, respectively. The wavelet transform results in a 2D mapping, where one axis belongs to the scale or frequency of the pattern, and the other axis belongs to its localization in time.

Although automotive scenarios are dynamic and objects in the surroundings are continuously moving, the chirp rate of FMCW radar sensors is several orders of magnitude faster than the velocity of the targets. This slow pace of change makes the wavelet transform unnecessary for this application, as its main benefit is not exploited in these scenarios.

2.4 Angle of arrival estimation

Section 2.1 introduced the relationship between the frequency components of the IF signal and the range and velocity of the targets in front of a radar sensor with a single antenna. In some cases, sensors comprise *multiple-input multiple-output* (MIMO) antenna channels, where several transmit and receive antennas are built together with a specific geometric layout. Figure 2.2 shows the example of a radar with such layout. This method allows the transfer of more data in parallel and, in the case of FMCW radar sensors, makes it possible to retrieve information about the incidence angles of the detected objects relative to the orientation of the sensor. Thus, for MIMO sensors, a complete spatial localization of the target includes the estimation of the *angle of arrival* (AoA), i.e., the target's orientation, together with its range and relative speed. The AoA estimation can include the angles with

the horizontal and vertical planes, referred to as azimuth and elevation, respectively. This section describes the most relevant methods for estimating the AoA when using an FMCW radar sensor. The choice depends on factors such as the required resolution. computational complexity, or sensor specifications like the number of virtual antennas. For the sake of simplicity, the section focuses only on the estimation of the azimuth, as the methods for calculating the elevation are identical, with the added complexity of dealing with one extra dimension.

A transmit and receive antenna combination in a MIMO sensor results in a virtual antenna with a signal path defined by a specific incidence phase [Eck+18]. For a sensor with n_{tx} transmit antennas and n_{rx} receive antennas, the total number of virtual antennas is

$$L = n_{tx} \cdot n_{rx} \,. \tag{2.50}$$

The AoA is calculated by detecting phase changes across the virtual antennas. Some approaches include superresolution techniques such as MUSIC or maximum likelihood models [Pat+17]. Looking at the brain, this problem resembles sound localization, which uses interaural time differences (ITD) for the AoA computation. In biology, highly experienced echo-locators such as bats employ interaural level differences (ILD) instead, which, in contrast to the ITD using their tiny heads, can capture a wide diversity of target cross-sections at different ranges by sensing pressure differences across their ears. Engineering ITD methods require the concept of phase locking and delay lines so that specific neurons show a high firing rate when a particular frequency arrives at a specific AoA [CK90]. The concept has been proven in neuromorphic hardware with spiking neurons [Pfe+13].

2.4.1 Phase-comparison monopulse

The simplest method for estimating the AoA involves applying the known geometry of the antenna array to infer the angle from the phase shift between antennas. The phase-comparison monopulse technique is a method that relies on phase information to determine the angle of arrival of a target. A phase-comparison monopulse radar setuptypically uses two antennas. Assuming that the distance R to the object (2.4) is much longer than the distance d between antennas, the incoming wave's incidence angle Θ will be the same for all antennas. Applying trigonometry to the MIMO geometric representation in fig. 2.4, the phase difference $\Delta\phi$ of the incoming wave between two consecutive antennas is

$$\Delta\phi = 2\pi \frac{dsin(\Theta)}{\lambda} \,. \tag{2.51}$$

By precisely measuring the phase in the antennas, we can apply (2.51) for extracting the target phase. This method is limited to situations where it is possible to isolate the reflection from a specific target . In any case, it serves as the basis for more sophisticated methods like using the FT in the angular dimension or beamforming, detailed in section 2.4.2 and section 2.4.3.

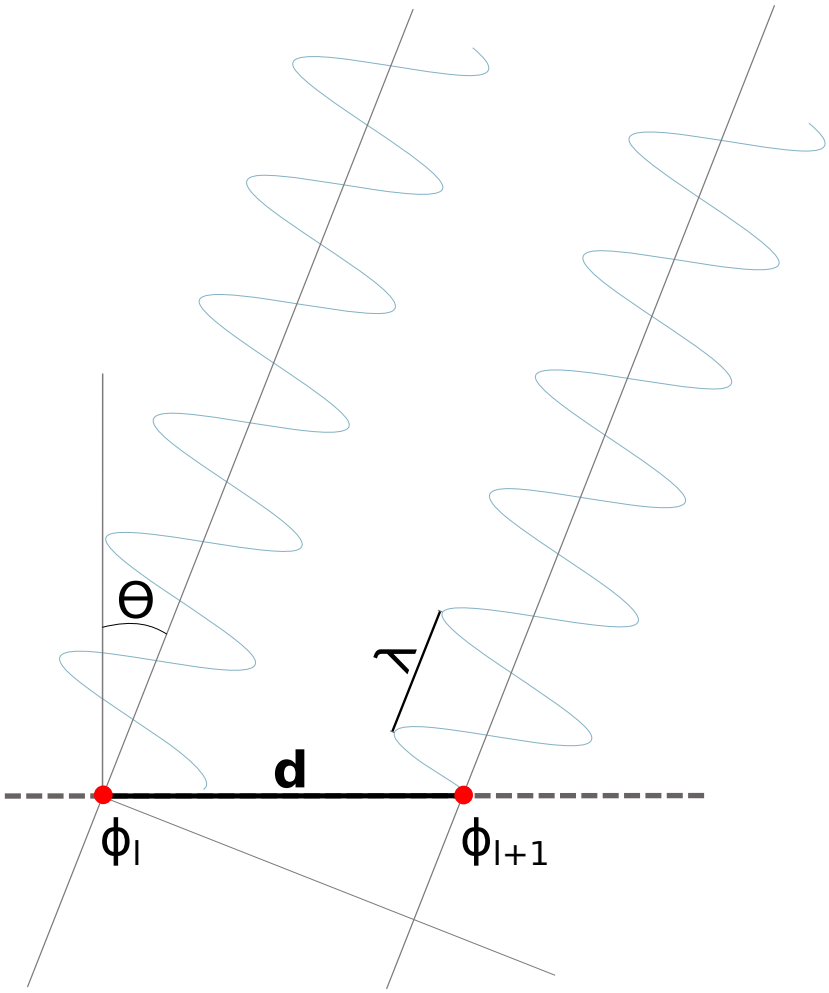


Figure 2.4: Geometry of a wave signal arriving to an array with two antennas. The incidence angle Θ is constant for all antennas, and the phase shift $\phi_l - \phi_{l-1}$ depends on Θ, the signal wavelength λ, and the distance d between antennas.

2.4.2 Frequency-based AoA

Instead of extracting the AoA using only the information of two antennas, we can apply the FT method and generalize (2.51) to assess the relationship between the signal's AoA and the incidence phase for an antenna array. The angle-FT can be combined with a range-FT for generating a range-angle 2D heat map, akin to the one in fig. 2.5, that facilitates the identification of multiple targets in two dimensions. Assuming a constant distance d between consecutive antennas, we calculate the phase at the lth virtual antenna as

$$\phi(l) = 2\pi \frac{l d sin(\Theta)}{\lambda}. \tag{2.52}$$

We can apply this equality to define the incoming signal as

$$s_r(l) = e^{-j(\omega t + \phi(l))} = e^{-j\omega t} e^{-j\phi(l)}. \tag{2.53}$$

At a specific time t, the first term in (2.53) stays constant, and the second term changes according to the spatial frequency given by (2.52). Note the similarity between (2.53) and (2.3), where the dependency over time t is replaced by a dependency in the spatial dimension defined by the antenna position l. In the same way that the FT can be applied in the

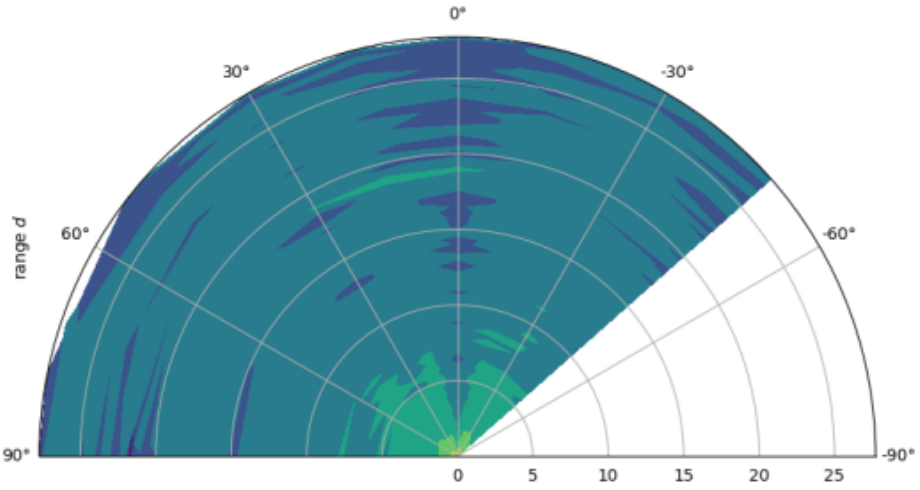


Figure 2.5: Output of a 2D FFT in the range and angle dimensions for an FMCW radar frame. The range and AoA magnitudes correspond to the radial and angular directions, respectively.

time dimension, we can apply the FT to extract the frequency spectrum over the antenna dimension. Thus, we can find the AoA by extracting the dominant phases and applying (2.52) afterwards. We can generate the frequency spectrum using the FT defined in (2.32),

$$Y_p = \frac{1}{L}\sum_{l=0}^{L-1} s_r(l) e^{-j2\pi\frac{kl}{L}}\,, \tag{2.54}$$

where Y_p is the pth phase frequency component, and $0 \leq p \leq L-1$. Contrary to the range FT, the output of the angle FT is not symmetric. The first bin, $p = 0$, corresponds to an AoA $\Theta = 0$, the following $L/2$ bins correspond to negative angles, and the last $L/2-1$ bins correspond to positive ones. Figure 2.5 depicts an example of a range-angle map of an FMCW radar frame obtained after applying the FFT to the range and angle dimensions. We obtain the maximum observable AoA applying (2.51). To distinguish the incidence angles without ambiguity, the resulting phase must stay in the range $\Delta\phi \in [-\pi, \pi]$, leading to

$$\Theta_{\text{max}} = sin^{-1}\left(\frac{\lambda}{2d}\right). \tag{2.55}$$

The angle resolution is given by the formula

$$\Delta\Theta = \frac{\lambda}{Ld \cdot cos(\Theta)}\,, \tag{2.56}$$

which indicates that the angular resolution is best at $\Theta = 0$ and starts decreasing until reaching Θ_{max}. To avoid aliasing, antennas are generally placed at a distance $d = \lambda/2$. Assuming this equality, (2.56) can be simplified to

$$\Delta\Theta = \frac{2}{L \cdot cos(\Theta)}\,. \tag{2.57}$$

2.4.3 Beamforming

Beamforming is a spatial filtering technique used in applications where signals originate from multiple sensors or antennas with a specific array-like geometry. The same way temporal filtering relates to the modification of a signal based on its intensity over time, spatial filtering relates to the modification of a signal based on its intensity over the space, i.e., The signal is filtered based on the spatial direction where it originated [VB88].

In beamforming, a processor called beamformer controls the phase and intensity of the signal at each antenna. Hence, they create constructive and destructive interference on the final emitted or received signal (see fig. 2.6). In a constructive setup, the signal sent to or received from a specific direction is enhanced, and the rest of the directions in the angle spectrum are cancelled or reduced. Analogously, in a destructive setup, the signal from a specific direction is reduced, and the rest of the directions are maintained.

For a scenario with a MIMO sensor with L antennas, the incoming data comprises L input signals $x_l(t)$, where $1 \leq l \leq L$. When exposed to a single sinusoidal source with an AoA Θ, each antenna l will provide a response signal

$$x_l(t) = e^{j(\omega t - \Delta_l(\Theta))}, \tag{2.58}$$

where j is the complex unit and $\Delta_l(\Theta)$ is the phase shift of antenna l for the specific Θ. We can represent the information arriving from the L channels in the vector form

$$\boldsymbol{x(t)} = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_L(t) \end{bmatrix}. \tag{2.59}$$

A beamformer filters the signal by multiplying the data from the L antennas by a weight vector $\boldsymbol{b(t)}$,

$$y(t) = \boldsymbol{b(t)} \cdot \boldsymbol{x(t)}, \tag{2.60}$$

where $y(t)$ is the output of the beamformer at time t. For the simple case where the AoA is constant over time, the phase difference Δ_l of each antenna stays constant. In this case, the beamformer modifies the signal from each antenna by using fixed vector weights, $\boldsymbol{b(t)} = \boldsymbol{b}$, also called the steering vector. The lth element of $\boldsymbol{b}$ is defined as

$$b_l = e^{j\Theta_{b,l}}, \tag{2.61}$$

where $\Theta_{b,l}$ is the phase correction introduced by the lth beamformer channel. Assuming a constructive beamformer that enhances signals arriving from the direction Θ_0, the phase correction for the lth antenna is $\Delta_l(\Theta_0)$, and (2.60) yields

$$y(t) = \sum_{l}^{L} e^{\Delta_l(\Theta_0)} e^{j(\omega t + \Theta_0 - \Delta_l(\Theta_0))} = \sum_{l}^{L} e^{j(\omega t + \Theta_0)} = L e^{j(\omega t + \Theta_0)}. \tag{2.62}$$

We can observe that the output in (2.62) is the aggregation of all inputs with their phases aligned for the direction Θ_0, resulting in a signal L times more powerful than the individual inputs.

The applications where the AoA changes over time need an *adaptive beamforming* that dynamically modifies the weights based on the conditions of the input signals. In other words, the beamformer steers the main beam to the optimal direction [KS15]. This adjustment is performed by feeding the output of the beamformer to an optimization algorithm block that minimizes an error measurement.

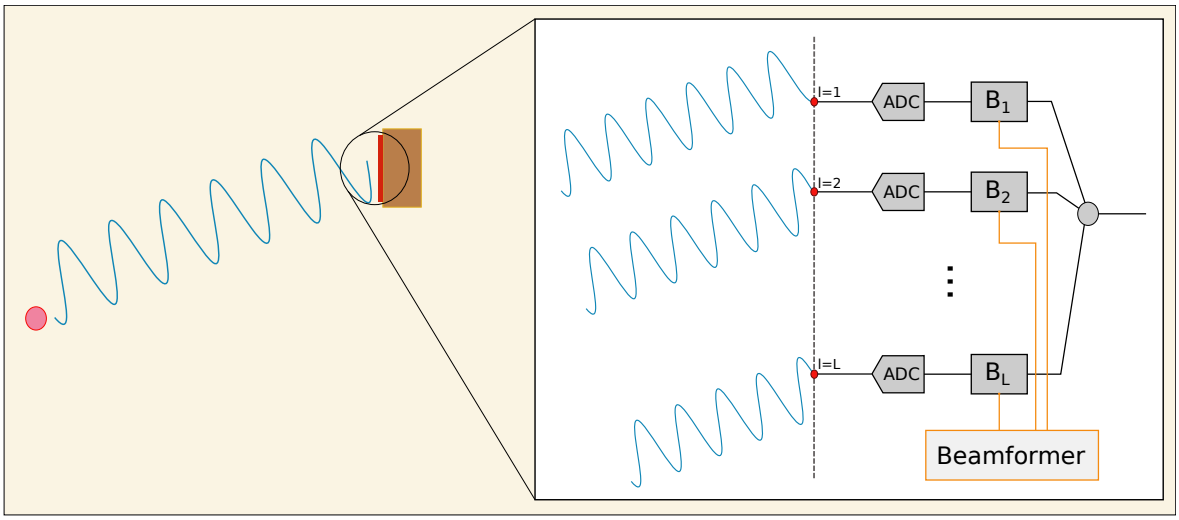


Figure 2.6: Representation of a beamformer applied to L antennas receiving a wave from a specific source. On the left, the source object is represented in red and the emitted wave in blue. The wave arrives to the sensor array, in red, with a direction of arrival Θ relative to the antenna plane. After converting the input signals to digital values on an ADC, the beamformer multiplies the signal from each antenna l by a beamforming coefficient b_l.

An interesting field of application of beamforming is millimetre wave communications [KS15]. The goal of beamforming in these applications is to enhance the final gain of the received signal by creating directional communication, which minimizes the path losses due to obstacles or humidity in the air. A general approach is to start the sensing operation with a sector-level sweep, where the optimal antenna sectors are selected, followed by a refinement phase where the phase array is determined and an optional tracking phase for adjusting channel changes.

Beamformers are also popular in automotive radar applications to reduce interference from other radar sensors. The beamformer estimates the received signal and minimizes the intensity of an AoA that increases the error signal [RDP18]. The adaptive algorithm makes use of an optimization strategy for correcting the received signal, e.g., by using a least mean squares algorithm [RDP18] or an iterative adaptive algorithm [Eck+18].

One of the most popular adaptive beamforming techniques is *Capon*. Its working principle is to adjust the response to noisy input by minimizing the variance of the incoming signals without introducing distortions to the carried information and maximising the signal-to-noise ratio of the signal of interest (SOI) [HN98; LSW03]. The input variance R when receiving information from L uncorrelated sources is

$$R = \sigma_0 \boldsymbol{a_0} \boldsymbol{a_0}^* \sum_{l}^{L} \sigma_l \boldsymbol{a_l} \boldsymbol{a_l}^* + Q\,, \tag{2.63}$$

where σ_l and $\boldsymbol{a_l}$ are the power and steering vector of the lth arriving signal, and the first index corresponds to the signal we want to optimize. $(\cdot)^*$ is the conjugate transpose operation, and Q is the noise matrix. A Capon beamformer adjusts the beamformer coefficients $\boldsymbol{b}$ by optimizing the expression

$$\min \boldsymbol{b}^* R \boldsymbol{b}\,, \tag{2.64}$$

and satisfying the condition

$$\boldsymbol{b}^* a_0 = 1\,. \tag{2.65}$$

From (2.65) and (2.64), it can be derived that the coefficients $\boldsymbol{b_0}$ that maximize the SNR for the first signal are

$$\boldsymbol{b_0} = \frac{R^{-1}\boldsymbol{a_0}}{\boldsymbol{a_0}^* R^{-1}\boldsymbol{a_0}}. \tag{2.66}$$

2.5 Object detection – Constant false-alarm rate

In the context of radar signal processing, object detection is the task of segmenting frequency spectrum data into bins that belong to targets and bins that belong to the background or noise. Contrary to data from other sensors like cameras, the intensity of radar reflections and noise decreases with the range, so radar object detection algorithms must adapt to the characteristics near the regions of interest. The most widely used family of algorithms for object detection in FMCW radar applications is the *constant false alarm rate* (CFAR). CFAR algorithms compare each input value with a threshold representing the noise level in its neighbourhood. They consist of a moving window that covers the whole input and determines whether the central point in the window, also called cell-under-test, belongs to a target. To do so, the intensity of this cell is compared with a false-alarm threshold, which is obtained as a function of the intensity of the neighbouring cells [JYB16].

At a given iteration, the sliding window contains the cell under test in the middle, with a value X_C, and an equal number of neighbour cells $N_N/2$ on each side. The N_G closest cells to the cell under test are called guard cells X_G. These guard cells are ignored to increase the algorithm's robustness, as the intensity of X_C can influence their value. Figure 2.7 illustrates two CFAR algorithms, with the data structure inside the moving window on the left side of the diagrams. X_C is modified by a scaling factor α and compared with the power of the noise P_N, which is obtained from the intensities of the neighbour cells. The CFAR algorithm yields 1 if αX_C is larger than P_N and 0 otherwise,

$$\text{OS-CFAR}(X_C) = \begin{cases} 1, & \text{if } \alpha X_C > P_N \\ 0, & \text{otherwise.} \end{cases} \tag{2.67}$$

The parameterization of CFAR algorithms involves setting the values for α, N_G, and N_N. This parametrization depends on the configuration of the radar sensor and the type of scenarios it will face. Moreover, the computation of the CFAR near the edges of the input map requires the padding of the data. A common approach is to pad the map with zeroes or the average intensity.

Depending on how P_N is calculated, we distinguish among different CFAR algorithms. The *cell-averaging CFAR* (CA-CFAR) computes the threshold as the average value μ_x inside the window, i.e., P_N is calculated by computing the average of the neighbour cells

$$P_N = \mu_x = \frac{\sum_n^{N_N} X_n}{N_N}. \tag{2.68}$$

Two alternatives to the CA-CFAR are greatest of cell average CFAR and smallest of cell average CFAR, where a noise estimation is individually computed for the neighbours preceding the cell under test and the neighbours leading it. The value P_N equals the largest or smallest of the two estimations, depending on the version. These versions show more robustness towards noise in the edge of the targets [JYB16].

Alternatively, the *ordered-statistics CFAR* (OS-CFAR) [Roh83] computes the threshold in a given window as a function of the kth largest value inside it, i.e., P_N is calculated by sorting

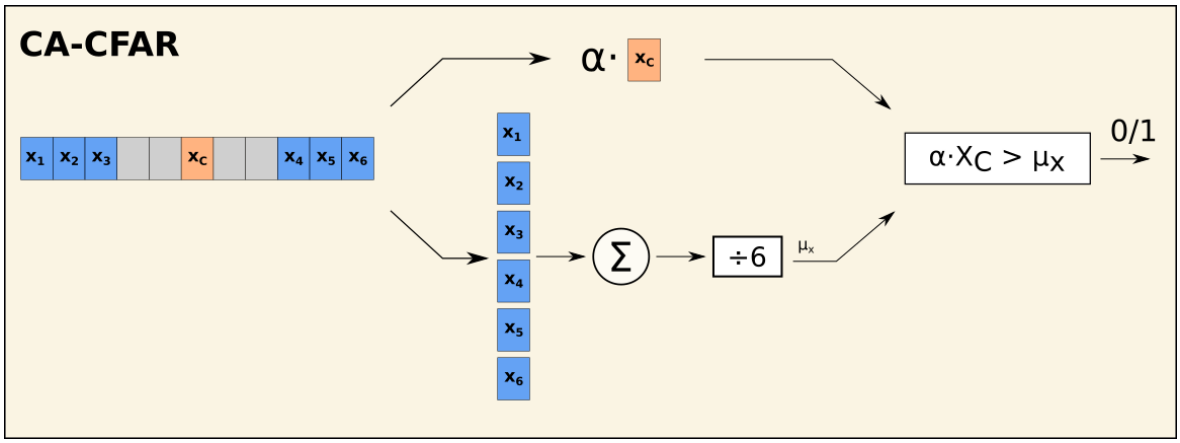
the neighbour cells $X_N = \left(X_1, X_2 ... X_{N_N}\right)$ in descending order and selecting the kth value,

$$P_N = \max_k \{x_i, \forall x_i \in X_N\}\,. \tag{2.69}$$

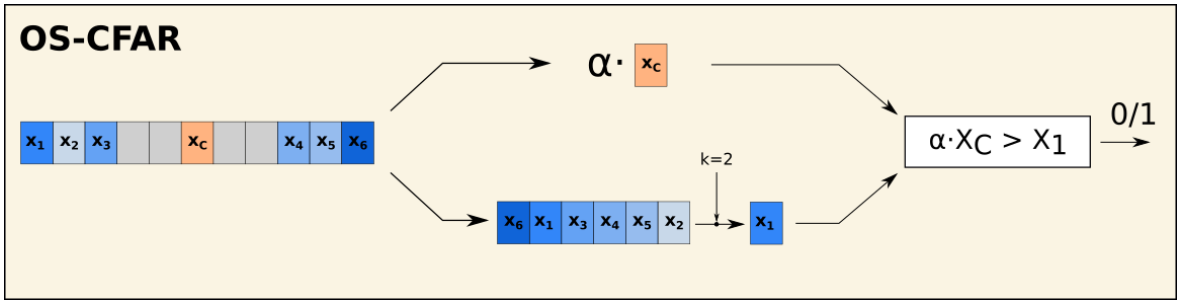
The tuning of the OS-CFAR includes the selection of k. A typical rule of thumb is to set $k = 3/4N_N$.

When compared with each other, the CA-CFAR is easier to tune and cheaper to implement, whereas the OS-CFAR is generally more robust against varying scenarios. The CA-CFAR gives good results when additive Gaussian distributions approximate the noise in the data well. The usage of guard cells is more critical for this algorithm, as the high intensities of the neighbour cells around a target significantly impact the average of P_N. For the same reason, the padding technique must be carefully selected to obtain a good result near the edges. On the other hand, the OS-CFAR is more robust against noise distributions different than additive Gaussian noise, e.g. Rayleigh, Weibull, or log-normal [JYB16]. This includes outliers and artifacts introduced by the sensor electronics. However, the tuning includes an extra term, the k constant, and it is generally advised to perform a logarithmic conversion to the input before processing it.

The implementation of CFAR algorithms for multidimensional radar pipelines typically includes variations of the CA-CFAR and OS-CFAR. For example, combining both by applying the OS-CFAR to the dimensions with the most complex noise and the CA-CFAR to the other dimensions.



(a)



(b)

Figure 2.7: Processing flow of the (a) CA-CFAR and (b) OS-CFAR algorithms, with $N_G = 4$ and $N_N = 6$. The sliding window on the left contains the cell-under-test with a value X_C, the neighbour cell with values $X_N = (X_1, X_2 ... X_6)$, and the guard cells X_G, represented in orange, blue, and grey, respectively. The algorithms fetch the neighbour cells and compute the average value μ_x and the kth largest value X_k, respectively. In parallel, X_C is modified with the scaling factor α. If the scaled X_C is larger than μ_x or X_k, the algorithm yields 1.

2.6 Clustering

Clustering is the process of grouping the point cloud formed by the object-detection algorithm into clusters. A clustering algorithm assigns a single label to each point in the input map, where points belonging to the same group share the same label. Besides the labels for the groups, these algorithms typically include a label for the background and another for the noise. The output of an ideal clustering algorithm consists of a single cluster for each target in the scene, and the spatial location of the clusters corresponds to the location of the targets they represent. Clustering algorithms are classified into partitioning algorithms, where the number of clusters is decided beforehand; and hierarchical algorithms, which organize clusters in a tree structure with an undetermined number of nodes. Even though the former offers higher computational and memory efficiency, they are inadequate for automotive radar processing as these sensors deal with unknown scenarios with a dynamic number of objects around them.

One of the most popular hierarchical clustering algorithms is *density-based spatial clustering of applications with noise* (DBSCAN) [Est+96]. DBSCAN is a density-based algorithm, i.e., it iterates over all points and computes the density around them as the number of neighbouring points, where two points are neighbours if they lie within a distance ϵ. DBSCAN considers the points with a density higher than a given threshold min_pts as core points. All density-reachable core points form a single cluster together with non-core points that are density-reachable to them. The non-core points that do not belong to a cluster are considered noise or outliers. algorithm 1 describes the algorithm in more detail. The pseudocode makes no assumptions about the input size or dimensions, i.e., the *point* variables are objects with an integer defining the label and an n-dimensional vector defining its coordinates.

An alternative algorithm for clustering is DENCLUE [HK98]. DENCLUE is a clustering algorithm with similar complexity to DBSCAN that creates a density map of the input space. The main difference lies in DENCLUE calculating the density gradient afterwards and performing a hill-climbing procedure for connecting points with a low-gradient path. When comparing both, DENCLUE shows small benefits in terms of efficiency, but its tuning is more complicated to generalize for adapting to changing environments.

Algorithm 1 DBSCAN algorithm computation

```
function DBSCAN(X, ϵ, min_pts)
    n_clusters ← 0
    detections ← X = 1
    for each point in detections do
        neighbours ← []
        _detections ← detections − point                ▷ Detections excluding point
        for each _point in _detections do
            if point − _point < ϵ then
                neighbours.add(_point)
            end if
        end for
        if len(neighbours) > min_pts then               ▷ Point is a core point
            if point.label = −1 then                    ▷ If unlabeled, create new cluster
                n_clusters ← n_clusters + 1
                point.label ← n_clusters
            end if
            for each n_point in neighbours do
                n_point.label ← point.label             ▷ Label all neighbours together
            end for
        else
            point.label ← 0                             ▷ Point is a non-core point
        end if
    end for
end function
```

3

Neuromorphic engineering and computing

The fast improvement of silicon performance predicted by Gordon Moore has been anticipated to decline over the past 20 years ago, due to the physical limits of the silicon itself. The massive demand for more powerful and efficient electronic systems by the market and society has been forcing the industry and the scientific community to look for alternative ways of improving the performance of computing systems. This motivation has led to the term "More-than-Moore", which represents the recent trend of improving computing power by means that lie outside of Moore's law. On the one hand, there is an active effort to improve the computing power from *the top*, i.e., software, algorithms, and hardware architectures [Lei+20]. On the other hand, new materials and technologies are emerging for replacing *the bottom*, i.e., traditional silicon wafers. This is the case of optics, organic materials, or novel silicon packaging approaches [Iñi23].

Neuromorphic engineering is an emerging field that is inspired by the brain for creating more efficient computing systems. Its motivation is dual, as it aims to get more insights on the working principles of the brain, while using these new findings for creating more efficient computing systems [Mar+20]. Based on the classification above, the main focus of neuromorphic engineering is to achieve computing improvements from the top, by using existing semiconductor technologies for developing novel system architectures and computing paradigms, which are based on the event-based, asynchronous, and highly parallel nature of the information processing in the brain [Mar+20]. Moreover, neuromorphic engineering seeks to benefit as well from advances from the bottom with approaches like memristors, photonics, or spintronic devices [Wan+23].

This chapter overviews the fundamentals of neuromorphic engineering and computing. The first section underlines the aspects of the brain that have the greatest impact on neuromorphic research. It continues with a description of the topics in neuromorphic research that are more closely related to this thesis. The last sections of this chapter describe the key technologies for implementing neuromorphic algorithms, focusing on hardware platforms and silicon neurons. Interested readers can get an extended overview of current research in neuromorphic engineering and computing in [FBI21], [Chr+22], and [Sch+22].

3.1 Biology fundamentals

Biology is the primary source of inspiration for developing novel neuromorphic systems. Neuromorphic hardware is based on the decentralized and asynchronous processing of spikes in

the brain, and neuromorphic computing algorithms are inspired by the computational models that describe how neurons'state evolve over time and what are the main factors that determine their firing [RJP19]. Therefore, understanding how the brain works is crucial for developing efficient and fast neuromorphic applications.

Depending on the scope and target of their work, researchers analyse the brain from different scales. This multiscale brain organisation is typically split into micro-, meso-, and macro- scales. The *microscale* spans from genetic and molecular structures to the morphology and dynamics of individual cells. The *mesoscale* deals with the connectivity and activity of microcircuits formed by multiple neurons inside one region. *Macroscale* research uses ensemble recordings for creating representations of brain areas and functions that help explain behaviour and cognition using abstract models [DJ22; HSK19]. Modern research on the function and structure of neural networks in the brain started with the work from Ramón y Cajal, who first hypothesized the existence of individual cells, the neurons, that are in charge of the communication and computing in the brain, and provided detailed drawings of the connection trees between neurons [Ram04; Chk04].

The design of neuromorphic hardware and computing algorithms is inspired mainly by the research on the micro- and meso-scales. Namely, in the organization of the biological neurons in the brain and the computational theory that models their behaviour, that includes the encoding and decoding of information, the modelling of neuronal dynamics, or the learning of the synaptic connection weights.

3.1.1 Neuron dynamics

Neurons are cells specialized in processing information and controlling the different mechanisms in the body. The anatomical structure of neurons is typically divided into the *soma*, which is the body of the neuron and is responsible for centralizing and processing information; the *dendrites*, which collect the information arriving at the cell; and the *axon*, that propagates information to the cells that are connected to it. The connection between dendrites and axons typically occurs via small dendritic protrusions called *spines* [Ram04]. Some researchers believe that the anatomical parts of the neuron play a role in the signal processing, i.e., dendrites, axons, and spines may perform non-linear operations, frequency-dependent filters, and compartment filters, respectively [Chk04; HA16].

Neurons process information by modulating the electrical voltage of their membrane. This electrical modulation happens by altering the chemical composition of the cell via the exchange of ions between the cell body and the environment, mainly potassium (K^+), sodium (Na^+), and calcium (Ca^{+2}) [MEL03]. The permeability of ions through the neuron membrane is regulated by opening and closing ion gates that allow only specific ions to pass through. The neuron's membrane voltage is the main parameter that determines the ion gates' state. Some gates are normally open and close when the membrane charges positively, while others are normally closed and open at high membrane voltages. When the membrane voltage reaches a certain threshold, a reaction called *action potential (AP)* originates. The AP starts with a sudden increase in the electric voltage called *depolarization* that propagates through the neuron and is sensed by the connected neurons. Afterwards, the neuron returns to its resting potential through a stage known as *repolarization* and a subsequent stage where the voltage overshoots below the resting potential called *hyperpolarization*. During the repolarization and hyperpolarization, the neuron is in a *refractory state* and is not sensitive to new incoming spikes.

The dynamics model of a neuron is the set of equations that describe the evolution of the neuron's state over time. The models explaining neuron dynamics are varied, offering different levels of abstraction depending on the studied properties or phenomena. The au-

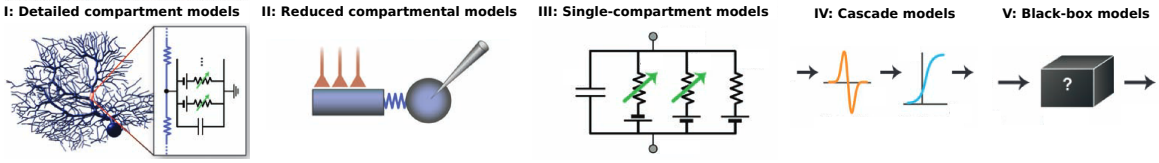


Figure 3.1: The five levels of abstraction for neuron dynamics models. The level of abstraction increases from left to right. From [Her+06]. Reprinted with permission from AAAS.

thors in [Her+06] classify neuron models on the five different levels of abstraction depicted in fig. 3.1, ranging from *detailed compartmental models* that represent neurons with up to 1000 compartments, to *black-box models* that model the output as a function of the input regardless of the internal structure. The rest of this section focuses on the third level, *single-compartment models*, which ignores the neuron's morphology, representing it as a point in space, and focuses entirely on the mathematical description of the spike generation process. These models describe how much the membrane potential increases after incoming spikes, how fast it returns to the resting voltage, which additional factors influence the spike generation, and which spike pattern will be produced by the neuron after getting excited. Single-compartment models are typically represented as electric circuits due to the electric nature of the neuron and the existence of electric components that can reproduce the behaviour described by the models. Hence, they inspire neuromorphic systems that aim to simulate accurate temporal dynamics in a physical system and are simple enough to scale to large populations of neurons. In general, the choice of a model is a trade-off between its computational cost and its accuracy and fidelity to biology.

The *Hodgkin-Huxley* (HH) model was introduced in 1952, and the authors obtained the Nobel Prize eleven years later thanks to it. This model was a significant breakthrough due to its insight into understanding how neuron signalling works [Ger+14, Chapter 2.2]. The model includes a capacitor C that represents the charge of the neuron membrane and several resistors that represent the inverse of the membrane conductance for the different ions that the neuron exchanges with the environment (mainly potassium, K, and sodium, Na). One last resistor models the membrane leak conductance. The ion conductance is explained by ion channels that allow the flow of specific ions through the membrane. The permeability of the ion channels varies over time depending on the current state of the membrane. This change in the permeability explains the shape of the AP. Every time the neuron receives spikes, its membrane potential temporally increases. While this potential stays below a threshold voltage, the membrane returns to its resting potential. A depolarisation stage starts if the threshold is reached, generating the AP. After the AP, there is a sharp decrease of the membrane voltage followed by a refractory period during which the neuron's membrane is less sensitive to incoming spikes.

The *leaky integrate-and-fire* (LIF) model is probably the most used dynamics model for neuron simulations due to its simplicity and accuracy in describing neuron charging behaviour [Lap07; Abb99]. It models the neuron membrane as an R-C electric circuit. Same as in the HH model, a capacitor represents the membrane charge, which decreases slowly over time. The main difference with the HH model is that LIF models the flow of ions with only one resistor, making the model cheap to compute. This simplification does not allow us to simulate the depolarization and hyperpolarization stages of the AP. To overcome this limitation, the implementation of the LIF model typically includes a second equation that forces a trigger of the AP and manually resets the membrane voltage to the resting potential. A consequence of this limitation is that the LIF model cannot model complex firing patterns. appendix B includes a formal description of the LIF differential equation, as well as its solution for some typical cases.

There is a plethora of neuron dynamics models besides the two just described. The *FitzHugh-Nagumo* model [Fit61] and the *Morris-Lecar* model [ML81] aim to provide simplified alternatives to the HH model while keeping its basic properties of excitation and propagation. They provide each a two-dimensional equation system for explaining the neuron membrane charging and the spike generation process. The former is a mathematical work aimed at simplifying the HH model, while the latter was the result of an empirical analysis of the dynamics of the Pacific Barnacle neural system. The model proposed by Izhikevich in [Izh03] is a middle point between the HH and the LIF models. By using two differential equations, it is able to describe different firing patterns observed in real neurons. The authors in [BG05] proposed the *adaptive exponential integrate-and-fire* model, abbreviated *AdEx*, as an iteration of the previous model. In this model, a first equation describes the exponential relationship between the neuron activation with the membrane voltage, and a second equation introduces a dependency of the voltage on an adaptation variable. Izhikevich also proposed the *resonate-and-fire* neuron model in [Izh01], where the excitation depends on the presence of periodic patterns in the input that repeat with a specific resonating frequency.

3.1.2 Neuron connectivity

Synapses are the connections between neurons. The main elements of a synapse are the *pre-synaptic* neuron, which emits information, the *post-synaptic* neuron, which receives information, and the gap between them. Spikes generated in the pre-synaptic neuron are transmitted to the post-synaptic neuron through the gap by either transmitting an electrical pulse (electrical synapse) or by releasing neurotransmitters that stick to the post-synaptic neuron and allow the flow of ions through its membrane (chemical synapse). Synapses are classified into excitatory or inhibitory, depending on whether the post-synaptic neuron's membrane potential increases or decreases after receiving a spike from the pre-synaptic neuron.

Neuron connections in the brain form complex branching structures that add up to 60% of the brain matter [Chk04]. The number and dimensions of the connectivity trees vary between the different neuron classes. A solid hypothesis that explains the shape and density of the dendritic and axonal trees is the *wiring cost* optimization, i.e., evolution finds a structure that provides a desired function by minimizing the wiring cost of the resulting neural network [Chk04; BS12]. The simulations provided in [Chk04] show that the organization of axons and dendrites in arbours reduce the connectivity volume compared to point-to-point connections. The same simulations show that dendritic spines also reduce the size of the branching network. A complementary hypothesis for the existence of complex dendritic trees suggests that the distance of a synapse to the neuron body affects neuron activation. According to this hypothesis, the structure of the dendritic tree can provide complex functionality to neurons. A typical neuron in the neocortex has thousands of inputs, where the majority are distal (i.e., distant from the soma), and a smaller number are proximal (i.e., close to the soma). While the latter synapses have a significant influence on the neuron by easily triggering an AP, and thus define the receptive field, the former have a lesser influence, creating a small depolarization in the neuron body, and serve as predictors for specific patterns in the input [HA16].

When analyzing the connectivity pattern of neurons at a mesoscale, it is possible to observe canonical patterns that emerge across the different regions of the neocortex. The neocortex is organized in *columns* and *layers*, and information flows mostly forward and in parallel [Bri20]. Neurons of the same layer are also connected via *recurrent* connections: neurons within a cortical column are densely interconnected via *horizontal* connections, and populations of different columns are sparsely connected via *lateral* connections. The information in the neocortex typically propagates throughout the different layers in the form of *feedforward*

connections, modeling the world with increasing levels of abstraction and adding new properties to the detected objects [Bri20]. They integrate input from a subset of the sensory space and models the prediction of the data for that specific subset [HA16]. The information in the neocortex also travels from the higher level layers to the lower level layers via *backward* connections. Backward connections are weaker, slower, and less efficient than feedforward connections, so the main hypothesis is that their function is modulatory rather than driving inference in the brain. Empirical evidence supports their role as a regulator of the timing of neuronal responses for the feedforward signaling. Other hypothesis also consider that backward connections have a role for attention mechanisms and communicating context to early layers [Bri20]. Similar to the cortex, the cerebellum is structured in feedforward, excitatory pathways. These pathways are combined with recurrent feedback loops that are typical formed by inhibitory connections. One specific type of recurrent connection is mutual inhibitions across neurons of the same type in the same neural layer. The connectivity is generally very rich and diverse, which explains the plethora of functions and characteristics that the brain and cerebellum can perform [DLR21].

The potential connectivity of a neuron with other neurons in its vicinity is 100%, i.e., a neuron can potentially connect to all its neighbours. However, the actual connectivity sparsity for neocortical pyramidal cells oscillates between 0.01 to 0.1, and the mechanism for modulating the connectivity is the creation and pruning of dendritic spines [Hol+03].

3.1.3 Sensory pathways

One of the richest fields of research in computational neuroscience is the study of the senses and how the brain processes the data they generate. The efficient processing of the data in the brain is the result of millions of years of evolution and is due to an efficient combination of the aspects discussed in the previous sections. Senses transform external stimuli into neural signals through specialized cells like retinal photoreceptors or cochlear hair cells. These signals travel through different anatomical areas of the nervous system and follow different paths until reaching the brain's neocortex. Sensory nerve fibres typically end in the thalamus, where information is then radiated to the 4th layer of different areas in the neocortex [Rho+20]. In the case of visual input, the processing of information starts in the primary visual cortex or V1. Neurons in V1 are responsive to specific features of visual stimuli, such as orientation, direction, and spatial frequency. Information processed in the primary visual cortex is then sent to higher visual areas in the brain for further analysis and interpretation, ultimately leading to the perception of visual scenes, objects, and motion [BP12]. The information flow follows specific directions through the neocortex to decode features and perform particular tasks [RJP19]. In the case of the visual paths, it was assumed for many years that the processing of visual information in the brain was split into two different pathways in the brain: the dorsal pathway, responsible for the location of visual objects and actions related to them, and the ventral pathway, responsible for the identification of visual objects. Typically, these two pathways are understood to answer the 'where', 'what', and 'how' in the visual scene. In the last years, researchers have found evidence of the existence of a third pathway, which would be located in the *superior temporal sulcus* (STS) [PU21]. This third pathway would be responsible for identifying the dynamic aspects of social interaction. This would not be limited to body and face movements but would also integrate other sensory information, mainly from the auditory cortex. Therefore, this region would perform the audiovisual integration of speech by integrating dynamic vision information and human voice.

3.1.4 Neural code

To transmit information in the brain, neurons encode and decode information as spike trains. Classic approaches for modelling spikes assume that information is stored in the existence of an AP, which is an "all-or-none" event, i.e., they are binary events, so the specific amplitude of the electric pulse is irrelevant as long as a pulse happens [Rie+99]. According to this view, the axon of a neuron performs an analogue-to-digital encoding, as it encodes the analogue value of the neuron's membrane voltage into a digital value, which is the occurrence of a spike [CH06]. Alternatively, some *In vivo* experiments refute this hypothesis, as they show that the shape of the AP varies in terms of the amplitude and width of the pulse [ZD19; CH06]. One source of the AP shape modulation is the inactivation of voltage-based ion gates during spike bursts, so consecutive APs decrease in amplitude and increase the spike width. These mechanisms could explain the increase of short-term synaptic plasticity during bursting events in the neurons. A second source of AP shape variations is neuromodulation, i.e., subthreshold membrane modifications via neuromodulators alter the spike shape and lead to an increase in the synaptic transmission. A last source of AP shape modulation is the varying density of ion channels across the axonal branch, so the AP transmission is stronger in different sections of the axon.

An encoding model can be formally defined as the probability of observing a response R when presenting a stimulus S, $p(R|S)$ [Sta13]. Literature often classifies encoding techniques according to the nature of this response, based on whether the information is encoded in the spike rate or in the precise timing of the spikes [DA01, Chapter 1]. For many years, the former was the de facto technique for modelling spikes [Bia+89; Kay+09; Kim+18]. From an experimental point of view, popular scanning techniques like magnetic resonance imaging are designed to record neuron activity over long time windows, rather than measuring the time of individual spikes. Computationally, the modelling and validation of the neural code are more straightforward when assuming rate encoding, as real numbers in a frame-based approach can represent the neuron activity, i.e., a single real number can represent the spike rate of a neuron or group of neurons. Physiological studies have found evidence over the last decades of the correlation between spike rates and certain variables encoded in the brain like the intensity of visual [CF00] and auditive [Pas+12] stimuli, or the force applied by muscles [Blu+17].

Proponents opposing rate coding argue that it fails to explain the fast speed and high energy efficiency of information transmission in the brain. A fast encoding paradigm is necessary for matching the reaction speed to certain visual stimuli like the recognition of other human faces. This would not be possible if neurons throughout the visual pathways had to integrate spike rates over time [Rie+99; Mar+18]. Temporal encoding approaches represent information through the spike times. Hence, the precise timing of a spike can define a variable or internal state. This approach maximizes the energy and time consumed to create the spike train. Even though early investigations of neural code focused on spike rates, the temporal component of the neural code was already possible to observe in early experiments [AM27]. Further experimental data shows that there is a correlation between specific sensed variables and the spike timing of neurons in the sensory pathways [GM08; BK96; Rei+00; Bra+12; Kay+09]. This is also the case for the cerebellum, which can react to sensory stimuli with very high temporal precision [DLR21]. The classification of time encoding models depends on the event or signal used as a reference for measuring the spike times. *Phase encoding* (PE) models encode information in the time difference between the spike and a global reference signal, such as the stimulus onset [GM08] or the phase relative to neural oscillations [KGV14; Kay+09]. *Inter-spike interval* (ISI) and *temporal contrast* (TC) models do not require a global reference signal, as they encode the information in the time difference of

the consecutive spikes that emanate from the neuron itself. The former encodes an absolute value in the time difference between consecutive spikes. In contrast, the latter produces a spike when the relative difference in the stimulus intensity exceeds a certain threshold. Literature suggests that the encoding type is application-specific, so different regions and layers in the brain probably use the code that processes information most efficiently. For example, there is strong evidence that the ganglion cells in the visual system encode the stimulus intensity by using PE, taking as reference the onset produced by the saccades in the retina, which are fast movements of the eye happening several times per second. The following neural layers encode information using ISI encoding, i.e., they use the time difference in the spikes of the ganglion cells for generating structural information like edges and contours. Statistical analysis shows that the noise in the latency across neurons is positively correlated, showing that ISI encoding in the ganglion cells is robust to noise [GM08].

Besides using a neural code for transmitting information, the brain is heavily influenced by the existence of *neural oscillations* or brainwaves [KGV14; Kay+09]. Neural oscillations result from aggregating the synchronized and periodic fluctuation of the membrane voltage of large populations of neurons. These waves serve as an attention mechanism in the sensory pathways, as the high excitability state corresponds to the times when relevant information is present in the sensory input. The timing of spikes relative to these oscillations is thus an effective method of encoding information [Kay+09]. The Dynamic Attending Theory argues that regular patterns in external stimuli drive the phase shift of brain oscillations to facilitate the attention toward those patterns [HG18]. Experiments show that the phase of brainwaves shifts to predict the occurrence of sensory events, so the oscillations are entrained to external rhythms [KGV14]. In other words, neural oscillations anticipate or predict the existence of sensory events.

Population code

Neural population coding refers to how groups of neurons work together to encode and process information. Research highlights that the interactions and correlations among neurons influence neural population properties. *Correlation coding* originates in neuroscience as a hypothesis stating that neurons in the brain transmit the information as a result of their synchronized excitation [deC98], i.e., according to this hypothesis, neurons work as *coincidence detectors* and information is available from the simultaneous spiking of neighbouring neurons. An alternative hypothesis is that neurons operate independently and are able to pass forward information without the synchronized intervention of additional neurons, i.e., neurons work as *integrators* of the spikes arriving at all their inputs. This hypothesis does not deny the existence of correlation in the spiking of different neurons, but it does not see it as a factor for filtering and selecting information. Another alternative is having neurons with an intermediate operation mode, i.e., they can achieve excitement with synchronous or asynchronous input spikes [Rat+13]. The authors in [Rat+13] distinguish *synchrony* from *rate-modulation*, depending on whether the correlation is assessed for spikes taking place inside narrow time windows that do not allow more than one spike per neuron, or inside broad time windows that let evaluate the cross-correlation of the spike rates of the input neurons.

The concept of synchrony is highly related to Hebbian learning. Namely, *synaptic timing-dependent plasticity* (STDP) learning methods modulate the weights between neurons based on the relative time between their spikes, facilitating the connections between neurons that spike at similar times. This topic is explained in more detail in section 3.1.5.

Towards a universal encoding paradigm

As exposed earlier, current hypotheses for explaining the brain's neural code are typically split into rate encoding and temporal encoding models. From an information theory perspective, these two models are not mutually exclusive [BT99]. Dynamic stimuli provoke changes in the instantaneous spike rate of neurons. At the same time, studies show a high temporal precision in the timing of the spikes, so every spike matters and temporal code can describe the stimuli, too. In other words, neurons can respond to stronger stimuli with faster spikes, which would turn into faster spike rates if the following neurons "listen" to the spike trains for longer times [Kay+09]. Such a universal code would explain the robustness of neurons to noise and stochastic patterns inherent to rate coding, as well as the brain's fast and efficient reaction time for many signal-processing tasks.

For example, the large dynamic range in the visual cortex in the brain for the perception of colour seems to be only realizable by using spike rates and long integration windows. However, the processing of shapes and detection of critical objects and situations, like the presence of predators or a sudden collision risk, demand a very fast processing chain that could hardly involve more than a few spikes per neural layer.

3.1.5 Learning

Learning is an optimization process that transforms the structure of neural networks in the brain to adjust to changing conditions. This change in conditions can be due to new experiences and interactions with the external environment or to internal processes in the brain. One of the most relevant neuronal mechanisms for learning is *synaptic plasticity*, i.e., the modification of the strength of the connections between neurons. An optimization method is classified as *supervised learning* when it modifies a neural network during a training stage to minimize an error between the final output and a target value that is known beforehand. On the other hand, *unsupervised learning* refers to methods that optimize the neural networks without using reference output values to compare with. Their main principle is to group input consisting of similar patterns active at similar times [DA01, Chapter 8]. Other popular learning paradigms are *semi-supervised learning*, a hybrid that uses both supervised and unsupervised learning, *reinforcement learning*, based on the existence of rewards that drive the optimization process, or learning through *evolution*, where new individuals randomize their features and best-suited ones propagate over time. Many theories explain from a computational perspective how to achieve synaptic plasticity. Some learning rules are highly biologically inspired, whereas others take a vague inspiration from biology and focus on optimizing a computation problem. For learning rules to be biologically plausible, they need to follow some fundamental properties:

- Locality: A synapse only has access to information on its vicinity and may only use this information for updating its weights.
- Saturation: The weight of a synapse has to be limited, otherwise they could grow towards infinity and the learning process would be unstable.
- Competition: The pre-synaptic neurons connected to a neuron need to compete for its excitation. In other words, the input weights to a neuron may not converge to the same value. This property grants the selectivity of the information arriving at the neuron so that it can filter out irrelevant input.

Hebb's rule is one of the most accepted theories for learning in the brain. Hebb's rule conjectures that if a pre-synaptic neuron A is connected to a post-synaptic neuron B, when a

neuron B repeatedly fires after neuron A, the connection from A to B strengthens. This is the basis of activity-dependent *synaptic plasticity*, which is believed to be one of the main mechanisms behind learning and memory [DA01]. It is also a plausible model for representing unsupervised learning, as the modification of the neural structures does not require labels nor a target output to achieve. The rule was later generalized to include as well the weakening of the connections in case the firing sequence is inverted (B fires before A). The increase or decrease of the synaptic strength is normally referred as *potentiation* and *depression*, respectively. When plasticity persist over time, it is called *long-term potentiation* or *long-term depression*, depending on the sign of the weight change. On the most basic form of Hebb's rule, the synaptic weights evolve based on the correlation of the spike rates in the input.

Scientists have proposed alternative learning rules that follow the basic principles of Hebbian learning. *Oja's* and *BCM* rules are more evolved versions of Hebb's rule. The former normalizes the computation of the input spike rates with the value of the output spike rate so it learns a principal component of the input data. The latter introduces a moving threshold that is compared with the input spike rates. This threshold increases or decreases when the activity is high or low, respectively. STDP is a learning rule that updates the weights based on the correlation in the precise timing of the pre-synaptic and post-synaptic neurons. Contrary to the aforementioned learning rules, STDP assumes that neurons transmit information on the precise timing of spikes. Its compatibility with time encoding makes it interesting for engineering problems, and the research on this type of learning is very active in recent years [SH22; Khe+18]. appendix A contains a mathematical description of all these learning models.

Experimental data has shown that the learning in the brain is more complex process than originally thought, including forms of learning other than synaptic plasticity. One example is *intrinsic plasticity*, a learning process that involves the change of the electrical properties of neurons as a response to different activity levels [DLR21]. Moreover, the brain may also achieve learning by changing its connectivity pattern, by creating and pruning the connections between neurons [Hol+03].

3.2 Spike encoding in neuromorphic applications

As stated in section 3.1.4, we can define encoding in a general way as the probability of obtaining a response R given an input i. Specifically, the neural code responds to the sensed data as spike trains. The format and dimensionality of the encoded data are application-specific. Information is represented by N-dimensional real-valued variables, typically in the form of digital samples or analogue signals. Choosing an adequate spike encoding is crucial for neuromorphic applications, as it directly impacts most performance metrics, e.g., sparsity, energy efficiency, latency, or accuracy. It is also important to distinguish encoding from decoding, as the latter refers to estimating the original stimuli from spike trains. The term encoding is often used loosely in literature, and it may refer to different concepts depending on the authors, field, or specific paper. Therefore, some works focus on phenomenological models that explain how single neurons encode spikes [HSS16], on filtering models for improving the performance of a group of neurons based on their correlation or statistical distribution [deC98], or on decoding models that reconstruct the stimulus encoded by neurons [PKK19]. In general, review papers do not address this aspect and either classify encoding techniques belonging to one specific group [PKK19], or group together the aforementioned models without putting the focus on the different nature of the models [For+22; Aug+21b]. Prior to classifying spike encoding, let us first provide a clear definition of what spike encoding is.

From a systems perspective, research typically approaches the topic of encoding by answering one of the two following questions:

1. For a given neuron, what is the relationship between its output spike train $s(t)$ and its input $i(t)$?

2. For a given population of neurons with N_I inputs and N_O outputs, what is the relationship between the output spike trains $s_O(t) = \{s_1(t), s_2(t), \ldots, s_{N_O}(t)\}$, and the received inputs $i_I(t) = \{i_1(t), i_2(t), \ldots, i_{N_I}(t)\}$?

Note that the first question answers how an individual neuron encodes input signals into spikes, which is a direct consequence of the neuron's dynamics model. For the rest of this work, let us refer to the computing techniques that address this question as *spike encoding*. The second question focuses on how groups of neurons provide collective outputs in response to an arbitrary number of inputs. The response depends not only on the dynamics of the individual neurons, but also on the collective response that emerges from the behaviour of all neurons in a population of neurons. Let us refer to the aggregated result of all these processes, collectively addressing the second question, as the *neural code*. In other words, spike encoding will refer to the models that encode input signals to spikes, and the neural code emerges from the global output of multiple encoders according to a population dynamics model.

Regarding how data is fed to encoders, we can distinguish encoding models that deal with frames of static data from models that process information from time-varying signals. Neuromorphic computing research has extensively studied the former group, as many computer science solutions (especially those using *artificial neural network*s (ANNs)) analyse static data (e.g., image frames), even if they belong to more extensive temporal sequences like video sequences. Researchers often propose different methods that convert the structure and parameters of the ANNs, encode the digital data frame-wise into spike trains and then feed them to the converted *spiking neural network*s (SNNs). These works perform exhaustive measurements of key performance indicators like precision across many data samples. However, the capacity of the models to work in real-time is usually disregarded, which is a crucial aspect for deploying SNNs in signal processing applications.

The most common categorization is to split encoding techniques into rate and temporal encoding techniques, where the former encodes data into the spike count in a time window, and the latter encodes data into the time of the specific spikes. Some literature includes as well population encoding as a third category. However, as pointed out in [Aug+21b], population encoding is a generalization of rate encoding. Moreover, in the view of some authors in the domain of neurobiology, rate encoding is a generalization of temporal coding [Bre15; Rie+99]. Their main point is that neuron dynamics produce single spikes at specific times, and rate encoding emerges as an integration of these dynamics over time. Based on this argumentation, this section distinguishes between *neural encoders* and *population dynamics* (see fig. 3.2). The encoders are the point processes that collect data and generate spikes over time, i.e., this section assumes that encoders always produce time-coded spikes. On the other hand, population dynamics includes the different processing approaches performed on the spike trains generated by individual encoders for generating more sophisticated information. In other words, an encoder circuit implements a dynamics model that produces spikes over time. The filtering and processing of these spike trains by SNNs may involve the notion of spike rates, but only as a result of processing the already encoded signal. Population dynamics techniques deal with the integration of information over time (rate coding), increasing the sparsity (N-of-M coding), combining all spike trains into a main stream of information (population coding), or finding patterns in the relationship between the different spike trains

(rank-order coding, correlation coding). The output dimensionality of these approaches depends on the specific process. Typically, it is not bigger than the input dimensionality. A common goal of population dynamics techniques is to create emergence, i.e., they extract information that is not possible to obtain from single encoder streams, by combining the data encoded by each of them.

According to the definition above, for reacting to a time-varying analogue signal, neuromorphic solutions can only use temporal encoding. Encoders measure the delay between spikes and reference points in time. As neurons do not have a notion of absolute time or a global clock, spike times need to be referenced to local events. From a control perspective, we define an *analogue-to-spike encoder* (ASE) as a processing component that takes a time-varying signal $i(t)$ and produces a spike train as output $s(t)$ by using a transfer function $\mathcal{E}$. In its simplest form, this process takes the form

$$s(t) = \mathcal{E}(i(t)). \tag{3.1}$$

For producing the spike train, some ASEs, need an external reference signal $r(t)$ that provides a time reference and synchronises the different encoders. Alternatively, they can reference the spikes to the previous spike generated. Other ASEs also need to store in memory after spiking the value of a variable $x(t)$ that is later used for generating the following spike. Thus, we turn (3.1) into the more generic form

$$s(t) = \mathcal{E}(i(t), r(t), x(t)). \tag{3.2}$$

Figure 3.2 depicts, on the left the representation of an ASE taking into consideration the aforementioned signals. Based on this representation, we classify spike encoders into three large groups: *phase encoding* (PE), which encodes the input into the spikes using a reference signal, *temporal contrast* (TC), which keeps in memory the input of the previous spike for comparing with the current input, and *inter-spike interval* (ISI) methods, which generate spikes based on the input without using a reference signal or storing information in memory. Table 3.1 summarizes these three types of encoders. Note that this is a simple classification

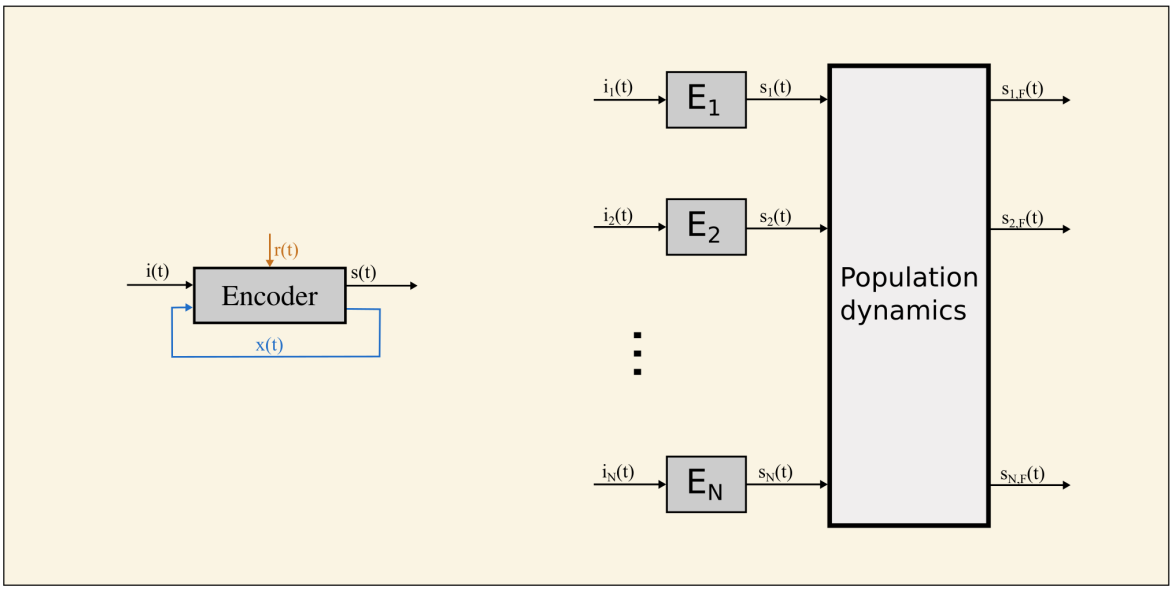


Figure 3.2: On the left, control diagram of a spike encoder block, that takes an input signal $i(t)$ and produces and output spike train $s(t)$. The encoder can use an optional reference signal $r(t)$ and an internal state variable $x(t)$. On the right, diagram of a population of N encoders. The encoders' outputs are filtered according to certain population dynamics that result on up to N filtered spike trains $s_{n,F}(t)$.

that can be further decomposed, as the described encoding techniques have multiple variants. Some authors also refer to the methods implemented by ASEs as neural recording techniques [CI15].

Table 3.1: Classification of analog-to-spike encoders according to their main properties from a systems perspective.

	Reference	Memory	Polarity
TC	NO	YES	Bipolar
PE	YES	NO	Uni-/Bipolar
ISI	NO	NO	Unipolar

3.2.1 Phase encoding / time to first spike

Techniques using PE assume that neurons transmit information in the timing of their spikes relative to a global reference signal [Kay+09]. Similar to PE, *time to first spike* (TTFS) is an encoding technique that assumes information is stored in the timing of the spikes relative to a reference point, $t_s - t_0$ [BF10; GQH07]. The difference between PE and TTFS lies in the nature of the reference signal rather than in the working principle, i.e., PE assumes that the reference is a global periodic wave, and TTFS calculates the spike times relative to the onset of a stimulus or a reset event that typically affects a population or layer of neurons. The signal used as reference for PE is typically a periodic wave with a period T, and the spiking phase α_{PE} is the ratio between the spike time t_s and T,

$$\alpha_{\text{PE}} = 2\pi \frac{t_s}{T}, \tag{3.3}$$

where α_{PE} is expressed in radians. Therefore, t_s and α_{PE} take values in the ranges $[0, T]$ and $[0, 2\pi]$, respectively. In PE techniques, the relationship between the spike phase α_{PE} and the input to the neuron is inversely proportional. In other words, the higher the input, the faster a neuron gets excited and spikes. This property facilitates the implementation of filtering techniques like *winner-takes-all*, where the most excited neurons in a population dominate the meaning of the output. For a neuronal sampling resolution of ΔT, a neuron can distinguish $N_{\text{steps}} = T/\Delta T$ encoding steps inside an encoding cycle. Thus, the number of different combinations N_C that a neuron using PE can encode is

$$N_C = 1 + N_{\text{steps}}. \tag{3.4}$$

Further iterations of this encoding paradigm map data into more than one spike, where the combination of all timings encapsulates the information. In other words, the information is not stored in the number of spikes but in the combination of their occurrence times. For example, the models introduced in [Kim+18] and [SM21] map scalars to several spikes where the contribution of each spike follows an exponential relationship with its timing and all spikes' contributions are added up. The output spike train in these methods is decoded as

$$\hat{y} = \sum_{t}^{N_{\text{steps}}} z(t) 2^{-(1+\alpha_{\text{PE}})}, \tag{3.5}$$

where $\hat{y}$ is the decoded output, and $z(t)$ is a binary function representing the spike train, i.e., it yields one when a spike takes place at time t, and otherwise it stays zero. section 3.3.2 contains more details on the implementation of the model in [SM21].

Empirical results with biological recordings show evidence that the encoding modelled by (3.5) may also be present in the brain [Kay+09]. The number of combinations in models applying this approach increases from (3.4) to

$$N_C = N_{\text{steps}}^2 \,. \tag{3.6}$$

Some literature uses the term *latency encoding* for naming PE techniques [Bra+12; GM08; Khe+18]. However, this term can bring confusion, as other works use latency encoding for referring to the encoding of information in the delay between consecutive spikes, i.e., ISI [Aug+21b]. For clarity, the present work avoids using the term latency encoding to describe any encoding technique.

3.2.2 Temporal contrast

Based on the assumption that sensory cells are sensitive to stimuli that change over time, a TC encoder cell produces spikes when there is a variation in the magnitude of the stimulus and stays silent when the input is static over time. To compare the current input with previous values, the encoder must store a state variable $x(t)$ holding the reference input level in memory and a parameter k_{tc} that sets the minimum change in the input that triggers a spike. For encoding positive and negative changes in the input, TC encoders generate a bipolar output spike train. Hence, a hardware implementation of TC must be able to produce both positive and negative spikes, and the hardware platform processing the spikes must recognize this type of encoding as well. There are different alternatives for implementing a TC encoder depending on how $x(t)$ is handled:

- *Threshold-based representation*, that compares the absolute change between consecutive values against a threshold. The threshold is based on the first derivative of the input signal,
$$x(t) = f(i(t-1)) \,. \tag{3.7}$$
- *Step-forward*, which generates spikes when the actual value exceeds a pre-determined threshold relative to the previous reference value or baseline. This reference value is increased/decreased by the value of the threshold. Therefore, changes in the value several times bigger than the threshold would produce several consecutive spikes.
$$x(t) = \text{const.} \qquad \text{iff} \quad z(t) = 0 \,, \tag{3.8}$$
$$x(t) = f(i(t), x(t-1)) \qquad \text{iff} \quad z(t) = 1 \,. \tag{3.9}$$
- *Moving window*, similar to the previous method, but the reference value is calculated as the average of a certain amount of preceding values.

According to the quantitative results provided in [PKK19], step-forward shows the best performance among the aforementioned TC methods, measured in terms of *root-mean-square error* (RMSE) and signal-to-noise ratio. It performs well against noise and can adapt to sudden changes in the signal's input intensity and long plateaus. Its tuning is trivial, as only the threshold parameter needs to be adjusted. This method fails to adjust to the signal's offset, as spikes are referenced to the initial value.

Step-forward is one of the most popular TC techniques for encoding the intensity of event-based sensors [LPD08; CI15]. The work in [LPD08] has served as a baseline for several implementations of event-based sensors in the last decade. There, the authors proposed an analogue circuit that produces positive or negative spike events, ON and OFF, when the

input signal V_{in} exceeds a positive or negative threshold, V_{ON} and V_{OFF}, respectively. The threshold values are updated after every spike event by adding to the input voltage a positive and negative constant, d_{ON} and d_{OFF}, respectively,

$$ON = 1 \quad \text{iff} \quad V_{\text{in}} > V_{ON} \quad \text{and} \tag{3.10}$$
$$V_{ON} = V_{\text{in}} + d_{ON}\,, \tag{3.11}$$

and

$$OFF = 1 \quad \text{iff} \quad V_{\text{in}} < V_{OFF} \quad \text{and} \tag{3.12}$$
$$V_{OFF} = V_{\text{in}} + d_{OFF}\,, \tag{3.13}$$

where $d_{ON} > 0$ and $d_{OFF} < 0$. Compared to alternative encoding techniques, step-forward offers reduced time latencies, wide dynamic ranges, and an asynchronous spike generation that does not require a global reference.

A critical parameter of a TC circuit is the maximum steepness it can detect in the sensor's output, i.e., its input gradient sensitivity. Signals with very steep curves will trigger the TC encoder very fast, and it is paramount that the implementation can produce and process the spikes.

3.2.3 Inter-spike interval

A neuron using ISI maps the intensity of the input to the time delay between consecutive spikes. ISI is often associated with *bursting*, which refers to the generation of multiple consecutive spikes after the neuron reaches an excitation stage. The burst frequency is proportional to the input intensity and inversely proportional to ISI [For+22]. ISI encoders often exhibit *frequency adaptation* as well, which relates to the gradual decrease of the spike frequency after excitation if the input stays constant [LI09; WD08]. The spiking behaviour depends on the specific dynamics that model the neuron's behaviour. In general, ISI circuits are based on dynamics models with two variables, one representing the membrane voltage and the other the frequency adaptation, like the AdEx model [BG05] or the model from Izhikevich [Izh03]. We can describe the operation of an ISI circuit as

$$C\frac{dV_m}{dt} = I_{\text{in}} + I_{\text{fb}} - I_L - I_{\text{a}}\,, \tag{3.14}$$

where V_m, I_{in}, and I_L are the neuron's membrane voltage, input current, and the leak current, respectively. I_{fb} is a feedback current that sharply increases when V_m approaches the switching voltage. I_{a} is the adaptation current, that slowly increases over time, facilitating the occurrence of consecutive spikes [LI09]. From eq. (3.14), the first spike is proportional to the input current, assuming $I_{\text{a}} = 0$, and the following spike times decrease over time until the burst is over.

The authors in [Izh+03] argue that some biological neurons possess a spike resonating frequency, so they achieve maximum excitation when the incoming spikes have a specific ISI.

3.2.4 Population dynamics

In this work, population dynamics refers to the dynamics models applied to the spike trains of populations of neurons (see fig. 3.2). These models do not focus on the rules for generating spikes at individual neurons. Instead, they process the spikes of neurons to extract meaningful features from output correlations, remove noise, or increase the population sparsity.

Rate and population coding

The basic principle of rate coding is that the information transmitted by neurons is stored in the number of spikes that are sent per time unit, which is why some authors also refer to it as *count rate coding* [Aug+21b]. The neural process that computes the spike rate integrates the received spikes over a certain time window.

The simplest approach for implementing rate code is to map linearly the intensity $i(t)$ of the stimuli to the rate $f(t)$, i.e., to the number of spikes per time unit,

$$f(t) = k_r i(t) + b_r\,, \tag{3.15}$$

where k_r and b_r are a proportionality and offset constants, respectively. If we apply (3.15) to time windows of constant length T and a constant resolution ΔT, the total number of time steps N_{steps} is calculated as

$$N_{\text{steps}} = \frac{T}{\Delta T}\,. \tag{3.16}$$

We can simplify (3.15) by expressing $f(t)$ as the number of spikes per T time units, $\bar{f} = T \cdot f$. The rate $f(t)$ can be easily computed by counting the number of spikes N_s in a time window of length T,

$$f(t) = \frac{N_s}{T}\,. \tag{3.17}$$

The number of different combinations N_C is maximized when the minimum frequency is set to zero, $\bar{f}_{\text{min}} = 0$, $b_r = 0$, and the maximum frequency is one spike per time step, $f_{\text{max}} = N_{\text{steps}}$, which yields

$$N_C = 1 + N_{\text{steps}}\,. \tag{3.18}$$

A drawback of this model is the required waiting time for mapping a single value to a spike train, as the spikes cannot be decoded until the end of the time window T. In general, the sparsity S of a network is reduced when using rate code, as this method requires generating multiple spikes for encoding the input data. Assuming a uniform distribution of data, each input will be represented on average by $N_{\text{steps}}/2$ spikes,

$$S = N_{\text{steps}}/2\,. \tag{3.19}$$

The high value of S introduces an overhead for the interneuron communication system, reducing the throughput and incrementing the required resources.

Rather than excluding the encoding methods described in section 3.2, rate code is compatible with ISI and TC encoding techniques. Some spike train decoders for ISI and TC encoders use the concept of a rate code, i.e., computing the rate from those encoders provides an estimate of the average intensity (ISI) or gradient (TC) over a time window. This is not the case for PE, where encoders always generate one spike per encoding time window, and the rate is thus constant regardless of the input intensity.

In [LPD08], the authors a simple way of computing a spike rate out of a TC encoding for event-based sensors,

$$f(t) = \frac{\text{TC}(t)}{\theta}\,, \tag{3.20}$$

where $\text{TC}(t)$ is the instantaneous temporal contrast and θ is the spiking threshold. (3.20) is agnostic of the spikes' sign. One option is to compute $f(t)$ as the aggregated difference of positive and negative spikes, so θ takes positive and negative values depending on the spike polarity. Another option is to use two rate variables, $f_{\text{ON}}(t)$ and $f_{\text{OFF}}(t)$, that represent the rate of positive and negative rates, respectively.

An alternative for reducing the length T of the integrating time window is to combine several neurons for encoding the same value, typically known as *population coding*. For a population with M neurons with the same firing rate $f(t)$, the total number of combinations is

$$N_C = (N_{\text{steps}} + 1)^M . \tag{3.21}$$

An interesting feature of this approach is its compatibility with the stochastic nature observed in the brain. In engineering applications, this property is useful for processing noisy data. If neurons in a population have a certain probability p of spiking proportional to the value they are encoding, the total number of spikes in a time window T will be

$$f(t, t+T) = p(t) \cdot M \cdot T , \tag{3.22}$$

where t and $t+T$ is the time range for counting the spikes, and p is assumed constant during this period.

Winner-takes-all

In temporal coding schemes, spikes carrying more information happen earlier. Ignoring late spikes allows neurons to process information faster while keeping most information from the input spike train. This is the working principle of *Winner-takes-all*, a technique for focusing all the attention on the first spike to arrive from a neuron population. This means that the post-synaptic neuron ignores all of the spikes after the initial one. This technique is helpful for tasks like peak detection, which focuses on finding the dominant stimulus in an input sequence. This technique allows as many combinations N_C as the number M of spike trains in the population,

$$N_C = M . \tag{3.23}$$

The spike sparsity S introduced by winner-takes-all is defined by

$$S = \frac{1}{M} . \tag{3.24}$$

N-of-M code

N-of-M coding is a generalization of winner-takes-all. Neurons with M input connections listen to only the first N input spikes, where $N < M$ [Fur+04]. N-of-M coding is a generalization of the *winner-takes-all* approach, as the latter would be equivalent to *1-of-M* encoding.

To compute N-of-M coding, a computing block needs to count the spikes produced by the input neuron population and inhibit the transmission of further spikes after the Nth spike arrives. The number of combinations that can be achieved with this technique is given by

$$N_C = \frac{M!}{N!(M-N)!} . \tag{3.25}$$

A shunt circuit that activates after N spikes and bypasses all incoming spikes after activation can implement this behaviour in a physical implementation [Chr+22, Section 21]. The sparsity S obtained after applying N-of-M code to a neurons population is calculated as

$$S = \frac{N}{M} . \tag{3.26}$$

Rank-order code

Rank-order code (ROC) assumes that the relative order in which neurons spike in a population contains stimulus information. This coding method was introduced in [TG98] as a possible explanation for the contrast and intensity invariance of the receptive fields in the visual cortex. Namely, a specific post-synaptic neuron that receives ROC-coded spike trains as input achieves maximum excitation when the input spikes arrive in one particular order. The excitation level decreases proportionally with the level of mismatch between the input spike sequence and the specific sequence the neuron was tuned for. Researchers showed that ROC can explain the encoding of visual stimuli in the human brain. They simulated the processing of different stimuli and reconstructed them with a performance around 70% [BF10].

From a computational perspective, a ROC implementation requires a computing block that reads the incoming spikes and computes the rank function $R(t)$ at time t. $R(t)$ produces values between 0 and 1 proportional to the number of spikes that have already arrived. The activation $A(t)$ of post-synaptic neuron uses $R(t)$ for normalizing the arriving spikes according to the order of occurrence,

$$A(t) = \sum_{m}^{M} w_m R(t) s_m(t), \tag{3.27}$$

where M is the number of pre-synaptic neurons, $s_m(t)$ is the spike train from the mth pre-synaptic neuron, and the weight vector $\overrightarrow{w} = \{w_1, w_2 \dots w_m\}$ follows a distribution that optimizes a specific input sequence. Assuming a linear distribution of $\overrightarrow{w}$ and a number of simulation steps N_{steps} larger or equal to the number of inputs, $N_{\text{steps}} \geq M$, the authors in [TG98] calculate the number of possible combinations as

$$N_C = M!. \tag{3.28}$$

If $N_{\text{steps}} < M$, it is not possible to encode M different values in the input and the number of possible combinations is reduced to $M_C = N_{\text{steps}}!$. As ROC gives an output based on the combination of all the inputs, this technique does not increase the spike sparsity S of the network, $S = 1$.

ROC has some features in common with N-of-M coding. They both work as a filter for spike trains from populations of neurons that use global-referenced spikes, and they serve as an attention mechanism that enhances the sensitivity to the first spikes that arrive to the neuron. Whereas N-of-M coding shifts the attention to the first N spikes in the input sequence, ROC's attention decreases proportionally to the number of spikes that have arrived. Contrary to N-of-M coding, ROC does not increase the temporal spike sparsity, as the number of spikes coming in and out of the ROC computing block are the same. ROC also eliminates any notion of the absolute intensity of the information encoded in the arriving spike trains. Thus, ROC is suitable when the information can be explained on the sequence of arriving stimuli. In contrast, N-of-M coding works best when the most intense stimuli dominate the information content [Chr+22, Section 21].

3.3 ANN to SNN conversion

An attractive application for neuromorphic computing is to provide machine learning algorithms that can be accelerated in neuromorphic chips and reduce the footprint of *artificial intelligence* (AI) systems. A key challenge is the implementation of learning, because spike functions do not have a derivative, which is an essential operation for implementing back-propagation. One possibility is to modify the output function of the neurons to overcome

the non-differentiability issue. The model introduced in [SO18], called SLAYER, convolves the spike output over time with a *spike response* kernel, resulting in a spike response signal that can be differentiated. The model assigns errors over the layers but also over the time iterations. The model has been implemented in CUDA and Loihi. The works in [Mos17] [Com+20] use the relative time between the first spike and the pre-synaptic spike time as the output to differentiate with backpropagation. The neuron voltages of the former are calculated according to an integrate-and-fire neuron model, and the latter on an alpha-synaptic function. Similarly, the work in [KM20] uses backpropagation, assuming that the spike time represents the output of a ReLU function. It uses an integrate-and-fire neuron model and encodes data using TTFS.

Another option for training neural networks in neuromorphic chips is to use an alternative learning paradigm. There is an increasing amount of research exploring unsupervised learning methods, as they can reduce the dependence on massive datasets and the obscurity of the models. One of the most popular unsupervised algorithms for SNNs is STDP [SH21; Khe+18]. STDP involves strengthening a neuron's connections when the corresponding inputs' spikes precede an output spike and weakening them when the inputs' spikes follow the output spike. The temporal nature of this learning mechanism makes it attractive for obtaining efficient neuromorphic applications that can learn temporal patterns and adapt to new data in real time [Lob+20].

Finally, an effective approach for achieving learning with neuromorphic systems is the conversion of ANNs to SNNs. The idea is to learn the network parameters using a classical approach on traditional computing platforms and then map the parameters to an SNN with analogous architecture. In other words, ANN to SNN conversion consists of an offline training stage and a latter inference in a neuromorphic chip [RJP19]. These methods became very popular thanks to their high accuracy results, similar to state-of-the-art ANN results. For many years, the most common approach was mapping real numbers to spike rates [Die+16; HE16; Rue+17; Sen+19]. These methods achieve state-of-the-art accuracy on well-known datasets like MNIST, CIFAR-10 or ImageNet. Moreover, implementing the mapping to spikes is straightforward. However, they require significant inference times per frame, yielding high simulation times and undesired power requirements. Last years have transitioned towards time-based encoding methods for converting ANNs [RL18; SM21].

3.3.1 Rate-based conversion

Converting ANNs to SNNs with spike rates is a reliable approach for the event-based implementation of ANNs's inference. Even though they are not practical due to the high number of spikes they require, which leads to a high power consumption [Dav+18; Dav+21], their understanding is desirable for advancing towards more efficient time-based approaches. The work from [Rue+17] was one of the first widely used models for converting ANNs to SNNs, thanks also to its public availability in an open-source library[1]. It is based on the principle that the spike rate of a neuron is directly proportional to the real number it encodes.

The forward pass of a *deep convolutional neural network* (DCNN) is defined by the activation function of the neurons, $a_i = f_i(x)$, where x_i is the weighted input to the neuron, calculated as

$$x_i = \sum_j w_{ij} a_j + b_i \,, \tag{3.29}$$

where w_{ij} is the synaptic weight between neurons i and j, and b and b_i is the bias constant of neuron i. Let us assume that neurons use a ReLU activation function, which generally serves

[1] https://snntoolbox.readthedocs.io/

as the de facto activation for most networks,

$$f(x) = ReLU(x) = \max\{0, x\}\,. \tag{3.30}$$

The conversion method in [Rue+17] uses an integrate-and-fire neuron model without a leak. The output of a neuron is defined by a spike function $\Theta(t)$, which is a binary function that yields one when there is a spike and zero otherwise. Neurons have an internal state represented by a membrane voltage $u(t)$, updated according to

$$u(t) = u(t-1) + i(t)\,, \tag{3.31}$$

where $i(t)$ is the input current to the neuron. The current depends on the input spikes present at the specific time step,

$$i(t) = \sum_j \bar{w}_{ij}\Theta_j(t) + \bar{b}_i\,, \tag{3.32}$$

where $\bar{w}_{ij}$ and $\bar{b}_i$ are the synaptic weights and bias for the spiking neuron. A neuron spikes when its membrane voltage crosses the value of a constant called threshold voltage u_{th},

$$\Theta_i(t) = \begin{cases} 1 & \text{, if } u(t) > u_{\text{th}} \\ 0 & \text{, otherwise} \end{cases} \tag{3.33}$$

When a spike occurs, the neuron's voltage is reset and starts charging again according to (3.31) and (3.32). The authors scale the neuron parameters $\bar{w_{ij}}$ and $\bar{b_i}$ in (3.32) with the threshold voltage of the neuron,

$$\bar{w}_{ij} = u_{\text{th}} w_{ij} \tag{3.34}$$

and

$$\bar{b}_i = u_{\text{th}} b_i\,. \tag{3.35}$$

By the end of the simulation time, each neuron will have spiked N_s times, which can be averaged over the number of time steps for obtaining the spike rate r_i. The spike rate of a neuron is proportional to the activation of its ANN counterpart, $r_i \propto x_i$, together with a residual error that can be minimised by increasing the number of simulation steps. Note that the number of simulation timesteps limits the spike rate, so the dynamic range is considerably smaller than the original ReLU function. This is depicted in fig. 3.3.

The authors discussed two strategies for the reset operation: *reset to zero* and *reset by subtraction*. The former sets $u(t) = 0$ after a spike, whereas the latter performs the subtraction $u(t) = u(t) - u_{\text{th}}$. They concluded that the reset by subtraction provides better results, as reset to zero always discards a residual value in $u(t)$ after crossing the threshold. They also used analogue data for the input layer of the networks by mapping the real numbers to input currents to the neurons according to

$$i(t) = u_{\text{th}} \left(\sum_i w_i x_i + b \right). \tag{3.36}$$

The authors applied their conversion technique to the MNIST, CIFAR-10, and ImageNet datasets, yielding accuracies close to the original ANNs. For MNIST they provide an implementation on the SpiNNaker chip with the LeNet architecture. They used the VGG-16 architecture for the ImageNet dataset.

Other approaches include the work from Hunsberger and Eliasmith, who proposed an alternative method for converting ANNs into SNNs by using a LIF neuron model and softening the neuron response function [HE16]. The core idea of this work is to replace the rectified linear nonlinearity with a modified LIF nonlinearity, which results in a conversion method

that requires fewer spikes than the work in [Rue+17]. The authors tested the SNN on multiple datasets, including MNIST, CIFAR-10, and ImageNet. They also implemented their work in the Loihi chip. The authors in [Sen+19] provide another rate-based conversion model that effectively processes the CIFAR-10 and ImageNet datasets, improving previous conversion results. The model, named *Spike-Norm*, is based on the independent balancing of the threshold voltage of each neuron based on the maximum value the ReLU function can yield, so the output dynamic range can be optimized (see fig. 3.3). In [Die+16], the authors provide a means of converting recurrent ANNs, taking advantage of time delays in the synaptic connections. They implemented their work on the True North chip, where the weights learnt in the original network had to be discretized to a 4-bit resolution.

3.3.2 Few spikes model

The authors in [SM21] introduced the *few spikes* (FS) neuron model for converting ANNs to SNNs with a focus on temporal sparsity. The neuron uses a temporal encoding technique that maps real numbers to N spikes, $0 \leq N \leq N_{\text{steps}}$, where $N_{\text{steps}} = \text{const.}$ is the length of the encoding time window. The neuron's membrane voltage $u(t)$ is updated on every time step t according to

$$u(t) = u(t-1) - h(t-1)\Theta(t-1), \tag{3.37}$$

where $\Theta(t)$ is the spike output of the neuron, and $h(t)$ is the reset value that is subtracted from the membrane voltage when a spike takes place. The neuron produces spikes when the membrane voltage is larger than a threshold variable u_{th},

$$\Theta(t) = \begin{cases} 1, & u(t) - u_{\text{th}}(t) > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{3.38}$$

Thus, the simulation needs to run for N_{steps} and record all spikes for decoding the output of a neuron. The initial value for the membrane voltage is the input of the neuron, $u(0) = x$. Algorithm 2 shows the logic of an algorithm that implements the FS neuron with a ReLU activation function.

The authors propose a two-stage approach for converting multi-layer networks, where a neuron reads the spikes from the previous layer on the first stage and generates spikes according to (3.37) and (3.38) during the second stage. We can calculate an estimate of the activation value of the neuron, $\hat{y}$, according to the spike train that the neuron generates,

$$\hat{y} = \sum_{t}^{N_{\text{steps}}} d(t)\Theta(t), \tag{3.39}$$

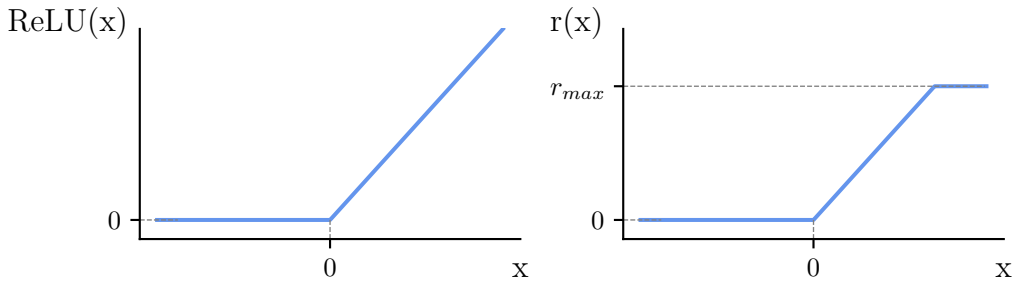


Figure 3.3: Output range of a standard ReLU function on the left, and a spiking integrate-and-fire neuron on the right. The spike rate is bound to the range $[0, r_{\text{max}}]$, which is orders of magnitude smaller than the range provided by floating point precision.

Algorithm 2 Encoding stage of FS neuron simulation for the ReLU activation function

$u(0) \leftarrow x$
$u_{\text{th}}(0) \leftarrow 2^{N_{\text{steps}}}$
$h(0) \leftarrow 2^{N_{\text{steps}}}$
$N \leftarrow n$
for $t = 1 : N_{\text{steps}}$ **do**
  $T(t) \leftarrow \alpha 2^{N_{\text{steps}}-t}$
  $h(t) \leftarrow \alpha 2^{N_{\text{steps}}-t}$
  $u(t) \leftarrow u(t-1) - h(t-1)z(t-1)$
  if $u(t) - u_{\text{th}}(t) > 0$ **then**
    $z(t) \leftarrow 1$
  else
    $z(t) \leftarrow 0$
  end if
end for

where $d(t)$ is the decoding function. Algorithm 3 describes the logic for decoding the spikes from a neuron that replicates the ReLU activation function. The authors optimized the functions $u_{\text{th}}(t)$, $h(t)$, and $d(t)$ for different activation functions using the backpropagation through time learning technique. They assigned larger weights to the loss in the regions that containing the most information and provided examples for the Sigmoid and SiLU activation functions. For the ReLU activation function, they use an analytical approach and fix $u_{\text{th}}(t) = h(t) = d(t) = \alpha 2^{K-t}$, where α defines the maximum input value that can be converted. This parameter set leads to the decoding function described in Algorithm 3.

Algorithm 3 Decoding stage of FS neuron simulation for the ReLU activation function

$a \leftarrow 0$
for $t = 0 : N_{\text{steps}}$ **do**
  $d(t) \leftarrow \alpha 2^{N_{\text{steps}}-t}$
  $a \leftarrow a + z(t)d(t)$
end for
$x \leftarrow w \cdot a$

The FS model is lossless, as no information is disregarded, and highly efficient due to its sparse nature. The authors converted large ANN architectures in the experiments, like EfficientNet and ResNet-50, on the CIFAR-10 and ImageNet datasets. The authors do not provide a replacement for the max pooling operation; instead, they use average pooling. The work does not include an implementation in neuromorphic hardware, so the actual efficiency of the algorithm was not evaluated.

3.4 Frequency domain representation

Computing the frequency spectrum is a cornerstone in many signal processing chains, as information often resides in the frequency components of the signals. Current signal processing techniques for generating the frequency spectrum are robust and well established, being the *Fourier transform* (FT) the most representative algorithm (see section 2.3). Although there is little research into neuromorphic techniques for generating and analyzing the frequency spectrum, there is an increasing interest in developing such techniques [Vog+22]. The repre-

sentation in the frequency domain and its posterior analysis is a specialized task that biology employs in the auditory pathway. Most information in auditive signals is stored in the different properties of the overlapping acoustic that propagate through the auditive senses. This is especially critical in the case of animals that heavily rely on this source of information, such as bats [MS03]. In humans, the frequency components in the sound are separated via the mechanical structure of the cochlea, where increasingly small receptive nerves react to increasingly high sound frequencies.

The frequency domain representation typically occurs at the start of the signal processing pipelines, so it is often computed on the sensor's raw signal. One approach for implementing neuromorphic applications in the frequency space is filtering specific frequencies in the signal. The work in [Jim+16] introduces a spike-based approach for decomposing audio signals into the main frequency components. Another option is using learning techniques to identify relevant frequencies for a specific task [Ben+23]. Alternatively, some authors have explored the *resonate-and-fire* (RF) model introduced in [Izh01], where the neurons' excitation depends on the frequencies present in the input signal [Aug+21a; Orc+21; Leh+23].

Resonate and fire neuron

In the neuron model introduced in [Izh01], a resonating frequency defines each neuron and resonates when the input contains a component with this frequency. Formally, a complex internal state $z(t) \in \mathbb{C}$ defines a resonating neuron,

$$z(t) = u(t) + j \cdot i(t). \tag{3.40}$$

$u(t)$ and $i(t)$ represent a voltage and current variables, and j represents the imaginary unit. Analogous to the LIF model, the RF neuron is a single-compartment with an internal state variable affected by an input signal $x(t)$,

$$\frac{\delta z(t)}{dt} = (\gamma - j \cdot \omega_0) z(t) + x(t), \tag{3.41}$$

where γ and ω_0 are the neuron's dampening constant and resonating frequency, respectively. In the absence of input, the state's magnitude decays over time for $\gamma < 1$, and oscillates indefinitely with the same amplitude for $\gamma = 1$. We can also express (3.41) as a set of two differential equations that define the evolution of $u(t)$ and $i(t)$, respectively,

$$\begin{cases} \frac{\delta u(t)}{dt} &= \gamma u(t) - \omega_0 i(t) + x(t) \\ \frac{\delta i(t)}{dt} &= \gamma i(t) - \omega_0 u(t) \end{cases}, \tag{3.42}$$

where we assume $x(t) \in (R)$. Note that the nature of the input in (3.42) is a continuous signal over time. In the original work [Izh01], this signal is treated as the weighted sum of N spike trains,

$$x(t) = \sum_{i}^{N} w_i \delta(t - t_i), \tag{3.43}$$

where $\delta(\cdot)$ is the Dirac function, and t_i is the spike time of the ith input. If $x(t)$ contains a periodic component with a frequency close to ω_0, the neuron enters a resonating behaviour, and the magnitude of its state increases continuously. When the input frequency differs from ω_0, the dampening prevails, and the magnitude decays over time, regardless of the input signal's amplitude. In other words, an RF neuron detects signals containing specific frequency components. Employing several RF neurons with a broad range of resonating frequencies

allows us to reconstruct the input's frequency spectrum [Aug+21a]. The spiking behaviour of the RF model is introduced artificially, the same as with the LIF model. Izhikevich proposed generating spikes when the imaginary component of the neuron surpasses a constant threshold voltage [Izh01]. However, this approach cannot be scaled to signals with frequency components with varied intensities, i.e., the threshold is tuned to a specific intensity range, so high thresholds hide frequencies with small intensities, and low thresholds are over-sensitive to clutter. The work in [Aug+21a] proposes an alternative where the threshold evolves exponentially, adjusting to a broader resonance range.

The neuron's model in (3.41) can also be expressed in a discrete form, as indicated in [Orc+21], so it can be simulated in a digital processor,

$$z[t] = \gamma e^{j\omega_0 \Delta t} z[t-1] + x[t], \tag{3.44}$$

where Δt is the simulation time step.

Compared to SNNs with learnt weights, RF neurons offer the advantage of operating directly on analogue signals. Moreover, RF neurons solve the spectrum representation problem analytically, which is crucial for obtaining reliable systems that provide deterministic responses.

3.5 Neuromorphic devices

Neuromorphic devices are a key component in neuromorphic computing, as they provide a platform to accelerate SNNs efficiently. They are designed to mimic the structures of neurons and synapses in the brain and perform event-based, asynchronous computations. Developing neuromorphic devices is a research endeavour that goes hand in hand with designing and implementing new neuromorphic algorithms, as they depend on each other [Mar+20] for providing efficient and accurate brain-inspired computing. In the broadest sense, neuromorphic devices range from traditional hardware architectures like FPGAs and graphic-processing units to dedicated digital neuromorphic chips and devices based on new technologies like spintronics, memristors or photonics [Chr+22]. For a device to be considered neuromorphic, its architecture and computing paradigm should possess most of the following properties:

1. Massively parallel: The computation is distributed among multiple nodes that perform computations independently from each other.

2. In-memory computing: Each node has a local memory that is not shared with other nodes, opposite to Von Neumann architectures, where devices contain a processing unit and a memory block that is shared by all processing threads.

3. Event-based: Nodes react and process data when data is available. This contrasts with traditional processing, where data is processed in a time-based fashion, i.e., a periodic clock signal forces the processes to operate regardless of the existence of new data in the input.

4. Asynchronous: The computing nodes do not operate simultaneously or with the same time scales. This property is closely associated with the previous one, as the nodes react asynchronously to events as they arrive.

Some works have estimated the energy consumption of SNNs and compared them to that of ANNs. These measurements are theoretical and hardware-agnostic. In [PAR20], the authors compare the number of multiply-accumulate operations and accumulate operations

needed for transmitting spikes in SNNs and ANNs, respectively. In [DF21], the authors study the efficiency of SNNs from an electronics perspective, analyzing the required components for performing the operations.

This section broadly describes the most popular digital neuromorphic devices. [Chr+22] contains a more detailed overview of state-of-the-art devices and materials for accelerating neuromorphic algorithms.

3.5.1 Digital neuromorphic hardware

Despite the high research value of novel materials, their availability is very limited. Furthermore, the number of neurons they can implement is constrained to a few hundred nodes. Thus, research on SNNs usually employs CMOS-based devices for testing and validating the networks. These digital chips vary in size, efficiency, and versatility, and their intended applications range from small niche embedded processing to the execution of brain-sized simulations in large computing clusters. Table 3.2 depicts a comparison of the specifications of some popular digital neuromorphic chips, including the semiconductor process size, the number of neurons and synapses a chip can host, the area of the chip, and the efficiency in terms of giga-synaptic operations per watt unit. [Bas+22] offers a detailed overview of state-of-the-art SNN accelerators.

SpiNNaker, which stands for spiking neural network architecture, was initially developed by the University of Manchester and mainly funded by the Human Brain Project. The SpiNNaker architecture is based on a grid of SpiNNaker chips, each consisting of 18 ARM968 processor nodes operating at 200 MHz, using fixed-point arithmetic, equipped with 96 KB of internal data memory, and connected to 128 MB of external SDRAM. Each chip is connected to its six neighbouring chips using a communications network-on-chip [Fur+14]. Whereas the processing nodes inside the cores update synchronously, the chips do not share a global clock and operate asynchronously. The SpiNNaker project's main challenge was the simulation of very large SNNs at a biological time scale (1 ms [Van+18]) using a number of synaptic connections analogous to the brain cortex, i.e., around 10^4 synaptic inputs per neuron [KF16]. SpiNNaker allowed for emulating large numbers of complex neural processes in parallel efficiently [Van+18; Rho+20]. SpiNNaker was one of the first multi-core neuromorphic chips, and the first one to achieve the simulation of large brain areas (>10k neurons) at brain timescales.

The Technical University of Dresden continued the SpiNNaker project to design and manufacture the SpiNNaker 2 chip [MHF19]. Conceived to bridge the gap between realistic brain models and AI, SpiNNaker 2 aims to create synergy between these two research domains. To do so, the newer chip version includes a dedicated machine-learning accelerator that enables hybrid applications that combine spiking and non-spiking algorithms. Besides the machine-learning accelerator, each chip includes 153 cores with 19MB of on-chip SRAM. The authors aim to build a single machine with 10 million cores, leading to the possibility of simulating over 5 billion neural units.

Loihi is a 14nm, 60 mm^2 digital neuromorphic chip developed by Intel [Dav+18]. Same as SpiNNaker, Loihi is a multi-core, digital neuromorphic chip. Each chip contains 128 cores that work asynchronously and communicate spikes via a mesh structure, where a network-on-chip routes the spike messages. At the end of each step update, all cores send a barrier message and wait for the rest of the cores to reach the same state. The Loihi 2 chip is the second generation of this family, offering improved performance [Orc+21] and more versatility. With Loihi 2, Intel shipped an open-source software library named Lava, intended to serve as a cross-platform neuromorphic algorithms development tool. Loihi's software stack is prepared by default for implementing on a high-level Python interface the current-

Table 3.2: Comparison of the main specifications of popular digital and mixed-signal neuromorphic chips. Data obtained from [Bas+22].

Chip name	Process (nm)	Neurons	Synapses	Area (mm^2)	Efficiency (GSOPS/W)
SpiNNaker	130	18k	18M	88.4	0.033
Loihi	14	128k	256M	60	42.4*
TrueNorth	28	1M	128M	413	400
Tianjic	28	39k	9.75M	14.4	649
DYNAPs	180	1k	64k	38.6	33.3
ODIN	28	256	64k	0.086	78.7
BrainscaleS	65	512	128k	27.9	1280
Braindrop	28	4k	16M	0.65	2630
ROLLS	180	256	128k	44	13000*

* The power consumption reported for Loihi and ROLLS only considers the energy needed by the cores for computing synaptic operations, excluding stating energy and neuron updates.

based LIF neuron model

$$\dot{u}_i(t) = -\frac{1}{\tau_u} u_i(t) + I_i(t) - \theta_i \sigma_i(t), \tag{3.45}$$

where θ_i is the firing threshold and the current $I_i(t)$ is obtained according to

$$I_i(t) = \sum_j w_{ij}(\alpha_u \delta_j)(t) + b_i, \tag{3.46}$$

where w_{ij} is the synaptic weight, $\delta_j(t)$ is the Dirac function, $\alpha_u(t)$ is a synaptic filter impulse response, and b_j is a constant bias. Implementing more complex neuron models is done by directly coding the microcode that runs on the neural cores.

TrueNorth, designed by IBM and disclosed in 2014, was one of the first multi-core digital chips [Mer+14]. It features 4096 cores, each containing 256 million synapses and 1 million digital neurons. The chip works at a low 1 KHz clock frequency and employs a big die area, which leads to a very high efficiency and allows the implementation of many neurons inside (see table 3.2). Although not meant for running neuromorphic algorithms, it is worth mentioning IBM's NorthPole chip, introduced in 2023, which builds upon the TrueNorth technology [Mod+23]. Rather than being aimed at accelerating SNNs, NorthPole is highly specialized in matrix multiplications with fixed-point precision and takes features from neuromorphic computing for running the inference of DCNNs with state-of-the-art energy efficiency, thanks to its massive computational parallelism, small-footprint weights (8-, 4-, and 1-bit integers), no branching (i.e., no conditional *if*s in the execution), and on-chip memory.

Tianjic is a fully digital neuromorphic chip developed at Tsinghua University that allows the acceleration of both ANNs and SNNs [Den+20]. It comprises a 156-core grid with highly configurable neural nodes that allow the implementation of almost 40k neurons. The authors reported an impressive energy efficiency, yielding 649 GSOPS/W.

DYNAPs is a family of chips developed by SynSense in close collaboration with the ETH Zürich. The name stands for Dynamic Neuromorphic Asynchronous Processors. It is a 180 nm hybrid chip [Mor+17], meaning the neural units compute their internal state using analogue dynamics, and the spike communication happens via digital logic. Each chip consists of 1024

neurons distributed over 4 individually configurable neural cores, connected by a hierarchical routing grid.

BrainScaleS and its follow-up BrainScaleS-2 are a mixed-signal, single-core neuromorphic architectures developed at the University of Heidelberg [Sch+17; Göl+21]. Analogue circuits implement the logic of the neurons and synapses, and the communication between the different components is carried out digitally based on event-based protocols. The chip itself is named HICANN, and a BrainScaleS-2 system comprises 384 HICANN chips on a single wafer, with a capacity of 512 neurons per chip. HICANN chips are clustered in groups of 8, sharing an FPGA responsible for the spike communication. The maximum capacity of the wafer is around 200k neurons and 43m synapses.

ODIN is a tiny digital chip specialized in online learning for embedded systems and provides with small energy consumption figures [Fre+18]. A crossbar architecture implements the synaptic connections, and the spike communication is handled by address-event buses. The chip can implement up to 256 neurons and 64k synaptic connections with a weight resolution of 3 bits.

Braindrop is a mixed-signal neuromorphic chip that works in the subthreshold domain to compute the neuron's internal state [Nec+18]. It can host up to 4096 neurons connected by 16 million synaptic connections with 8-bit synaptic weights. The energy per synaptic spike operation is as low as 0.381 pJ and offers the best ratio of synapses per chip's surface from all the analysed chips in [Bas+22].

ROLLS, developed at the ETH Zürich, is another mixed-signal, single-core neuromorphic chip. [Qia+15]. It is tailored for real-time operations at biologically plausible timescales while maintaining low power consumption. Consuming 0.077 pJ per synaptic event, this chip yields the best energy efficiency among the analysed chips.

3.5.2 Silicon neurons

The chips introduced in section 3.5.1 contain circuit architecture designs for implementing several neurons in parallel, ranging from hundreds to millions. Most are reconfigurable and flexible regarding their application, input dimensions and synaptic connectivity patterns. Additionally, they use digital communication protocols for transmitting and receiving spikes. Alternatively, many embedded applications are better suited for ad-hoc circuits with analogue interfaces that compute spikes online from the raw signals of sensors. These circuits consist of single-channel electric designs that replicate the function of biological sensing cells and can be implemented efficiently as integrated circuits; hence, they will be called *silicon neurons* for the rest of this chapter. These circuits fulfil the task of encoding analogue signals to spikes [Zha+17], which was the original goal of neuromorphic Engineering [Chr+22, chapter 10]. Multiple silicon neurons can be assembled together to form multi-node hybrid chips like the ones introduced in section 3.5.1. There is a high diversity of silicon neurons due to the wide variety of dynamic models [Ind+11; Chi+14]. They typically work on the sub-threshold region of MOSFETs, also called the weak-inversion region, and most of them contain a capacitor that charges and discharges depending on the input to the circuit, which can be modelled as a voltage or a current. Depending on the complexity, silicon neurons replicate simple integrate-and-fire models [Van01] or more complex models like the Morris-Lecar model [Sou+17] and the adaptive exponential integrate-and-fire model [LI09]. Most of these models show an inverse relationship between the input and the spike time. This turns into a logarithmic or hyperbolic mapping from input to spike times. Most of the applications of silicon LIF neurons encode information into spike bursts using ISI [Rot+22] and decoding the spike rate afterwards.

[WD08] introduces a circuit for a bursting neuron with a *slow variable* that modulates the leak current, inspired by the neuron model from Izhikevich [Izh+03]. The circuit can produce spike trains of 200 Hz with an efficiency of 900 pJ per spike. [LI09] presents a conductance-based ISI circuit with adaptation and refractory period, compatible with the address-event representation. The work in [Rot+22] uses a voltage-controlled oscillator for generating spikes at a rate proportional to the voltage input. They realize the oscillator using an RC circuit together with a MOSFET transistor, where the resistive element is voltage-controlled, enabling a hysteresis effect. In [Zha+17], the authors thoroughly described different neural encoders and designed an analogue circuit based on nMOS transistors for obtaining an ISI encoder. Their implementation did not rely on operational amplifiers, and they carried comprehensive results that showed considerable robustness and error tolerance. The work in [CI15] implements a TC encoder for neural recording based on the principle of a delta-modulation *analog-to-digital converter* (ADC) [Tan+13]. The circuit produces positive and negative pulses compatible with the address-event representation depending on whether the input increases or decreases over a threshold. In [Sou+17], the authors introduce a silicon neuron that faithfully implements the Morris-Lecar neuron model, focusing on minimizing the power consumption and the circuit's size thanks to ultra-low capacitance values, down to 4 fF. Besides the original circuit, the authors provide a simplified circuit that optimizes the performance at the cost of less biological accuracy. The simplified circuit occupies $35\mu m^2$ and yields spike bursting behaviours up to 26 kHz and a consumption of 105 pW, resulting in an energy of 4 fJ per spike including the DC power.

3.5.3 Alternative technologies

Most neuromorphic devices are implemented with traditional CMOS technology with architectures that enable the asynchronous and parallel nature of neuromorphic computing algorithms. However, research has also explored novel materials such as magnetic alloys, organic materials, phase-change memory devices, and 2D materials; or circuits formed by memristors, optical processors, spin-torque nano-oscillators, and quantum processors. [Chr+22] offers an overview of the trends with these alternative technologies and the outlook for the following years.

Memristors are passive analogue devices composed of successive layers of electrodes and insulators. Memristive devices comprise a large grid of such components, where each element works as a memory and computing element, i.e., memristors implement in-memory computing. Their state persists in the absence of voltage, which makes them ideal for providing non-volatile memory. Memristors can provide binary values or continuous voltages within a given range. When combining memristors with digital processors, their input needs to be interfaced with an ADC, and their output with a DAC [Bao+22]. This is one of the main limitations of memristors, as it considerably degrades the energy and latency performance.

Spintronic memory devices use nanoscale magnetization dynamics to implement non-volatile memory devices. The working principle of these devices is the generation of magnetic oscillations in the presence of an input current. The oscillations are then converted into a voltage across a magneto-resistance [Tor+17]. The output voltage and the input current follow a non-linear relationship of the form

$$V_{\text{out}} \propto \sqrt{I_{\text{in}} - I_0}\,, \tag{3.47}$$

where I_0 is a threshold constant, below which the voltage remains zero. The value of V_{out} depends too on the previous state of the device, granting it the memory property. Thus, spin-torque nano-oscillators can emulate the functionality of neurons. This technology has significant potential in neuromorphic engineering due to its low energy consumption, reliability

and area requirements [Wan+23]. The authors in [Wan+23] report an energy consumption of 486 fJ per spike and a firing rate up to 17 MHz. In [Tor+17], authors use spintronic oscillators for replicating neuron reservoirs and learning the task of speech digit recognition, yielding a competitive accuracy. The scalability of spintronics has yet to be proven by creating experiments beyond toy examples and offering high accuracy for real-world scenarios.

Optical processors implement what is known as photonic computing methods by communicating information using photons as carriers of information. Compared to electronic implementations, the main advantages are their high throughput, high degree of parallelization, low power consumption and high speed during information transport [Chr+22, chapter 13]. The research on the application of photonics for AI has increased considerably in the last decade due to the demand for alternatives to traditional semiconductor technology. Optical processors are suitable for transmitting massively parallel data for the multiply-accumulate operations in ANNs, thanks to *wavelength-division multiplexing*, where multiple optical carriers can be transmitted on a single fibre by using different wavelengths. The weighting of the synaptic lanes is implemented within the amplification and attenuation circuits that multiplex the signals. Research in photonics has provided mechanisms for implementing the spiking behaviour in SNNs [Sha+21b]. For example, the work in [Rob+20] shows the implementation of a LIF neuron with multiple weighted inputs based on a vertical-cavitysSurface-emitting Laser, a standard and cheap technology for implementing photonic systems. The system is tested for integrating and coincidence detection tasks, with a pulse duration of 100 ns.

4

Temporal charge before spike neuron model

Signal processing has been a fruitful field for several decades, providing many efficient algorithms that can process data from different sources and generate solutions for a broad range of tasks. Many signal-processing algorithms use matrix multiplications to compute their output, and optimising this operation is paramount for achieving efficient processing pipelines. In recent years, neuromorphic computing has focused on applications that go beyond understanding the brain, and solve computational problems such as constraint satisfaction, graph algorithms, or partial differential equations [Aim+22]. This chapter introduces and describes in detail the *temporal charge before spike* (TCBS) neuron model, one of the main contributions of this thesis. The initial goal of the TCBS was computing the *Fourier transform* (FT) [Lóp+22]. Besides its application as a replacement for the FT, the TCBS model also serves as a tool for converting *deep convolutional neural networks* (DCNNs) to *spiking neural networks* (SNNs) [LRK22]. Section 4.3 describes the general approach for converting DCNNs and appendix D shows results on the task of digitclassification.

4.1 Working principle

The TCBS neuron model is a variation of the *leaky integrate-and-fire* (LIF) for converting to the spiking domain matrix-vector multiplications of the form

$$\boldsymbol{y} = W^T \boldsymbol{x}\,, \tag{4.1}$$

where $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{y} \in \mathbb{R}^M$, and $W \in \mathbb{R}^{N\times M}$ are the input vector, the output vector, and the coefficients matrix, respectively. A spiking version of (4.1) needs to replace the input and output vectors by binary spike trains unrolled over time, $s_{\text{in}}(t)$ and $s_{\text{out}}(t)$, where $s(t) \in \{0,1\}$,

$$\boldsymbol{s}_{\text{out}}(\boldsymbol{t}) = \hat{W}^T \boldsymbol{s}_{\text{in}}(\boldsymbol{t})\,. \tag{4.2}$$

As the aim of the TCBS is to improve the efficiency of sensor processing on neuromorphic hardware, the design had two priorities in mind:

1. To perform a lossless conversion so the output of the original matrix multiplication persists. This means that the output spikes shall hold a mathematical relationship with the expected output in the original operation and that the original data can be reconstructed after the mapping to spikes.

2. The spike density needs to be as sparse as possible for granting an energy-efficient computation. For this reason, the TCBS uses time coding.

4.1.1 Information encoding

The TCBS neuron model is based on temporal spikes, i.e., the information travelling across the synaptic connections is stored in the precise spike times using *phase encoding* (PE) (see section 3.2). The TCBS neuron model does not perform a conversion from analogue values to spikes. Instead, the neuron's inputs and output are spike trains. Namely, the model assumes that the information is encoded into the delay of the arriving spikes,

$$t = t_{\min} + (t_{\max} - t_{\min}) \cdot \frac{x_{\max} - x}{x_{\max} - x_{\min}}, \tag{4.3}$$

where the original data x is converted from the range $[x_{\min}, x_{\max}]$ to a spike time t in the range $[t_{\min}, t_{\max}]$. Figure 4.1 shows the mapping described in (4.3).

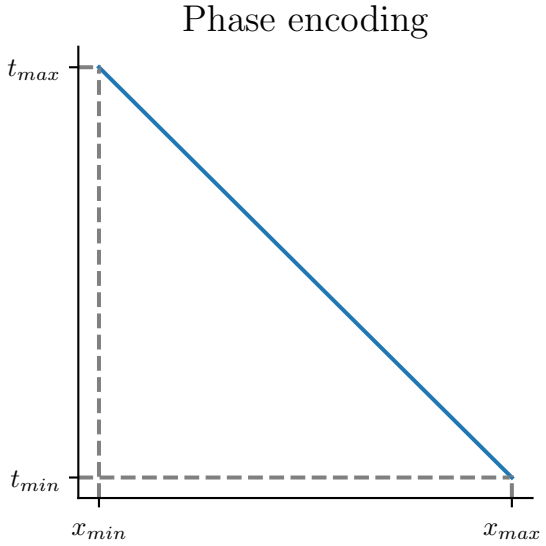


Figure 4.1: Conversion from an analogue value x in the range $[x_{\min}, x_{\max}]$ to a spike time t in the range $[t_{\min}, t_{\max}]$ by using phase encoding.

For simplicity, let us assume that the spike times start at $t_{\min} = 0$. Let us also denote as γ the relationship between $t_{\max}$ and the input data range

$$\gamma = \frac{t_{\max}}{x_{\max} - x_{\min}}. \tag{4.4}$$

Thus, we can simplify 4.3 to

$$t = \gamma(x_{\max} - x). \tag{4.5}$$

The resolution of the mapping in (4.3) depends on the number of time steps used for generating the spikes. As this number is generally lower than the size of the data type used for representing x, the encoding leads to a decrease in the data resolution. The minimum observable difference in the input Δx is determined by

$$\Delta x = \frac{x_{\max} - x_{\min}}{t_{\max}} \cdot \Delta t, \tag{4.6}$$

where Δt is the time step resolution of spike the output. Δx is considerably smaller than the original resolution of the data, as the number of time steps stays in the range $10^1 - 10^2$, which is several orders of magnitude of resolution often used on digital signal processors.

An important property of (4.5) is that high values of x are mapped into small spike times t, and vice versa. This property simplifies the computation of certain algorithms, like *winner-takes-all* algorithms or max pooling operations, where only the value of the most dominant inputs is relevant. These algorithms filter out low input values and propagate to the output high values.

Encoding negative values

The introduced encoding implements a negative linear mapping from original values x to spike times t trough the whole range $[x_{\min}, x_{\max}]$. This mapping is based on the intuition that the higher x is, the more information it carries. However, this statement does not hold when working with input ranges that include negative values, i.e., if $x_{\min} < 0$. On the negative spectrum, x typically carries more information the lower its value is. Hence, the mapping (4.3) has to be modified so early spikes represent negative values with high information as well.

Two alternative methods are proposed for encoding negative values (see fig. 4.2):

1. *polar encoding*, where negative spikes represent negative values.
2. *discontinuous encoding*, where spike times smaller and larger than $t_{\max}/2$ represent positive and negative values, respectively.

For implementing *polar encoding*, the spikes are represented not only by the spike time t, but also by the polarity $p \in \{-1, 1\}$. Assuming $x_{\max} = -x_{\min}$, the spike times for *polar encoding* are calculated as

$$t = t_{\min} + (t_{\max} - t_{\min}) \cdot \frac{x_{\max} - |x|}{x_{\max}}, \tag{4.7}$$

where $|x|$ denotes the absolute value of x. The polarity of the spike is based on the sign of x,

$$p = \begin{cases} 1, & \text{if } x > 0 \\ -1, & \text{if} x < 0 \end{cases}. \tag{4.8}$$

The corner case $x = 0$ can be mapped to a spike at $t_{\max}$ with positive polarity, i.e., turning the condition for the first half of (4.8) to $x \geq 0$. A more meaningful option is to not generate a spike in this situation, as in most problems, $x = 0$ does not carry any information. This approach is more efficient and keeps the mapping of the positive and negative values symmetric. However, careful design is required when the encoding is implemented in dedicated neuromorphic hardware.

For *discontinuous encoding*, (4.3) is split into two segments. The positive values of x are mapped to the range $[0, t_{\max}/2]$, and the negative values of x are mapped to $[t_{\max}/2, t_{\max}]$. Assuming $x_{\max} = -x_{\min}$, *discontinuous encoding* is modelled as

$$t = \begin{cases} t_{\min} + t_{\text{half}} \cdot \dfrac{x_{\max} - |x|}{x_{\max}}, & \text{if } x > 0 \\ (t_{\min} + t_{\text{half}}) + t_{\text{half}} \cdot \dfrac{x_{\max} - |x|}{x_{\max}}, & \text{if } x < 0 \end{cases}, \tag{4.9}$$

where $t_{\text{half}} = t_{\max}/2$. Same as for *polar encoding*, $x = 0$ can either be mapped to $t_{\max}$, or it can be ignored.

Discontinuous encoding has the advantage that it can be easily implemented in neuromorphic hardware, as it does not require an extra variable for representing polarity. The main disadvantage is that for the same time range, the resolution of *discontinuous encoding* is half than that of *polar encoding*.

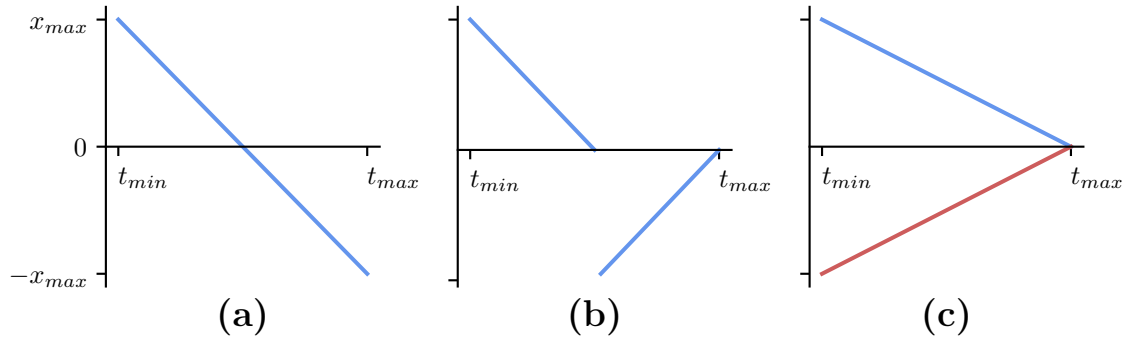


Figure 4.2: The three schemes proposed for converting real values to spike times using latency encoding. From left to right: (a) the default scheme, (b) *discontinuous encoding*, and (c) *polar encoding*, where the red plot represents negative spikes.

4.1.2 Silent and spiking stages

The TCBS neuron model is inspired by the work from Rueckauer and Liu [RL18]. In this work, the authors propose a neuron model that only requires one spike per input, translating into a charge of the neuron membrane potential. Assuming that the neuron is simulated on discrete time steps, the membrane potential of neuron i on time step t depends on the spike times of every pre-synaptic neuron j that have spiked before t ($t_j < t$), also called *causal neurons* and represented by $\Gamma_i^<$. The change of the membrane voltage between two consecutive time steps is given by

$$\Delta u_i = \Delta t \sum_{j \in \Gamma_i^<} w_{ij}\,, \tag{4.10}$$

where Δt is the simulation time step. In other words, the instantaneous rate of change of the membrane potential is equal to the sum of the weights from the pre-synaptic neurons that have already spiked. The work in [RL18] includes two spiking mechanisms for the model described by (4.10). The first mechanism consists of a fixed threshold that triggers a spike when $u(t)$ reaches it, and the second estimates the missing input by setting a dynamic threshold that decreases with each arriving spike. The main drawback of this model is that it cannot transform the membrane potential into output spikes without losing information. For a fixed threshold, the input spikes that would have arrived after the neuron reaches the threshold are not considered for computing the output. Moreover, due to the wide range of input-weight combinations, there are specific input patterns that do not bring the membrane voltage to a value higher than the threshold, not generating a spike and hence making these values go unnoticed. The second method relies on a likelihood method that attempts to estimate the missing information based on the weights of the missing input without any knowledge of the input's magnitude.

Instead, the TCBS introduces a lossless spiking mechanism, splitting the neuron's principal operation into two stages over time, as shown in fig. 4.3. The neuron accumulates the information from all input spikes during the first stage, called *silent stage*. During the second stage, called *spiking stage*, the neuron maps the stored information into a single output spike. The neuron is time-based, and its spike time is deterministic and based on the information from all inputs.

Silent stage:

The neuron i receives input spikes during this stage, and its membrane voltage $u_i(t)$ gets charged accordingly. Moreover, the neuron cannot produce output spikes during this stage. The contribution to the membrane voltage of a spike from the jth connection is proportional to the synaptic weight w_{ij} in the same way as in (4.10), i.e., the neuron gets charged from the spike time t_j until the end of the silent stage, at time t_s. Hence, the final contribution to the membrane voltage will be proportional to $t_s - t_j$ times the value of the jth synaptic weight and thus proportional to the output of the original vector-matrix multiplication,

$$u_i(t_s) \propto y_i \,. \tag{4.11}$$

Generalizing for a neuron i with several input connections, we define the membrane voltage at time t as

$$u_i(t) = \sum_{j \in \Gamma^<} w_{ij}(t - t_j) + o_i \,, \tag{4.12}$$

where $\Gamma^<$ is the set of neurons that have spiked before time t, also called causal neurons, and o_i is an offset constant used for correcting shifts in the neuron.

The voltage at the end of the silent stage is computed from (4.12) for $t = t_s$. Using the mapping for the temporal encoding (4.4) on (4.12), for $t_{\max} = t_s$, the voltage results in

$$u_i(t_s) = \sum_j \left(\gamma W_{ij} x_j + W_{ij} t_s - \gamma x_{\max} W_{ij}\right) . \tag{4.13}$$

Using (4.1) and collecting the second and third terms of the sum under a constant β, we can simplify (4.13) as

$$u_i(t_s) = \gamma y_i + \beta \,, \tag{4.14}$$

where β is calculated as

$$\beta = -\gamma x_{\min} \sum_j w_{ij} \,. \tag{4.15}$$

Equation (4.14) indicates that the membrane voltage at time t_s is equivalent to the original output y after applying a scaling γ and a shift β, which are constants defined by the hyperparameters of the SNN. To preserve the proportionality with the original multiplication (4.11), we must set the neuron's offset constant to $o_i = -\beta / t_s$. We can observe that the offset is zero when $x_{\min} = 0$ and non-zero for all other cases.

Derivation. *For obtaining the value of β in* (4.15)*, we expand* (4.13) *and combine it with* (4.1)

$$u_i(t_s) = \gamma y_i + (t_s - \gamma x_{max}) \sum_j w_{ij} \,. \tag{4.16}$$

Then, we solve the value of t_s on the second term using (4.4)*, which results in*

$$u_i(t_s) = \gamma y_i - [\gamma(x_{max} - x_{min}) - \gamma x_{max}] \sum_j w_{ij} \,. \tag{4.17}$$

Simplifying, we obtain

$$u_i(t_s) = \gamma y_i - \gamma x_{min} \sum_j w_{ij} \,. \tag{4.18}$$

We obtain the value of β from the second term in the last equation and express $u_i(t_s)$ as in (4.14).

□

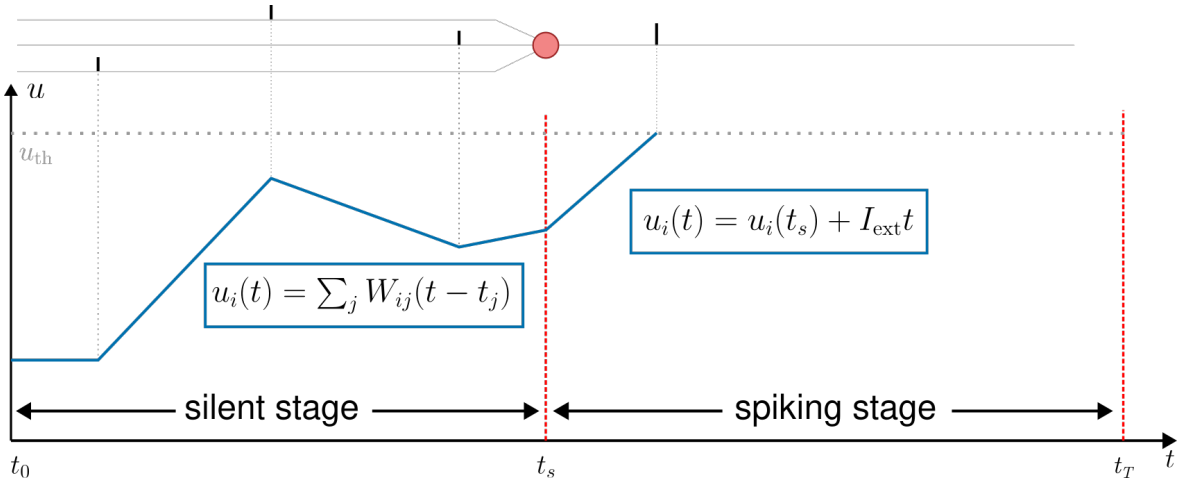


Figure 4.3: On top, representation of a TCBS neuron with three synaptic inputs and one synaptic output. The neuron receives one spike over time from each input, respectively, and generates one single output spike. At the bottom, a plot of the membrane voltage of the TCBS neuron. During the charging stage, the neuron collects input spikes and charges its membrane voltage. During the spiking stage, the neuron transforms the membrane voltage into a precise spike over time. ©2022 IEEE

Spiking stage:

During this stage, the neuron's membrane potential is charged until it reaches the membrane threshold u_{th} and produces a spike. The charging is modelled in a way that the spike time of the neuron is inversely proportional to the membrane voltage at the start of this stage and follows the same encoding as for the input (4.3). To achieve this behaviour, the time of the output spike needs to take the form

$$t_i = t_{\min} + (t_{\max} - t_{\min}) \frac{y_{\max} - y}{y_{\max} - y_{\min}}, \tag{4.19}$$

where $y_{\max}$ and $y_{\min}$ are the maximum and minimum values that the output y can take. Equation (4.19) is analogous to (4.3), hence the output spike follows the same encoding principle as the input. We obtain the spike time described in (4.19) by charging the neuron with a constant current I_{ext}

$$u_i(t+1) = u_i(t) + I_{\text{ext}}. \tag{4.20}$$

The membrane voltage at a time t is obtained as

$$u_i(t) = u_i(t_s) + I_{\text{ext}}(t - t_s), \tag{4.21}$$

which can be simplified to

$$u_i(t) = u_i(t_s) + I_{\text{ext}}t \tag{4.22}$$

if we assume $t_s = 0$. To calculate the spike time t_i, we apply (4.21) for a membrane voltage $u(t_i) = u_{th}$,

$$t_i = t_s + \frac{u_{th} - u_i(t_s)}{I_{\text{ext}}}. \tag{4.23}$$

The values of the parameters I_{ext} and u_{th} are determined based on the voltage range at the end of the silent stage $u(t_s) \in [u_{\min}, u_{\max}]$, and the number of simulation steps. If the ratio $u_{\text{th}}/I_{\text{ext}}$ is not large enough, small values of $u(t_s)$ will not yield a spike, as shown in fig. 4.4. The membrane threshold u_{th} is fixed so that the maximum possible input yields a spike at the start of the spiking stage t_s. A naive approach for computing u_{th} is to fix it to the theoretical

maximum voltage at the end of the silent stage

$$u_{th} = \max_{t=t_s}\{u(t)\} = \sum_{w\in W^+} wt_s\,, \tag{4.24}$$

where W^+ is the subset of positive weights. Equation (4.24) is obtained after making two assumptions: (1) all positive inputs will take the maximum possible value, which is mapped to a spike time $t(x_{\max}) = 0$, and (2) all negative inputs will take the minimum possible value $t(x_{\min}) = t_s$, so they do not contribute to the final value. As this situation is highly unlikely, the value u_{th} can instead be adjusted to a lower value depending on the nature of the data it will receive,

$$u_{th} = \alpha_{\text{th}} \max_{t=t_s}\{u(t)\}\,, \tag{4.25}$$

where α_{th} is a coefficient for scaling down the threshold.

The value I_{ext} needs to fulfil the boundary condition that the minimum membrane voltage at the end of the silent stage $u_{\min} = \min\{u_i(t_s)\}$ produces a spike at the end of the simulation time t_T. This results in

$$I_{\text{ext}} = \frac{u_{th} - u_{\min}}{t_T - t_s}\,. \tag{4.26}$$

The relationship between the spike time t_i and the original output y_i is obtained from (4.23) by computing the values I_{ext} and u_{th} using the mapping in (4.14) for $u_{\max}$ and $u_{\min}$. The spike time results in

$$t_i = t_s + (t_T - t_s)\frac{y_{\max} - y_i}{y_{\max} - y_{\min}}\,, \tag{4.27}$$

which is an instance of (4.19) for the specific case of $t_{\min} = t_s$ and $t_{\max} = t_T$.

Derivation. *The spike time of the neuron is obtained by combining* (4.23) *and* (4.12)*, which results in*

$$t_i = t_s + \frac{u_{th} - \beta}{I_{ext}} - \frac{\gamma}{I_{ext}} y_i\,. \tag{4.28}$$

Using the mapping (4.14)*, the limits of* $u_i(t_s)$ *are*

$$u_{max} = \gamma y_{max} + \beta \tag{4.29}$$

and

$$u_{min} = \gamma y_{min} + \beta\,. \tag{4.30}$$

Now, we expand I_{ext} *using* (4.26)*, and we replace* u_{th} *with the equality* $u_{th} = u_{max}$*. Then* (4.28) *results in*

$$t_i = t_s + \frac{\gamma y_{max} + \beta - (\gamma y_i + \beta)}{\frac{\gamma y_{max} + \beta - (\gamma y_{min} + \beta)}{t_T - t_s}}\,. \tag{4.31}$$

Equation (4.31) *can be simplified to*

$$t_i = t_s + \frac{y_{max} - y_i}{\frac{y_{max} - y_{min}}{t_T - t_s}}\,. \tag{4.32}$$

By rearranging the fractions, we can turn (4.32) *into* (4.27).

□

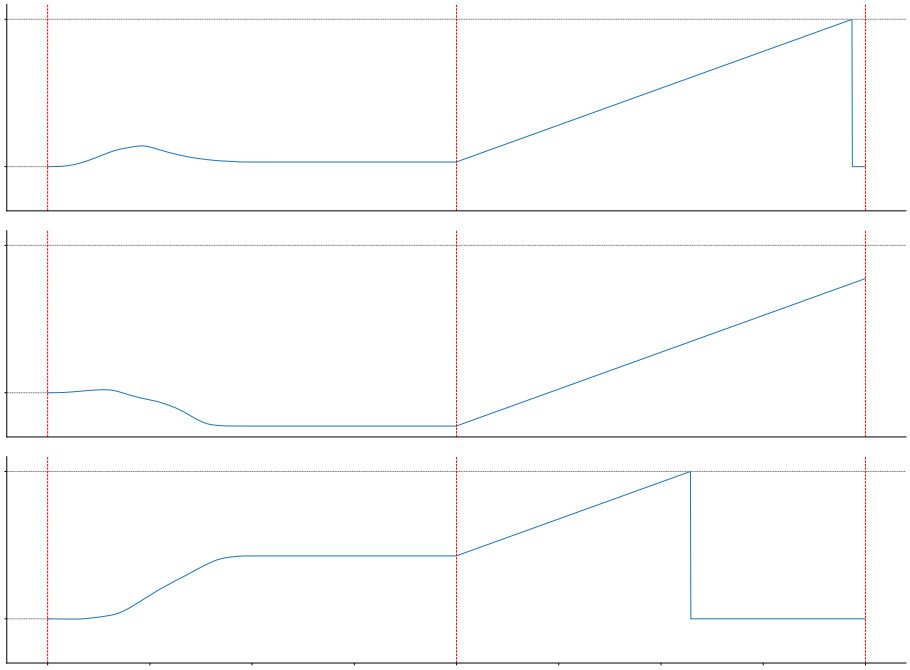


Figure 4.4: Evolution of the membrane potential of 3 different TCBS neurons. The first neuron charges to a low voltage and produces a late spike, the second charges to a voltage that lies beneath the effective range, and the third charges to a high voltage that produces a fast spike.

4.1.3 Multi-layer structure

Section 4.1 describes the working principle of the TCBS when we have a single neuron with multiple inputs. The neuron enables the computation of vector-matrix multiplications where information is carried on time-coded spikes. Often, signal processing algorithms are structured in recursive operations, where the multiplication output is chained to further multiplications. For example, DCNNs are organized in multiple layers of neurons, where the depth is up to tens of layers. Efficient computations of the FT also split the algorithm into consecutive layers, resulting in butterfly structures (see section 2.3).

Let us then generalize the TCBS neuron model for an architecture composed of several neural nodes structured in a multi-layer architecture. For a correct function of an SNN composed of TCBS neurons, the simulation of the silent stage of each layer overlaps with the spiking stage of the previous layer,

$$(t_T - t_s)^{l)} = (t_s - t_0)^{l-1)}, \tag{4.33}$$

where the superscript $(\cdot)^{l)}$ denotes the layer l to which the times within the parenthesis refer to. Therefore, it is possible to implement SNNs with an indefinite number of layers if the requirement in (4.33) is met. This results in a highly sparse SNN over time, as an SNN with L layers will have a total of $L+1$ processing stages. For a given input vector, each layer will only be active during two stages (see fig. 4.5).

For an SNN with L layers, the total processing time τ_T for generating an output spike train from an input vector will be

$$\tau_T = t_0 + \sum_{l}^{L} (t_T - t_S)^{l)}, \tag{4.34}$$

where t_0 is the simulation time for the input spike train. If all layers have the same stage simulation time $(t_s - t_0)^{l)} = (t_T - t_s)^{l)}$, (4.34) yields $\tau_T = (L+1)\tau_l$, where τ_l is the simulation time per stage. Another property of the TCBS SNN is that, for sequential data, it is possible to feed an input vector before the output for the previous input vector is generated. Namely, the shortest input period T_T is given by the slowest stage in the network,

$$T_T = 2 \cdot \max_l \{(t_t - t_S)^{l)}\}. \tag{4.35}$$

In case that all layers have the same stage simulation time, (4.35) yields $T_T = 2 \cdot \tau_l$. Implementing a single-layer SNN does not require correcting the scaling in the output spike in (4.27), as it can be applied after the decoding. However, a multi-layer architecture must account for this and scale the spike times back to the original range. This can be achieved by applying to the synaptic weights a scale based on the parameters of the pre-synaptic neuron.

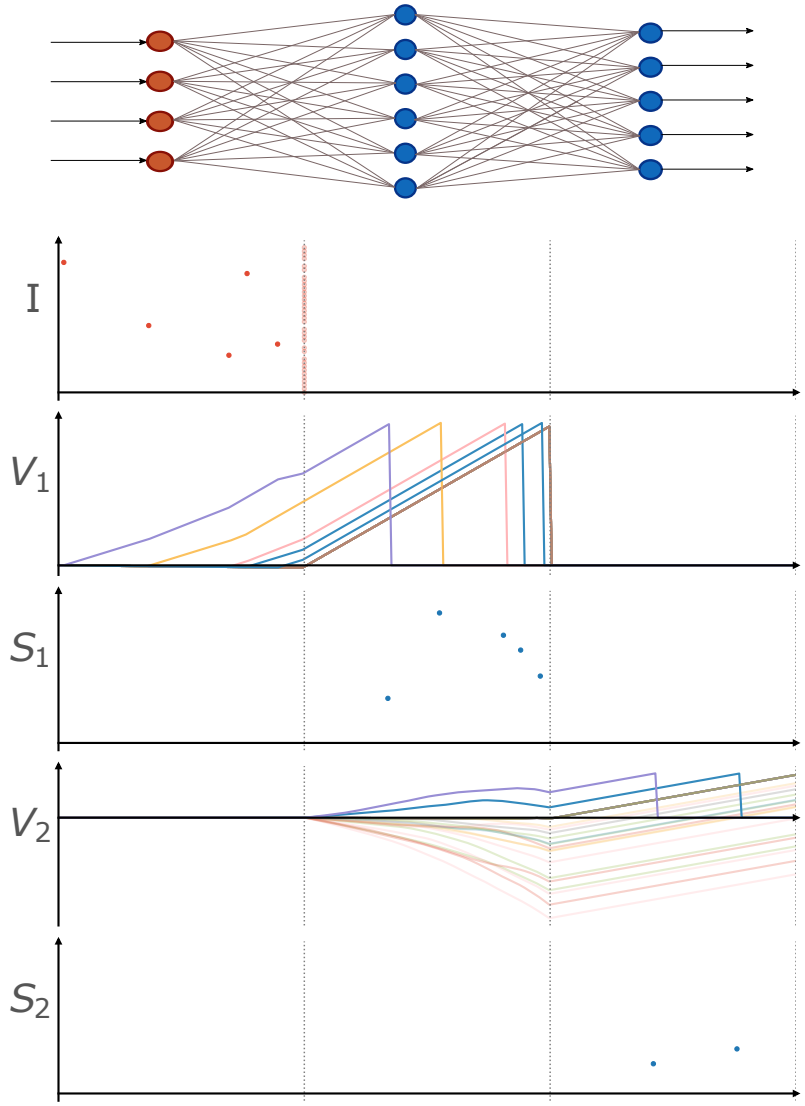


Figure 4.5: Representation of the voltages and spikes of a two-layer TCBS network over time. The spike times in the input layer (red) overlap with the silent stage of the first layer. The remaining layers' silent stages overlap with the preceding layers' spiking stage. Each layer stays idle outside its silent and spiking stage. In general, the density of spikes diminishes with the depth of the network.

4.1.4 Computational cost

The TCBS model uses (4.12) to update every neuron's membrane potential on each time step of the silent stage. Implementing this model implies going on every time step over the weights of all causal neurons (i.e., neurons that have already spiked). Alternatively, the variable I_{ext} can accumulate the weights of all causal neurons and thus avoid the redundant summation in (4.12), as well as avoiding to store the identities of the causal neurons at time t, leading to

$$I_{\text{ext}}(t) = I_{\text{ext}}(t-1) + \sum_j w_{ij} s_{\text{in}}(t), \tag{4.36}$$

where $s_{\text{in}}(t)$ is the spike input at time t. After updating the current in (4.36), the membrane potential is reduced to the simple sum

$$u_i(t+1) = u_i(t) + I_{\text{ext}}(t)\,. \tag{4.37}$$

During the spiking stage, I_{ext} is constant and each time step needs only to compute the addition in (4.37). The model represented by (4.36) and (4.37) is a current-based LIF neuron model, which is supported by digital chips like SpiNNaker and Loihi. For $2T$ time steps split evenly between the silent and spiking stages, and $\bar{n_s}$ input spikes, the total number of addition operations per post-synaptic neuron for the silent stage is $n_{a1} = T + n_s$, and $n_{a2} = T$ for the spiking stage, yielding a total number of addition operations

$$n_a = 2T + n_s\,. \tag{4.38}$$

4.2 Neuromorphic computation of the Fourier transform

The supremacy of the FT for the frequency spectrum analysis is unquestionable, and there is no evidence that a learning algorithm could beat it. However, we can take advantage of the event-based, parallel architecture of neuromorphic hardware and spiking neuron models to replicate the functionality of the FT. While offering equivalent output, neuromorphic implementations can provide a fast, low-energy alternative to the FT.

Some works in recent years proposed spiking networks for doing a partial or full analysis of the frequency spectrum of temporal signals. In [Jim+16], the authors explored the usage of SNNs for extracting specific frequencies from silicon cochleas, i.e., neuromorphic implementations of the cochlea that output spikes [CLS07]. Another alternative is to use resonate-and-fire (RF) neurons [Izh01].

Alternatively, the TCBS neuron model can solve the FT algorithm. The FT trigonometric form (2.33) is a linear mapping of the input data of the form

$$Y = W_F \cdot X\,, \tag{4.39}$$

where W_F is the matrix containing the Fourier coefficients. Thus, the TCBS model can be used for a lossless computation of the FT, i.e., the spiking times in (4.27) are proportional to the output in (4.39), given that the weights are replaced by W_F and the input X is converted to spikes using (4.4).

As the output of the Fourier transform are complex values, the output layer of the *spiking Fourier transform* (S-FT) is split into neurons representing the real components and the imaginary components of the K frequency bins. Thus, (4.39) can be rewritten as the algebraic linear system

$$\begin{bmatrix} Re\left(Y^{(N)}\right) \\ Im\left(Y^{(N)}\right) \end{bmatrix} = \begin{bmatrix} W_{Re} & W_{Im} \\ -W_{Im} & W_{Re} \end{bmatrix} \begin{bmatrix} Re\left(Y^{(N-1)}\right)^T \\ Im\left(Y^{(N-1)}\right)^T \end{bmatrix}, \tag{4.40}$$

which can be implemented as a neural layer with $2 \times L$ neurons, where half represent the real values of the DFT and the other half represent the imaginary values, and W_{Re} and W_{Im} are the weights derived from (2.33). Therefore, the input of the S-FT (4.39) is computed in parallel with the real and imaginary Fourier coefficients.

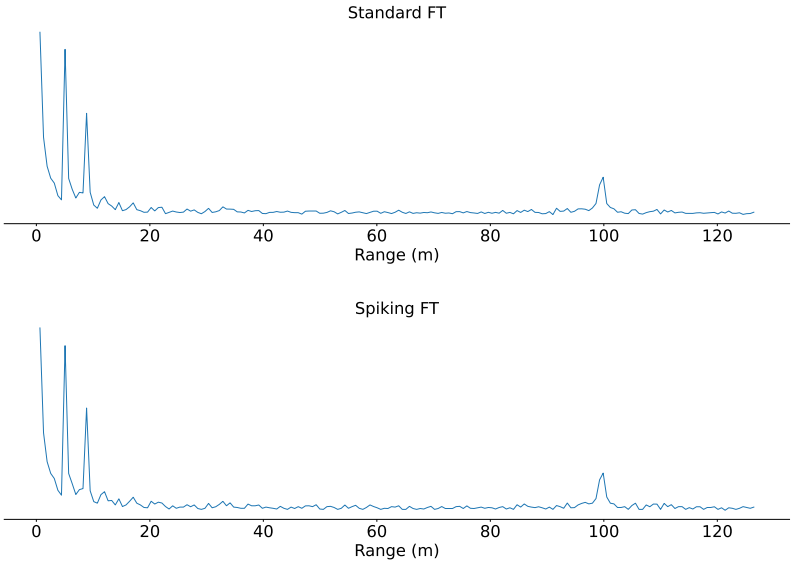


Figure 4.6: Example of computing the range FT on a sample chirp from simulated radar data using (a) a software-based *fast Fourier transform* (FFT), and (b) the S-FT.

table 4.1 shows the summary of parameters for an S-FT length of $N = 512$ bins. Note that the output dimension is $2N$, as bins are split into real and imaginary components, each represented by one neuron. fig. 4.6 depicts an example of a frequency spectrum generated with the S-FT, and a standard *fast Fourier transform* (FFT) for reference.

Table 4.1: Parameters for a time-coded S-FT solving a 1D FT of length 512.

Parameter	Value
Nº of neurons	2^{10}
Nº of synapses	2^{20}
Nº of input spikes	2^{9}
Nº of output spikes	2^{10}
Variables per neuron	2(u, Iext)

L2 norm of the FT

Whereas the computation of the FT yields a result in complex coordinates, some processing pipelines demand the calculation of the output's modulus and phase. The modulus F of the FT spectrum is defined as

$$F = \|\mathbf{Y}\|_2 = \sqrt{Re(Y)^2 + Im(Y)^2}\,. \tag{4.41}$$

If the application needs the exact value of the modulus, the operation in (4.41) has to be computed using an arithmetic logic unit. Some neuromorphic chips like SpiNnaker 2 provide the logic onboard for computing arithmetic operations. Alternatively, the power and root

operations can be simplified by turning (4.41) to a logarithmic scale and assuming that the logarithm of a sum equals the max of its elements,

$$\begin{aligned} log(F) = \frac{1}{2} log(Re^2 + Im^2) \approx \frac{1}{2} \cdot max\{2 \cdot log(Re), 2 \cdot log(Im)\} \\ \Rightarrow log(F) \approx max\{log(Re), log(Im)\}. \end{aligned} \tag{4.42}$$

The previous computation can be implemented if the neuron model produces spikes whose timing indicates the logarithm of the real and imaginary parts, respectively,

$$\begin{aligned} t_{iRe} = log(Re) \\ t_{iIm} = log(Im). \end{aligned} \tag{4.43}$$

4.2.1 Computation of N-Dimensional FT

The introduced model can be adapted for computing an FT with multiple dimensions, as in the case of the range-Doppler map of a radar frame. The output of the 1st layer of the S-FT produces spikes within the range $[0, 1]$, whereas the FT values F they represent are in the range $[-F_{max}, F_{max}]$. Thus, the conversion from the ith FT value F_i to its corresponding spike time t_i is

$$t_i = 1 - \frac{F_i + F_{max}}{F_{max} - (-F_{max})} = \frac{F_{max} - F_i}{2F_{max}}. \tag{4.44}$$

To obtain an FT with several dimensions, the output each transform is fed to the FT function N times. For example, to obtain the 2nd dimension of a 2D FT, we compute

$$F_j^{2)} = \sum_{l=0}^{L-1} F_k^{1)} \cdot [C_l - i \cdot S_l], \tag{4.45}$$

where the input vector $F^{1)}$ has L components, C_l and S_l are the cosine and sine terms of the lth element, and i is the complex operator. We can also express (4.45) in matricial form,

$$F^{2)} = F^{1)} \begin{bmatrix} W_{Re} \\ -W_{Im} \end{bmatrix}. \tag{4.46}$$

To obtain equivalent FT values for input spike times that follow the rule in (4.44), (4.44) and (4.45) are combined:

$$\begin{aligned} F_j^{2)} &= \sum_{k=0}^{L-1} F_{max}(1 - 2t_i) \cdot [C_k - i \cdot S_k] \\ &= -2F_{max} \sum_{k=0}^{L-1} t_i \cdot [C_k - i \cdot S_k] + F_{max} \sum_{k=0}^{L-1} \cdot [C_k - i \cdot S_k]. \end{aligned} \tag{4.47}$$

We can rearrange the previous equation as a linear combination of the input spike times t_i with the L different cosine and sine terms over the chirp dimension, plus a constant value. This can be rewritten in algebraic form as

$$\left[F^{2)}\right] = \boldsymbol{t} \begin{bmatrix} -2F_{max}W_{Re} \\ 2F_{max}W_{Im} \end{bmatrix} + \begin{bmatrix} F_{max}W_{Re} \\ -F_{max}W_{Im} \end{bmatrix}. \tag{4.48}$$

Thus, the second dimension of the FT can be directly calculated from the spike times of the S-FT's first layer. As the computations of the FT's first and second dimensions are identical, a two-layer S-FT can be used for computing a two-dimensional FT. The spike times of the second layer hold the relationship defined in (4.44) with the 2D FT,

$$t_i^{2)} = \frac{F_{max}^{2)} - F_i^{2)}}{2F_{max}^{2)}}. \tag{4.49}$$

where $t_i^{2)}$ and $F_i^{2)}$ are the second layer's ith neuron's in the spike time and its corresponding FT output, and $F_{max}^{2)}$ is the maximum value of the FT in the second layer.

Figure 4.7 shows the result of computing the range-Doppler map of a sample radar frame with a 2D S-FT architecture.

The method proposed above can also be applied to generate range-angle maps. Contrary to the range-Doppler map, the range-angle map simultaneously generates data from different channels. Thus, the range-angle map can either be calculated by computing one FT after the other, as in (4.46), or by combining both in a single pass,

$$Y = \sum_{n=0}^{N-1}\sum_{c=0}^{C-1} X_{nc}[C_n - i \cdot S_n][C_c - i \cdot S_c]. \tag{4.50}$$

As the operations are identical as for the 1D FT, we can create a single-layer SNN that replicates the functionality of the range-angle map by combining the 2 FTs into the single linear equation defined in (4.50)

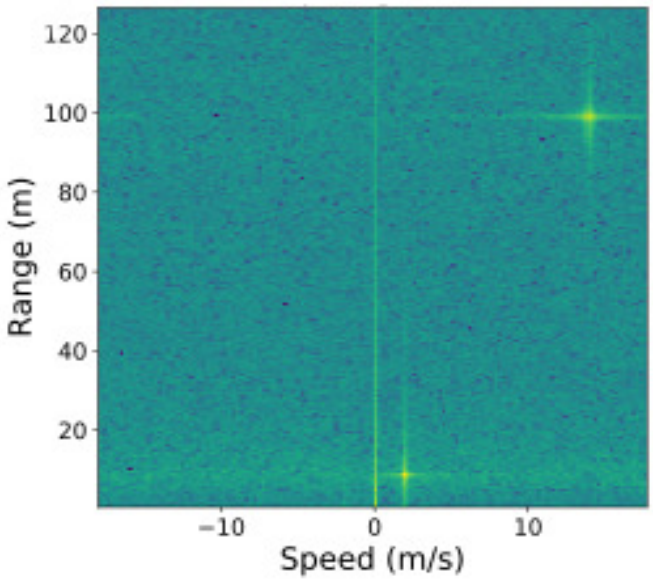


Figure 4.7: Example of computing the range-Doppler FT with the S-FT on a frame from simulated radar data. The frame contains three targets, at distances $R = \{0, 9, 100\}$ m and velocities $v = \{0, 2, 14\}$ m/s, respectively.

4.2.2 Computation of the fast Fourier transform

As described in section 2.3, the FFT is an efficient algorithm for implementing the FT by splitting the computation into smaller sums that can then be combined to obtain the same result as the *discrete Fourier transform* (DFT). Thus, the FFT is formed by consecutive layers where the summations and multiplications that lead to the final result are intertwined into a butterfly structure. In other words, the FFT rephrases the linear mapping (4.39) into

$$Y = W_{F_1} \cdot W_{F_2} \dots W_{F_L} \cdot X\,, \tag{4.51}$$

where L is the number of FFT layers.

By applying the principle of the TCBS model described in section 4.1.3, we can design an SNN that computes the FFT by chaining L layers connected by weights with the FFT coefficients.

Table 4.2: Comparison of different attributes of the TCBS when computing the DFT and the FFT

Parameter	S-FFT	S-DFT
Nº layers	$\log_4(N)$	1
Nº neurons	$2N \cdot \log_4(N)$	$2N$
Nº spike ops.	$8 \cdot 2N \cdot \log_4(N) + 2N$	$N \cdot 2N + 2N$
T_f	$2\tau_l$	$2\tau_l$
τ_f	$\tau_l \cdot (\log_4(N) + 1)$	$2\tau_l$

4.2.3 Compatibility with pre-processing algorithms

The computation of the FT with a time-coded SNN is an important step for achieving energy and time-efficient neuromorphic signal processing pipelines. To fully integrate the S-FT on neuromorphic hardware, it is crucial to apply the algorithms preceding the FT in the pipeline using compatible implementations. One of the most relevant processing stages prior to the FT when using *multiple-input multiple-output* (MIMO) sensors is beamforming, due to its high complexity and impact on the final result. Section 2.4.3 describes this operation and some popular versions. From a data processing perspective, adding a beamformer to a pipeline has two main implications:

1. The data arriving from each antenna is modified with the beamformer coefficients. Then the data from all antennas is fused and finally processed by the FT algorithm.

2. An adaptive beamformer calculates dynamically its coefficients by taking the input data and performing an optimization solution on it.

The first point is independent of which specific beamforming algorithm is chosen for optimizing the coefficients. When using a beamformer, the input data is linearly modified according to (2.60). If the output of the beamforming stage is fed to an FT, the final output can be calculated by combining (2.60) with (4.39) applying the Hadamard product of both weight matrices,

$$Y(t) = (W_B(t) \odot W_F) \cdot X(t)\,, \tag{4.52}$$

where $W_B(t)$ are the beamforming coefficients and W_F are the FT coefficients. In the case of non-adaptive beamforming, the coefficients stay constant over time, $W_B(t) = W_B$. The feasibility of the second point depends entirely on the chosen optimisation approach. Moreover,

fixed beamformers have pre-calculated coefficients that need not be dynamically updated, so their implementation is trivial.

Another algorithm often used before the FT is the windowing function. This function modifies the input with fixed weights linearly for removing ripples in the FT due to artifacts at the start and end of the sampling period. As a window function consists of a linear mapping of the input data, it can be integrated into the S-FT in the same fashion as with the beamforming

$$Y(t) = (W_W \odot W_F) \cdot X(t), \tag{4.53}$$

where W_W are the windowing coefficients.

(4.52) and (4.53) are compatible with the TCBS neuron model, as a single weight matrix can be obtained by multiplying W_F, W_W, and W_B. Thus, the window and beamforming functions can be integrated into the S-FT by modifying the input weights, so the weights applied in the update of the neuron during the charging stage, defined in (4.20), can be expressed as

$$W = W_F \odot W_W \odot W_B\,. \tag{4.54}$$

The weights obtained in (4.54) assume they are constant parameters. For the case of adaptive beamforming or online learning mechanisms, they have to be updated on each time step accordingly. The online modification of synaptic weights is a feature that not all neuromorphic chips possess, so the implementation of adaptive beamforming depends on the features of the specific neuromorphic chip.

4.3 Conversion of convolutional neural networks

Conversion is the process of learning the architecture and synaptic weights of neural networks using traditional machine learning methods prior to applying the learnt parameters to an SNN with the same structure. Conversion techniques aim at keeping the high accuracies obtained by traditional *artificial neural networks* (ANNs) while considerably reducing their energy footprint thanks to the sparse, event-based nature of neuromorphic systems. One of the most relevant aspects for improving the performance of neuromorphic hardware and providing gains in energy consumption is using as few spikes as possible during inference. Davidson and Furber found out in 2021 that 1.6 spikes per neuron are the theoretical threshold below which SNNs improve the performance of their ANN counterparts when using digital neuromorphic chips [DF21].

A potential application of the TCBS model introduced in this chapter is to convert DCNNs to the spiking domain. One benefit of the TCBS model is its temporal sparsity, as only one spike is necessary for representing a real number. Contrary to other temporal conversion techniques [RL18], TCBS-based conversion is lossless, as output spike times are mathematically equivalent to the output of the original analogue neuron, i.e., all inputs and their corresponding weights are computed in the spike-generation process. This statement holds as long as neurons' voltages remain below the threshold voltage during the silent stage. After this point, the neuron saturates, resulting in a dynamic range similar to rate-based conversion approaches (see fig. 3.3). Moreover, the model exploits properties of DCNNs for enhancing spatial sparsity, such as rectified activation functions, pooling operations, or convolutions. These operations contribute to activating a reduced number of neurons in the network.

Rectified linear unit

The rectified linear unit, or ReLU, is a nonlinear activation function that returns 0 if it receives any negative input and outputs the input directly if it is positive. It is the most commonly used activation function in ANNs due to its simplicity, efficiency, and performance. We can formally describe the ReLU as

$$y(a) = \begin{cases} a, & a > 0 \\ 0, & \text{otherwise} \end{cases} . \tag{4.55}$$

Assuming $x_{\min} = 0$, the TCBS model can replicate the behaviour of ReLU by constraining the output spike to occur only if the voltage at $t = t_s$ is positive. This is achieved by setting $u_{\min} = 0$ in (4.26), which yields

$$I_{\text{ext}} = \frac{u_{th}}{t_T - t_s} . \tag{4.56}$$

Therefore, all voltages that are smaller than zero at the end of the silent stage will not suffice to reach the threshold voltage during the silent stage and, hence, will not spike. This way, the ReLU activation function becomes a technique for increasing the spatial sparsity, as the neurons with a negative activation will stay silent.

Moreover, when applying the condition $x_{\min} = 0$ to (4.15), the coefficient β is reduced to zero.

Pooling

Pooling techniques are used in DCNNs for dimensionality reduction by splitting the data into small regions of interest and reducing each region to a value representing its members. One popular approach is *average pooling*, where the average value of the region's members replaces each neighbourhood,

$$y_i = \frac{1}{N} \sum_{n}^{N_N} x_n , \tag{4.57}$$

where N_N is the number of neighbours in each region of interest. Alternatively, *max pooling* reduces the output to the most dominant component in each region of interest by applying a *max* operation,

$$y_i = \max_{n \in N_N} \{x_n\} . \tag{4.58}$$

Generally, max pooling is preferred over average pooling as it yields better results for DCNN, and state-of-the-art ANNs based on the ReLU activation function apply this pooling method. Rate encoding conversion methods cannot convert the max pooling to a spiking version. They typically do post-processing of the spike rates for computing the max value [Rue+17] or implement instead average pooling [HE16]. Implementing max pooling using a time-coded conversion technique like TCBS is simpler than for rate-coded SNNs. As spike times represent the actual magnitude of the neuron output, the pooling operation can be obtained by using a winner-takes-all approach, i.e., a pooling neuron spikes as soon as an input spike arrives,

$$o(t) = 1, \qquad \text{if } i(t)_n = 1, \forall n \in N_P , \tag{4.59}$$

where $o(t)$ and $i(t)_n$ are the output and input spike trains, respectively, and N_P is the number of neurons on each pooling region of interest. To avoid further input spikes triggering the output, the pooling neurons go into a refractory state until the next data frame is sent.

Batch normalization

Batch normalization is a technique used to standardize the inputs to a network, either applied to the activations of a prior layer or directly to the inputs. Whereas normalization is applied dynamically during training, during inference, it uses a moving average of the mean and standard deviation of the batches seen during training. After calculating the activation of a neuron, we normalize it according to

$$\bar{a}_i = \gamma \frac{a_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \,, \tag{4.60}$$

where mu and $sigma$ are the batch's mean and standard deviation, and γ and β are scaling and shift factors. These parameters can be directly applied to the weights and bias of the network, resulting in

$$\bar{w}_{ij} = \gamma \frac{w_{ij}}{\sqrt{\sigma^2 + \epsilon}} \tag{4.61}$$

and

$$\bar{b}_i = \gamma \frac{b_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \,. \tag{4.62}$$

As the parameters obtained through batch normalization can be directly applied to the weights and bias using (4.61) and (4.62), the normalization can be easily applied to the converted SNN as well.

5

Analogue to spike encoder

The *temporal charge before spike* (TCBS) model introduced in chapter 4 computes the frequency spectrum of a signal by taking as input time-coded spikes that represent the signal's amplitude over time. This means the signal needs to be transformed from analogue voltages to spikes. The simplest approach to implement this mapping is to use an *analog-to-digital converter* (ADC), typically present in all sensor chips. After digitizing the signal, the sensor time series would be stored in memory before being converted to time-coded spikes using a dedicated software routine. This pipeline can be optimized if a dedicated circuit directly converts the incoming sensor voltage signal into spikes. Implementing this encoder circuitwould eliminate the delays and energy costs associated with the ADC, memory storage, and conversion software (see fig. 5.1).

This chapter describes the design of an *analogue-to-spike encoder* (ASE) for converting analogue signals into time-coded spikes to compute the frequency spectrum with the *spiking Fourier transform* (S-FT) on a digital neuromorphic chip.

5.1 Working principle

The ideal behaviour of the encoding circuit is to provide time-coded spikes that follow the linear mapping in (4.3), which consists of a membrane voltage that charges linearly over time. Due to its simplicity and low consumption, the designed ASE circuit is based on the *leaky integrate-and-fire* (LIF) neuron model, which implements a biologically inspired logarithmic mapping between input stimuli and output spike times. The circuit is single-input, single-output, i.e., the sensor voltage is fed to the input channel, and the output is formed by the resulting spike train. The ASE samples the input signal by integrating the sensor current I during the sampling time T_S, defined by a sampling clock CLK_S provided by the sensor circuit, where the sampling rate is $f_s = 1/T_S$. Assuming small changes in the current I during the sampling time, the circuit gets charged following the general LIF equation

$$u(t) = IR\left(1 - e^{t/\tau_m}\right), \tag{5.1}$$

where R and τ_m are the resistor and time constant of the LIF model, respectively. Alternatively, if a voltage signal $V(t)$ drives the input, which is the case of sensors like automotive radar, we can express the membrane voltage as

$$u(t) = V(t)(1 - e^{-t/\tau_m}) \tag{5.2}$$

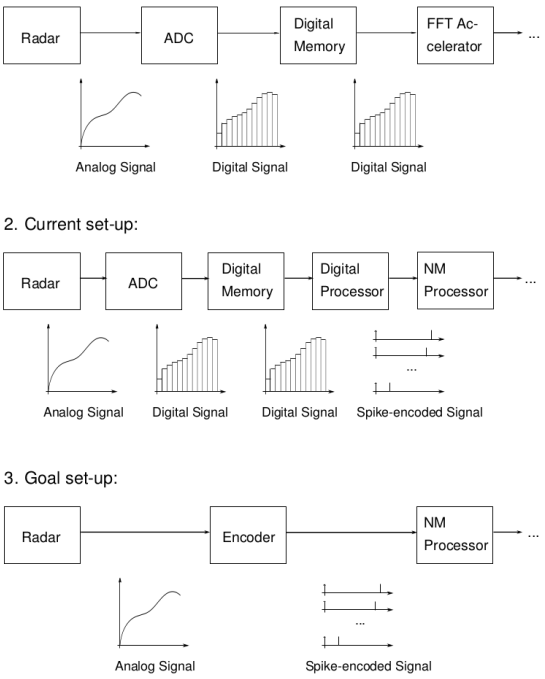


Figure 5.1: Top: Main blocks for computing the FFT of a radar analogue signal. Middle: Simple approach for replacing the FFT by the S-FT. Bottom: Final system diagram after implementing the analogue-to-spike encoder.

The reader can refer to appendix B for details on how to derive (5.1) and (5.2).

Figure 5.4 shows the temporal dynamics of the model. For a given sample, the charging process defined in (5.2) starts at $t = t_0$, and the output spike occurs at $t = t_s$ when the membrane voltage $u(t)$ reaches the threshold voltage u_{th}. At that point, the spike output digital signal, SPK, is set to 1, and the circuit enters a refractory state until the end of the sampling period $t_0 + T_S$. We assume that the input is a voltage source whose dynamics are much slower than the sampling time T_S. Thus, we can apply the relationship $I = V_S/R$, where V_S is the voltage $V(t)$ for $t_0 < t < t_0 + T_S$. The spike time t_s is obtained by replacing $u(t_s) = u_{\text{th}}$ in (5.1), which yields

$$t_s = -\tau_m \ln \left| 1 - \frac{u_{\text{th}}}{V_S} \right|, \tag{5.3}$$

which defines the dynamics of the ASE. The main features of this model are:

- The spiking time of the ASE is inversely proportional to the voltage fed to the encoder.
- The output t_s follows a logarithmic function that can be approximated by a hyperbolic function for $u_{\text{th}} <<< V_S$.
- From the previous point, the parameter u_{th} determines the likelihood of the ASE with a hyperbolic mapping of the voltages into spike times.
- The parameter τ_m defines the time scale of the output spike times.

The working principle of the Fourier transform assumes constant time intervals between samples. To replicate its functionality with the ASE and the S-FT, the refractory state of the ASE after each spike must maintain a regular sampling rate.

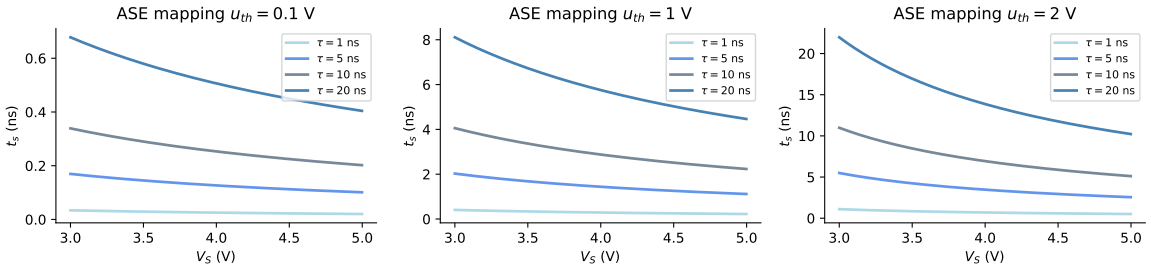


Figure 5.2: Mapping of voltages to spike times with the introduced ASE for different values of u_{th} and τ_m. The used voltage range was $[3,5]V$.

5.1.1 Parameter tuning

The design of the ASE involves carefully tuning its parameters, namely τ_m and u_{th}, and defining appropriate time ranges for the equivalent linear encoder. One of the goals of this tuning process is to achieve optimal linearity in the mapping between the input and output of the ASE. In other words, to make the ASE model (5.3) closely resemble the linear mapping in (4.3). An additional goal of the tuning process is to maximize the output dynamic range of the encoder so the resolution is maximal for the chosen $t_{\max}$. Figure 5.2 shows how τ_m and u_{th} affect the mapping of voltages to spike times.

Linearity error

Let us call $f(V_S)$ and $g(V_S)$ the ASE logarithmic function and the S-FT linear encoding function given by (5.3) and (4.3), respectively. We call linearity error ε_L the difference between both functions for the given voltage range $\Delta V_S = [V_{S\min}, V_{S\max}]$. If we express this error as a mean absolute difference, and assuming $f(V_S)$ and $g(V_S)$ are continuous functions, ε_L can be calculated as an integral over the input range

$$\varepsilon_L = \frac{1}{\Delta V_S} \int_{V_{S\min}}^{V_{S\max}} |f(V_S) - g(V_S)| dV_S \,. \tag{5.4}$$

In discrete form, (5.4) can be expressed as

$$\varepsilon_L = \frac{\sum_{V_S} |f(V_S) - g(V_S)|}{\Delta V_S} \,. \tag{5.5}$$

We normalize the result with the modulus of the middle value for the input voltage, so the result is agnostic of the voltage magnitude across experiments,

$$\tilde{\varepsilon}_L = \frac{\epsilon_L}{\frac{|V_{S\max}| - |V_{S\min}|}{2}} \,. \tag{5.6}$$

Assessing linearity with Taylor series

The encoding dynamics in (5.3) are modelled as a logarithmic relationship between the spike time t_s and the input voltage V_S. We can simplify the model by expanding (5.3) using Taylor series about $V_S = V_0$, which yields

$$t_s = -\tau log(1 - \frac{u_{th}}{V_0}) + \sum_{n>0}^{\infty} \tau \frac{((u_{th} - V_0)^{-n} - (-V_0)^{-n})(V_S - V_0)^n)}{n} \,. \tag{5.7}$$

When using the first polynomial, (5.7) converges to the linear expression

$$t_s = -\tau log(1 - \frac{u_{th}}{V_0}) + \tau \frac{u_{th}}{V_0(u_{th} - V_0)}(V_S - V_0) + \mathcal{O}^2\,, \tag{5.8}$$

where $\mathcal{O}^2$ is the error of the first polynomial expansion. Thus, (5.8) converges to a linear function for small values of $\mathcal{O}^2$, which can be achieved by fulfilling the condition

$$|\frac{u_{th}}{V_S}| << 1\,. \tag{5.9}$$

In other words, the ASE model shows a linear behaviour when the values of u_{th} are very small compared to the input voltage V_S.

Time efficiency

Let us define the dynamic range t_{spk} of the output of the ASE as the difference between the maximum and minimum spiking times, $t_{\max}$ and $t_{\min}$ respectively,

$$t_{\text{spk}} = t_{\max} - t_{\min}\,. \tag{5.10}$$

For a given working time range t_0 to $t_{\max}$, we define the time efficiency of the ASE as the amount of the time range that is effectively used for encoding input into spikes

$$\mu_t = \frac{t_{\text{spk}}}{t_{\text{spk}} + t_{\text{wait}}}\,, \tag{5.11}$$

where t_{wait} is the time needed for charging the ASE when input takes its maximum value $V_{S\max}$, which yields the fastest spike time $t_{\min}$,

$$t_{\text{wait}} = t_{\min} - t_0\,. \tag{5.12}$$

Assuming $t_0 = 0$, we can combine (5.11) with (5.10), which results in the expression

$$\mu_t = \frac{t_{\max} - t_{\min}}{t_{\max}}\,. \tag{5.13}$$

For the condition $u_{th}/V_S << 1$, if we apply Taylor series (5.8) for calculating $t_{\min}$ and $t_{\max}$ we find that the dynamic range depends only on the input voltage range

$$\mu_t \approx \frac{V_{S\max} - V_{S\min}}{V_{S\max}}\,. \tag{5.14}$$

For larger values of u_{th}, the time efficiency increases proportionally with the ratio $u_{th}/V_{S\min}$. Thus, once the voltage limits are fixed, the only way of improving τ_t is by increasing u_{th}.

Encoding loss optimization

By combining the linearity error ϵ_L and the time efficiency μ_t, we create a loss term L_{ASE} for optimizing the parameter tuning,

$$L_{\text{ASE}} = \alpha\varepsilon_L + \beta\mu_t\,, \tag{5.15}$$

where α and β are the relative weights given to ϵ_L and μ_t, respectively. Tuning the ASE parameters will consist in finding the values of $[u_{th}, \tau, t_{\min}, t_{\max}]$ that minimize L_{ASE}. Depending

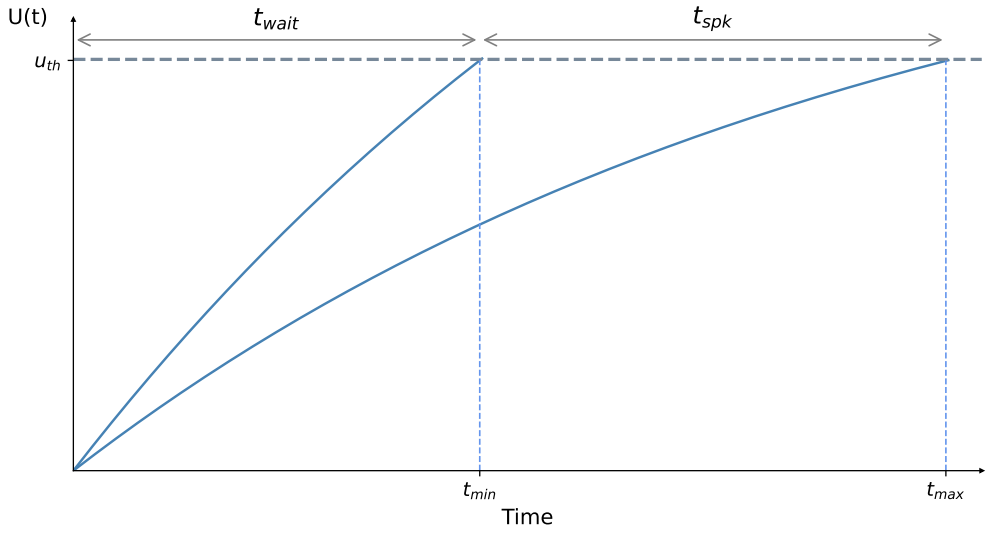


Figure 5.3: Voltage membrane curves for $V_S = V_{S\min}$ and $V_{S\max}$. Each curve has a spike time of $t_s = t_{\max}$ and $t_{\min}$, respectively. These limit values for the spike times define the dynamic range t_{spk}, as well as the waiting time t_{wait}.

on the nature of the experiments, we can fix some of these parameters beforehand. For example, in an implemented circuit, τ_m is usually fixed, as it depends on the LIF capacitor and resistor. On the other hand, the threshold voltage is easier to adjust. The search of values for $t_{\min}$ and $t_{\max}$ is constrained to values near the maximum and minimum spike times obtained for the logarithmic ASE at the limits of the input range, t_{ASEmin} and t_{ASEmax},

$$\begin{aligned} t_{\min} &= t_{\text{ASEmin}} \cdot k_{t1}\,, \\ t_{\max} &= t_{\text{ASEmax}} \cdot k_{t2}\,, \end{aligned} \tag{5.16}$$

where k_{t1} and k_{t2} are the scaling factors for the minimum and maximum spiking times, respectively. From (4.3), the value of $t_{\min}$ serves as an offset to the linear mapping, whereas $t_{\max}$ is used for fixing the slope of the curve. We can perform the minimization of L_{ASE} with an optimization algorithm. Section 6.2 shows detailed results for this optimization problem with a *differential evolution* algorithm.

The following steps are a strategy for applying the optimization algorithm for a given voltage encoding problem:

1. Fix the range of the input voltage $[V_{S\min}, V_{S\max}]$.
2. Select τ range based on the desired time magnitudes, e.g., $ns, \mu s, ms$.
3. Fix the bounds of k_{t1} and k_{t2} for adjusting $t_{\min}$ and $t_{\max}$, respectively.
4. Obtain $[u_{th}, \tau, t_{\min}, t_{\max}]$ by applying the chosen optimization algorithm the minimization of $L(\cdot)$.
5. Chose $\hat{\tau} \approx \tau$ with the available electronic components.
6. Recalculate $[u_{th}, t_{\min}, t_{\max}]$ for $\hat{\tau}$.

5.1.2 Integration on a neuromorphic pipeline

Figure 5.4 shows the block diagram of the general application of the ASE. Conceptually, the ASE is a block that generates spikes as a function of the sensor voltage $V(t)$ during the sampling period. The spikes are then sent to neuromorphic hardware, which will process the spikes for an application-specific purpose. If the neuromorphic hardware is a digital chip, the spike train is translated into a digital signal SPK formed by binary pulses. The clock signal CLK_S defines the sampling time Δt. This signal can be provided by the sensor itself (as in fig. 5.4), by the neuromorphic hardware, or by an external clock. If the ASE is connected to a digital chip, both components require a communication interface with a synchronization neural clock CLK_N. The neuromorphic chip will typically provide this interface, determining the maximum error in estimating the spike time. Assuming negligible transmission delays, the spike time error ϵ_t is the difference between the spike time t_s and the actual time of the estimated spike time $\hat{t}_s$

$$\epsilon_t = \widehat{t}_s - t_s \,. \tag{5.17}$$

The maximum error is the period T_N of the neural clock CLK_N. Moreover, the ASE generates the flag S_{refr}, which indicates a refractory state in the encoder. At the start of the refractory period, the ASE voltage $u(t)$ drops to zero and stays unaffected by the input voltage until the end of this period. Figure 5.4 shows the temporal dynamics of the digital signals generated by the ASE in relationship with the reference clocks and the sensor voltage.

5.2 Electric design

Figure 5.5 contains an electric diagram of the implementation of the ASE. The encoder integrates the sensor voltage $V(t)$ over time using an R–C circuit replicating the LIF neuron model dynamics in (5.2). The values of the resistor R_{LIF} and capacitor C_{LIF} are chosen based on the ranges of the input voltage and for mapping all possible spike times during the sampling. The output voltage $u(t)$ of the LIF integrator is connected to a comparator *C1* that changes to a HIGH state or digital 1 when $u(t)$ is higher than the threshold voltage u_{th}. The synchronization of *C1* with the digital chip is done with the D-type flip-flop *A1*, which is set to HIGH with the output of *C1* via the preset signal PRE. On the next rising edge of the neural clock CLK_N after PRE $= 1$, the spike signal SPK is set to 1, and the digital chip reads the spike, responding with the onset of the reset of *A1*. The HIGH state of *C1* triggers as well the refractory period by setting the flag $S_{\text{refr}} = 1$ with a second D-type flip-flop *A2*, which triggers two switches: the first switch opens the connection between the sensor and the LIF integrator, and the second switch discharges the capacitor C_{LIF}. The reset of S_{refr} takes place on the rising edge of the sampling clock CLK_S, which marks the start of a new sample. The proposed circuit design assumes that the sensor voltage $V(t)$ will be able to produce a spike for every sampling time, i.e., the threshold u_{th} will always be reached during the sampling period T_S.

The comparator *C1* requires two auxiliary components: A voltage divider R_{div} that provides the desired u_{th}, and a pull-up resistor R_{pull} that enables the HIGH state once the threshold is crossed. The existence of R_{pull} depends on the specific model used for implementing *C1*. Moreover, *C1* needs to be in a LOW state for generating the spike flag *SPK*, as PRE works with negated logic. The boundaries of the voltage threshold u_{th} are $[0, V_{cc}]$ due to the properties of the comparator *C1*. The sensor voltage $V(t)$ cannot take values below zero, as it would force the input of *C1* below its working range. These limits can be easily adjusted by shifting the source and sink of *C1* below the minimum possible value of $V(t)$, e.g., by not having a common ground between the sensor and *C1*.

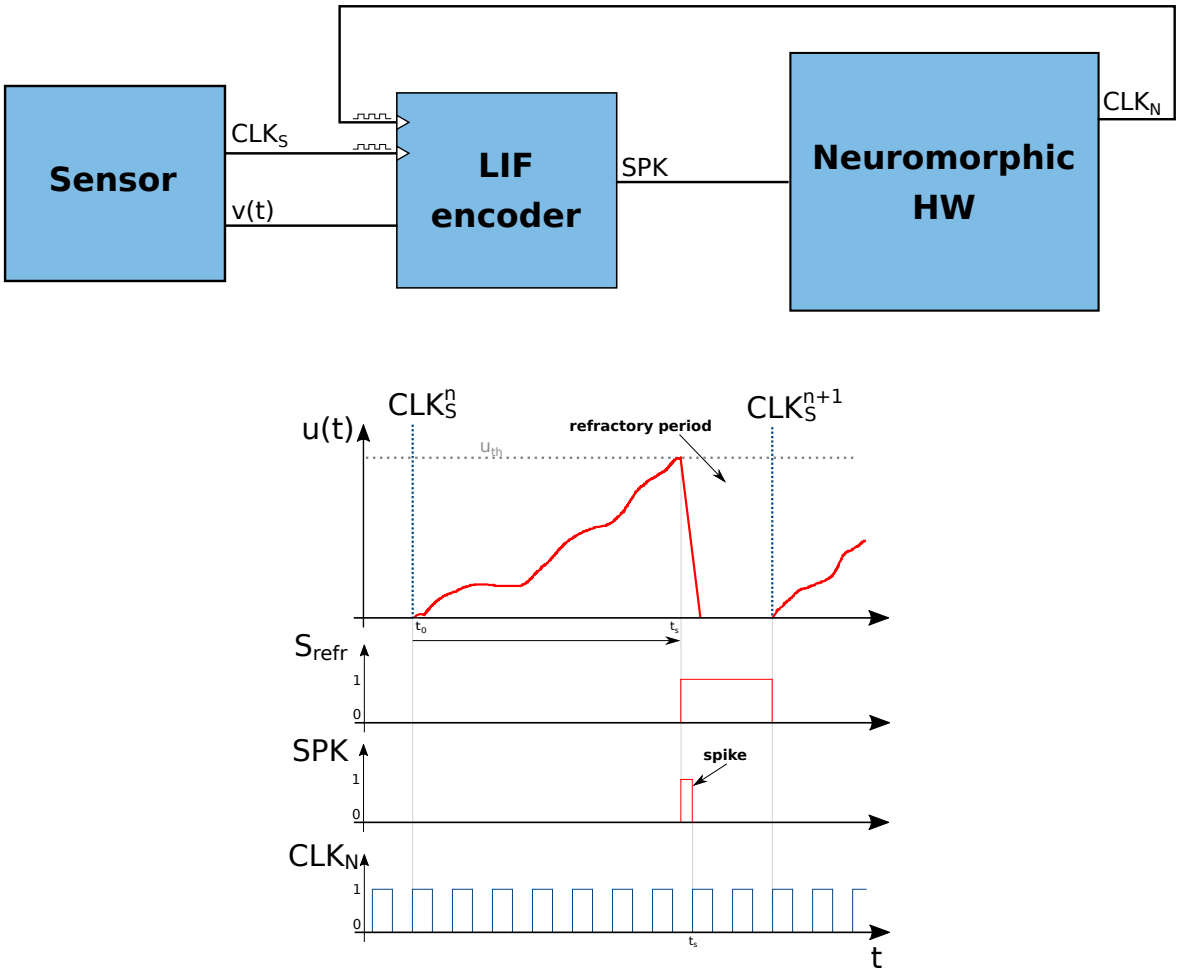


Figure 5.4: Top: Block diagram of the LIF-based ASE circuit and its peripherals. Bottom: Plot over time of the voltage $u(t)$, refractory period flag S_{refr}, and spike output SPK of the ASE for one cycle of the sensor sampling clock CLK_S, as well as the sampling clock of the neuromorphic device CLK_N. The sampling time for the n^{th} sample starts at t_0, and a spike is generated at t_s when the voltage reaches the threshold value u_{th}. The circuit stays in refractory state until the following clock cycle CLK_S^{n+1} starts.

5.2.1 Energy estimations

As one of the main motivations of the ASE is to minimize the consumed energy, it is paramount to obtain an estimation of the power dissipation of the proposed circuit. In the remainder of this subsection, the methods used to estimate the power dissipation of each component are explained. The method differs for each component, as in some cases the information can be obtained from the component datasheet, in other cases the most suitable approach is to estimate the dissipation using circuits theory, and in other cases a calculation via SPICE simulation is the best option. These estimations assume that the losses due to the wiring and coupling effects are negligible. Table 5.1 summarises the power dissipated by each component and their energy consumption per sample.

The energy consumption detailed in this section is an estimation based on models of ideal components, assuming no losses in the wiring. A more precise energy profile of the ASE requires an optimized implementation using ad-hoc comparator and flip-flop circuits and a specific integrated circuit design, which is out of this thesis' scope. The interested reader may check optimized biological circuits, which can yield a consumption in the order of fJ per spike [Sou+17].

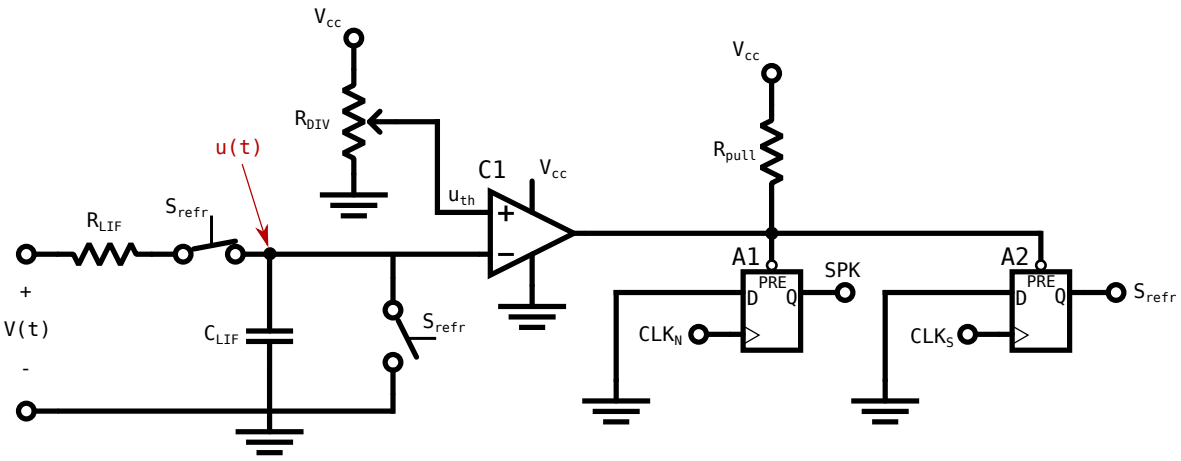


Figure 5.5: Schematic of the designed ASE circuit. The sensor is represented by a variable voltage $V(t)$ and a very high impedance. The dynamics of the ASE are determined by the resistor R_{LIF} and the capacitor C_{LIF}. The comparator *C1* checks the crossing over the threshold voltage u_{th} of the ASE voltage $u(t)$. The output of *C1* triggers the digital spike SPK on the flip-flop *A1*. The following rising edge on the neuromorphic clock CLK_N resets the encoder output voltage. The flip flop *A2* forces a refractory period from the appearance of a spike SPK until the next rising edge on the sampling clock CLK_S.

Flip-flops

The calculation for the *RS* flip-flops is based on the dynamic power-dissipation capacitance C_{pd} provided in the corresponding datasheet. We can calculate the dynamic energy consumption per switching operation according to [Sar97]. Adding the dissipation due to static power consumption we obtain

$$E_{RS} = C_{pd} V_{cc}^2 + V_{cc} I_{cc} T \,, \tag{5.18}$$

where V_{cc} is the supply voltage of the circuit, I_{cc} is the leakage current, and T is the sampling period. For the selected flip-flop, the SN74HCS72, $C_{pd} = 10pF$ and $I_{cc} = 0.1\mu A$. The losses due to leakage currents are negligible for the high sampling frequencies of the circuit, as T is very small.

Comparator

The comparator is a circuit based on an operational amplifier laid out in an open loop. Theoretically, the op-amp has very high impedance, thus the power dissipation in the input is negligible. The power dissipation in the output is due to two factors: The quiescent current I_q that flows from one power input to the other, and the output current I_o that flows outside of the amplifier, sometimes referred to as sink current. Thus, the total dissipated power P_C in the comparator is calculated as

$$P_{\text{comp}} = I_q(V_{cc}^+ - V_{cc}^-) + I_o(V_{cc}^+ - V_o), \tag{5.19}$$

where V_o is the output voltage. Although in an ideal case the voltage difference $(V_{cc}^+ - V_o)$ is zero, the actual value is non-zero and is typically provided by the manufacturer. From circuits theory, the total dissipated energy E_{comp} for one sample is

$$E_{\text{comp}} = P_{\text{comp}} T \,. \tag{5.20}$$

For the MAX951 ultra-low power comparator, the quiescent current is $I_q = 7\mu A$ (called supply current), $(V_{cc}^+ - V_o) = 0.4V$, $I_o = -1.8mA$ (low out) OR $2.0mA$ (high out). It is relevant to note that manufacturers often do not focus on creating precise dissipation measurements. Thus, a real implementation of the component is necessary to have a clear value of the dissipated power.

Table 5.1: Energy consumption per sample and dissipated power for each of the components of the ASE, for a constant input voltage $V(t) = 2.5V$, supply voltage of $V_{cc}^{+} - V_{cc}^{-} = 10V$, and a sampling rate $f_s = 300kHz$.

Component	nJ/sample	mW	Ratio(%)	Ref.
C_{LIF}	$5.3e-4$	$1.6e-4$	0.01	$1nF$
R_{LIF}	0.07	0.02	2.20	$50k\Omega$
C1	2.6	0.8	82.02	MAX951
A1	0.25	0.075	7.89	SN74HCS72
A2	0.25	0.075	7.89	SN74HCS72
Switches	$3.4e-5$	$10e-6$	<0.01	ADG719
TOTAL	**3.17**	**0.97**		

Resistor

The power dissipated by the resistor can be calculated using Joule's law, i.e., $P = IV$. The charging of the capacitor dominates the voltage difference at the resistor until the spike happens. We assume that $V_m = 2/3V_{th}$ for an average spiking time of $t_s = T/2$. After the spike, the switch closes and the resistor dissipates the whole voltage provided by the source. We can then calculate the energy dissipated by the resistor as

$$E_R = \frac{V_{in}^2}{R_{in}} T/2 + \frac{1/3V_{in}^2}{R_{in}} T/2 = \frac{4V_{in}^2}{3R_{in}} T/2 \,. \tag{5.21}$$

The dissipation that happens after the spike can be easily avoided by adding a switch connected to the supply that opens during the refractory period. For this layout, the second term in (5.21) becomes zero and the energy dissipated takes the form

$$E_R = \frac{TV_{in}^2}{6R_{in}} \,. \tag{5.22}$$

From (5.22), we can observe that the energy dissipation follows a quadratic relationship with the input voltage. Therefore, reducing the voltage range is the most effective approach for minimising the energy consumption of the circuit.

6

Implementation and experiment results

The preceding chapters introduced the two main contributions of this work: the *temporal charge before spike* (TCBS) neuron model and the *analogue-to-spike encoder* (ASE). These innovations can replace the *Fourier transform* (FT) and *analog-to-digital converter* (ADC) stages in a radar signal processing pipeline. Minor contributions also include the replacement of the *ordered-statistics CFAR* (OS-CFAR) algorithm with a sparse *spiking neural network* (SNN) for object detection and the application of the TCBS model for the conversion of *deep convolutional neural network*s (DCNNs). This chapter deals with implementing these models and their assessment in various experiments. The *spiking Fourier transform* (S-FT) and the *spiking OSCFAR* (S-OSCFAR) implementations were executed on neuromorphic hardware, while the ASE was implemented on a prototype board controlled by a dsPIC microcontroller. The implementations were evaluated independently using data from real-world automotive scenarios and electric signals generated in the laboratory, respectively. The evaluation of the TCBS for conversion of DCNNs is detailed in appendix D.

The primary objective of the experiments was to validate the potential of the proposed algorithms and neuron models as substitutes for traditional *frequency-modulated continuous-wave* (FMCW) radar signal processing techniques. The results showcase the errors these SNNs introduced across various scenarios and parameter configurations. Additionally, the experiments evaluated the performance of these models in terms of energy consumption and time latency. It is essential to note that the limited energy measurement capabilities of the employed neuromorphic boards by the time the experiments were run constrained the assessment of the energy consumption. Hence, the energy profiling was limited to estimations.

6.1 Spiking Fourier transform

This section describes the implementation of the neuron model introduced in chapter 4 and presents the outcomes of the experiments conducted to validate its performance. The S-FT was realized on neuromorphic hardware and tested in automotive radar scenarios. The main purpose of the experiments was to validate the applicability of the S-FT as a potential replacement for the *fast Fourier transform* (FFT). This involved ensuring its output exhibits minimal errors compared to an ideal FT.

The first experiment ran the S-FT on data obtained from an automotive radar simulator provided by Infineon AG. This tool offers more control over the experiment, facilitating the assessment of the algorithm's performance in aspects like sensitivity to noise or compatibility with a target detection algorithm. Complete knowledge of target locations allows the

measurement of the accuracy of target detection pipelines based on the S-FT. Namely, the OS-CFAR and *density-based spatial clustering of applications with noise* (DBSCAN) algorithms were applied to the output of the S-FT, and a detection score was assigned to the final output. The detection accuracy was compared with the output of the same algorithms applied to the FFT, which made it possible to analyse the real impact of the error in the frequency spectrum on the entire processing pipeline.

A second set of experiments ran the S-FT on five real-world scenarios sensed with an FMCW radar sensor. These experiments let us analyse whether the performance of the S-FT drops in realistic scenarios. The lack of information about the true output limits the scope of this experiment to the analysis of error metrics regarding an ideal FT, as there is no available information about the location of the targets in the scene. In other words, the experiment does not evaluate the accuracy of object detection tasks. It only assesses the deviation of the S-FT compared with an ideal FT for real-world data.

6.1.1 S-FT Implementation

The various implementations of the S-FT share a common operational principle tailored to specific platforms. The neuron model employed, the TCBS, is an adaptation of the *leaky integrate-and-fire* (LIF) model with two different behaviours, called silent and spiking stages (see section 4.1). The change from one to the other occurs at the end of the silent stage, i.e., at $t = t_s$. The threshold voltage u_{th} that determines when spikes occur is calculated as

$$u_{th} = 0.25\alpha_{\text{th}} \sum_{n}^{N} w_{0n} t_s \,, \tag{6.1}$$

where w_{0n} represents the nth coefficient of the zero-mode FT bin, and t_s is the number of timesteps in the charging stage. The reduction coefficient α_{th} falls within the $[0, 1]$ range and is determined empirically. Even though the theoretical maximum power of the S-FT leads at t_s to a voltage $\max_{t=t_s} u(t) = u_{th,\alpha_{\text{th}}=1}$, this value is never reached when using real-world data. To adjust to the reduced working range, α_{th} lowers the maximum value of $u(t_s)$ that can be identified, and beyond this value, all voltages are truncated. Its tuning is a trade-off between resolution and sensitivity to high bin intensities. A lower value of u_{th} corresponds to a higher output resolution and a narrower dynamic range. However, very low values of u_{th} truncate FT bins with high intensities. Assuming the silent and spiking stages have the same duration, the value of the offset current for the spiking stage is determined as

$$I_{\text{ext}} = \frac{2u_{th}}{t_s} \,, \tag{6.2}$$

which is the current necessary to provoke a spike at $t = t_{\max}$ for $u(t_s) = u_{\min}$. For the special case of the S-FT, the minimum voltage is set to $u_{\min} = -u_{th}$, as the output of an FT is symmetric around zero.

Traditional processor

The initial implementation of the S-FT was conducted on a conventional processor. This choice of platform offered the advantage of greater flexibility in parameter modification and the exploration of various features without the overhead of adapting to the architecture of a neuromorphic chip. To facilitate the study of the S-FT's nature, this implementation aimed to closely emulate the behaviour of neuromorphic implementations.

The PC implementation was coded in Python using the NumPy library directly. The main motivation was to have a versatile and easy-to-modify implementation. Hence, higher-level libraries like Brian2 or NEST were discarded. The S-FT was implemented as a class that took the input encoded as spike times and iteratively updated its state for the number of time steps, similar to how a digital neuromorphic chip would do it. The class returns spike times as output, as well as the value of the neurons' voltages for debugging purposes.

Loihi

The implementation of the S-FT in Loihi is based on the current-based LIF neuron model, which is the basic model implemented in the Loihi core. The voltage u of a given neuron changes stepwise according to

$$u[t] = u[t-1] + i[t]\Delta t - (1-\alpha_{\text{lif}})u[t-1], \tag{6.3}$$

where i, Δt, and α_{lif} are the current, simulation step time, and the decay rate, respectively. The neuron's current is obtained from the N input spike trains to the neuron,

$$i[t] = i[t-1]\sum_{j=0}^{N} w_j s_j[t] + b\,, \tag{6.4}$$

where $s_j[t]$ defines the spike train of the jth input, and b is the bias function of the neuron.

The model defined by (6.3) and (6.4) was modified to implement the S-FT neuron model. The TCBS neuron was implemented as a standalone model based on the current-based LIF. Implementing the silent stage is trivial, as the only modification it requires is a decay set to $\alpha_{\text{lif}} = 0$. Implementing the spiking stage requires two changes at the end of the silent stage ($t = t_s$):

1. The contribution from the inputs needs to be inhibited from this point, by setting $i[t_s] = 0$ and blocking the arrival of further spikes.

2. The neuron's bias has to be fixed to the value I_{ext}, which is determined using (6.2).

The model was developed in the NxSDK library from Intel Labs for the Loihi 1 board. The library has a higher-level layer where versions of the most popular neuron models can be created. On top of this, a sequential neural interfacing process (SNIP) was programmed to change between the two working stages. SNIPs interact with the neuron compartments for, among other tasks, changing between operation phases [Dav+21]. For the Loihi 2 board, the model was developed with the Lava framework. The TCBS was implemented as a new class that inherits its core functionality from the class that implements the basic LIF neuron model. The transition between stages was implemented using a microcontroller code (ucode) script that takes two new parameters, *t_half* and *charging_bias*. When the simulation reaches *t_half*, the current is fixed to the value of *charging_bias*.

SpiNNaker 2

The S-FT was implemented in the SpiNNaker 2 board based on the native LIF model. The model includes the parameter T_silent, which determines the behaviour change from the silent to the spiking stage. For simulation times $t < T\_\text{silent}$, the neuron is charged according to

$$\begin{cases} u[t] & = u[t-1] + I_{\text{syn}}[t] \\ I_{\text{syn}}[t] & = I_{\text{syn}}[t-1] + I[t] \end{cases}, \tag{6.5}$$

where $u[t]$, $I_{\text{syn}}[t]$, and $I[t]$ are the membrane voltage, neuron synaptic current and input at time step t, respectively. A neuron that follows (6.5) accumulates all inputs that took place at previous time steps and modifies the membrane voltage v based on a linear combination of those inputs. For simulation times $t \geq T\_\text{silent}$, the neuron is charged according to

$$u[t] = u[t-1] + I_{\text{offset}}[t], \tag{6.6}$$

where $I_{\text{offset}}[t] = I_{\text{ext}}$ is the offset current of the spiking stage and is determined using (6.2). During this stage, the input information stored in I_{syn} is ignored. (6.1) determines the threshold voltage value. After generating a spike, the neuron starts a long refractory period that prevents the occurrence of further spikes.

6.1.2 Experimental results

The experiments with the S-FT assessed its performance in different scenarios and how its output compared with that of a standard FFT. Initial experiments were performed on simulated data with the knowledge of the true values of the recorded scenarios. Knowing the number of targets in the scene and their exact positions is paramount for evaluating target detection tasks and calculating the actual precision of the processing pipeline. Further experiments were conducted on real data from street scenarios with an FMCW radar. These scenarios lack knowledge of the actual position of the targets, and their purpose was to validate the behaviour of the S-FT in real-world scenarios by comparing its output with that of an FFT accelerator.

Experiments on simulated data

The first batch of experiments for the S-FT was conducted on simulated data for an FMCW radar sensor obtained from a library developed by Infineon AG. The dataset was split into four sets, the first two belonging to scenarios containing a car and the last two containing a pedestrian. Moreover, half of the scenarios for each target type were simulated under low-noise conditions, and half were simulated under high noise. Thus, the dataset was comprised of the *car_lownoise*, *car_highnoise*, *pedestrian_lownoise*, and *pedestrian_highnoise* subsets. Each subset comprises 200 scenes containing one target each, adding up to 800 scenes. The noisy scenes included several sources of noise: analog front-end noise, thermal noise, phase noise, and ADC non-linearities, saturation, and noise.

Table 6.1 shows the main specifications of the radar for the simulated scenarios. The range resolution and maximum range were calculated using (2.21) and (2.22), yielding $\Delta R = 0.25$ m and $R_{\text{max}} = 126.4$ m, respectively. The car targets were simulated at incremental distances in the range $[2.8, 113.8]$ m, with a radar cross-section $\sigma_{\text{car}} = 15 dBsm$ for all scenes. The pedestrian targets were simulated at distances in the range $[1.6, 63.2]$ m, with $\sigma_{\text{ped}} = 0$ dBsm for the low-noise scenes, and $\sigma_{\text{ped}} = 5$ dBsm for the noisy scenes. The radar cross-section had to be increased in the *pedestrian_highnoise* subset, otherwise, the target could not be distinguished from the noise.

Table 6.1: Specifications of the radar used in the simulated scenarios. N is the number of samples in a chirp.

Parameter	Value
f_0	76 GHz
Δf	607.7 MHz
ΔT	40.96 μs
N	512

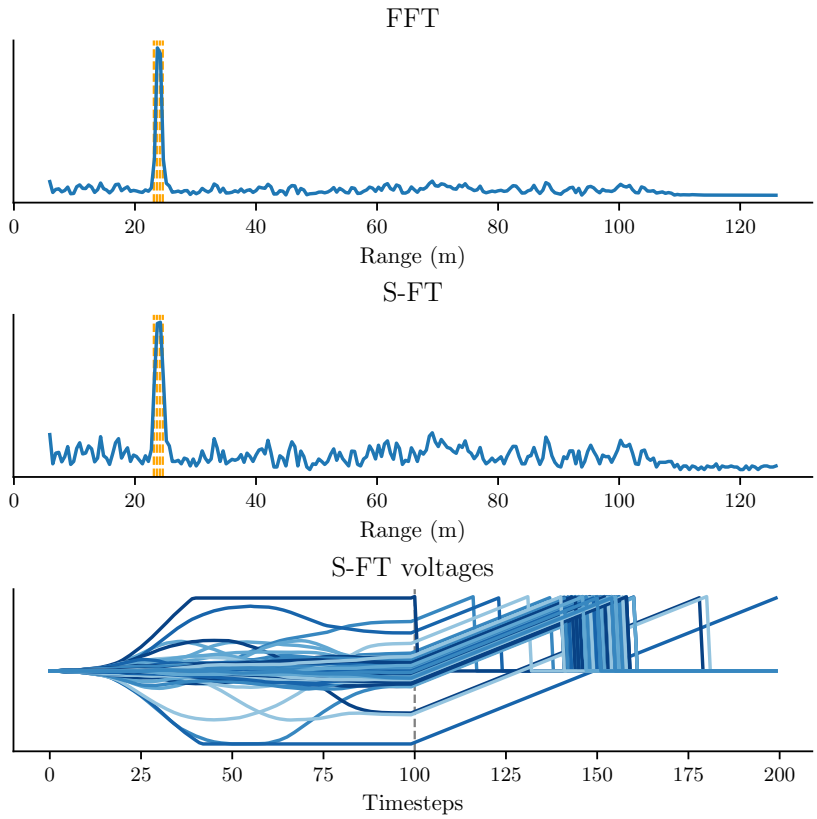


Figure 6.1: Computation of a chirp from the *car_highnoise* subset with the FFT (top) and the S-FT (middle). The orange markers show the cluster obtained after applying an OS-CFAR and DBSCAN algorithms to the output of the frequency spectrums. The bottom plot shows the evolution of the membrane voltage of the different neurons of the S-FT for the whole simulation, where the starting voltage is $u(t = 0) = 0$ V for all neurons. The dashed gray line at timestep 100 indicates the change from the silent stage to the spiking stage.

The generated dataset contained the raw data of the radar and the labels with the real distance of the targets in the scenes. Two different pipelines processed the data: a standard pipeline and a spiking pipeline. Both pipelines included a pre-processing stage with a Hann window and a mean-shift operation. The standard pipeline computed the FFT using the NumPy library, and the spiking pipeline encoded the data into spikes and computed the S-FT. The spiking pipeline was run for different numbers of simulation steps and was compared with the FFT output by computing the *root-mean-square error* (RMSE),

$$E = \sqrt{\frac{\sum_{n=0}^{N_{\mathrm{FT}}-1}(\hat{y}-y)^2}{N_{\mathrm{FT}}}}, \tag{6.7}$$

where N_{FT}, $\hat{y}$, and y represent the number of bins in the frequency spectrum, the estimated output, and the ground-truth output, respectively. The offset bins and the negative sides of the frequency spectrums were ignored before computing (6.7), which reduced the size of the FTs to

$$N_{\mathrm{FT}} = \frac{N}{2} - n_{\mathrm{offset}}, \tag{6.8}$$

where n_{offset} is the number of offset bins that are ignored in the post-processing of the frequency spectrum. The offset level of the input signal, which holds no information about the targets, dominates these bins. It is produced by the reflections of the sensor casing and the car bumper, as well as by the bias introduced by the thermal noise in the sensor. The results shown in this section were obtained for $n_{\mathrm{offset}} = 12$. To make results comparable, the

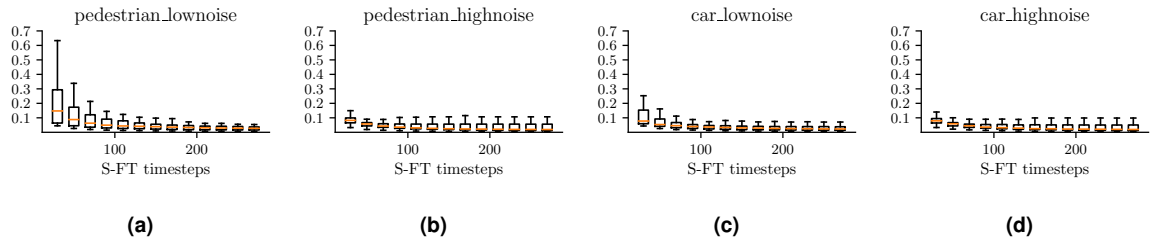


Figure 6.2: Boxplots of the RMSE between the S-FT and the FFT for different SNN simulation times t_s for the scenarios simulated with the library from Infineon AG. From left to right, the plots depict the results of the *pedestrian_lownoise*, *pedestrian_highnoise car_lownoise*, and *car_highnoise* subsets. The distribution outliers are not represented in the plots.

output of the S-FT and FFT were mapped to the same scale by normalizing them between zero and one. The experiments pipeline solved a target detection task to assess whether the errors obtained in the S-FT propagate forward on a signal processing pipeline. Namely, the standard and spiking pipelines included object detection and clustering algorithms after the FT. In the case of the spiking pipeline, the output spike train of the S-FT was decoded to real numbers before feeding the data to the detection algorithm. Object detection was implemented using an OS-CFAR algorithm with a window size of 32 cells, 4 guard cells, a scaling constant $\alpha = 0.4$, and an ordered-statistics $k = 6$. The clustering of the detected peaks was performed using a DBSCAN algorithm tuned with $min\_pts = 2$ and $\epsilon = 3$. Chapter 2 includes a description of both algorithms. Figure 6.1 depicts the result of computing one chirp with the S-FT for 100 timesteps per SNN stage, and the cluster labels obtained after applying the OS-CFAR and DBSCAN to the resulting spectrum. The plot also includes the result of the FFT for comparison and the evolution of the membrane voltage values during the simulation. The main difference between the S-FT and the FFT is that the relative intensities of the low-power S-FT bins are higher than those of the equivalent FFT bins. This is due to the saturation of the membrane voltage of the S-FT neurons that represent those bins, which happens when the voltage reaches the threshold value u_{th} before the end of the silent stage. This increases the intensity of low-power bins w.r.t that of the saturated bins. This phenomenon can be observed in the voltages' plot in fig. 6.1, where one neuron saturates positively and a second neuron saturates negatively at timestep $t \approx 40$. This particularity of the S-FT can be avoided by increasing u_{th} to a higher value so the voltage does not saturate during the silent stage. Even though this effect increases the final error, it benefits the frequency spectrum, allowing distinguishing targets with lower intensities. Note that the spectrum plots show the absolute value of the FT bins, assuming the spike timestep equivalent to a null intensity $|F| = 0$ is at $t = t_T \cdot 0.75$.

Figure 6.2 shows the boxplots of the RMSE between the S-FT and the FFT for the four different simulated subsets, for $t_s \in [30, 300]$ simulation steps. The RMSE diminishes in all cases with a higher number of simulation steps, following an exponential decay pattern and stabilizing after 100 timesteps. The variation of the RMSE in terms of standard variation and number of outliers improves with the number of simulation steps as well. This improvement is steeper for the low-noise scenarios, pointing towards the difficulty of the S-FT of replicating bins without targets nor noise, i.e., with values close to zero. The differences between the car and pedestrian scenarios are small, reflecting that the intensity of the reflected target does not have an influence in the performance of the S-FT. This is specially the case of the *pedestrian_lownoise* subset, where the configurations with few simulation steps yield unstable results, with a high standard deviation and a considerable number of outliers with high RMSE.

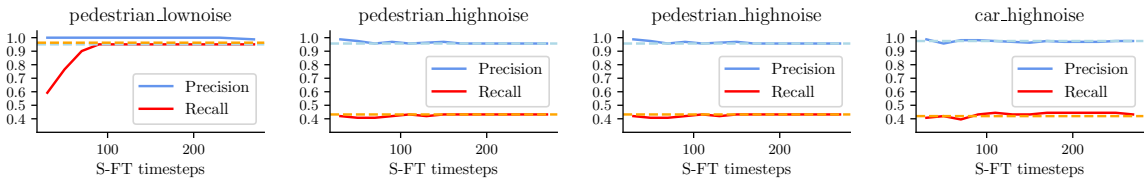


Figure 6.3: Performance of the S-FT and the FFT for different SNN simulation times t_s for the scenarios simulated with the library from Infineon AG. The measurements include the precision and the recall, in blue and red, respectively. The dotted lines indicate the performance of both metrics for the FFT. From left to right, the plots depict the results of the *pedestrian_lownoise*, *pedestrian_highnoise car_lownoise*, and *car_highnoise* subsets.

To evaluate if a labelled cluster corresponds to any of the targets in the scene, the post-processing of the pipeline assessed if the distance of any point p_i in the cluster to the target is smaller than a tolerance, $d < \text{tol}$, where $d = |p_i - p_{\text{target}}|$. All the experiments used $\text{tol} = 2$, i.e., clusters could not deviate more than one bin to be evaluated as correctly labelled. The analysis of the pipeline accuracy included the measurement of the precision and the recall of the labelled clusters, defined as

$$\text{precision} = \frac{TP}{TP + FP} \tag{6.9}$$

and

$$\text{recall} = \frac{TP}{TP + FN}\,, \tag{6.10}$$

where TP, FP, and FN are the true positives, false positives, and false negatives, respectively.

Figure 6.3 shows the accuracy results for the four different subsets, where the number of simulation steps varies in the same way as in the previous experiment. The solid lines in the plots show the performance of the S-FT and its variation for different values in the number of simulation steps. The dashed lines show the performance of the FFT used as a reference. As expected, the plots show a similar performance for the S-FT and the FFT. This similarity also holds for the different number of simulation steps, except for $t_s < 100$ for the *pedestrian_lownoise* subset. The precision of the S-FT tends to be better than that of the FFT, although both values converge for high values of t_s. This is probably because false-positive clusters typically occur when noise is labelled as a target, and noise is generally too low to be labelled when an FT with low quantization resolution is used, which is the case of the S-FT. When comparing the results in fig. 6.2 and fig. 6.3, we can see that even when the S-FT has higher errors with respect to the FFT, the performance of the object detection pipelines is analogous. As the error introduced by the S-FT is due to quantization shifts, the relative intensity of the peaks w.r.t their neighbours generally stays the same.

Validation on real data

The remainder of this section validates the S-FT by applying it to radar data obtained from real-world scenarios and comparing its performance with the output of an FFT accelerator. Contrary to the previous experiment, the data from this experiment does not contain labels with the location of the targets in the scene, so the evaluation was limited to the error between the spiking and non-spiking FT. This experiment aimed to validate the performance on a few scenarios posing different challenges for the FT processing, as collecting a large dataset on real scenarios was out of the scope of this experiment.

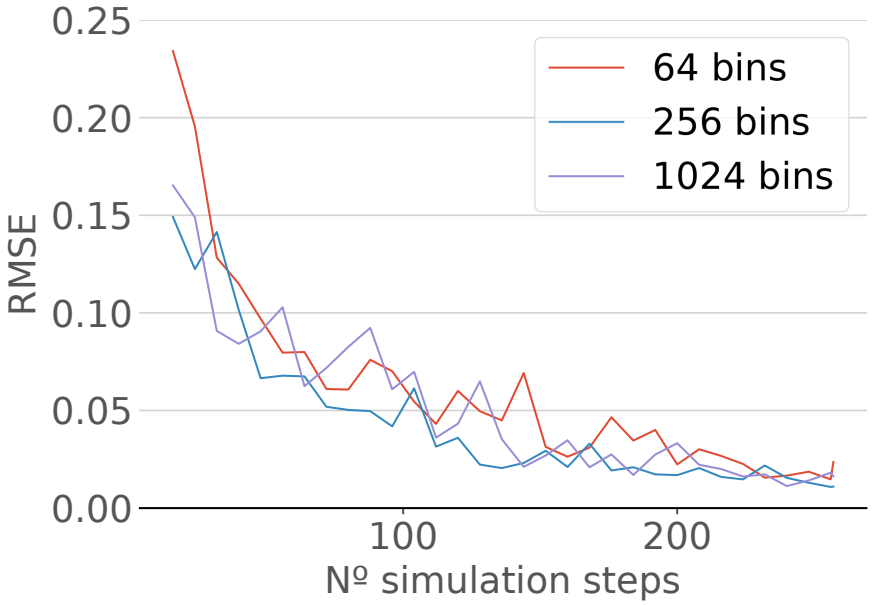


Figure 6.4: Average error of the S-FT for the five scenarios considered, computed using (6.7). The plots show the error as a function of the number of simulation steps and for three different FFT bin size configurations, $N \in [64, 256, 1024]$. ©2022 IEEE

The validation experiment of the S-FT took place on five different street scenarios:

1. One target close to the sensor, and a second target far from it.
2. A weak target in front of the sensor.
3. Two targets close to each other.
4. Multiple targets in front of the sensor.
5. A target moving in front of the sensor.

The first four scenarios were one-dimensional, i.e., they collected data in the range dimension. On the other hand, the last scenario was two-dimensional, i.e., the output of its frequency spectrum was a range-Doppler map. The experiment assessed the output of the S-FT implemented on the Loihi 1 board using the RMSE as metric (6.7). The measurements used the output obtained from a software-based FFT as a reference, i.e., from the implementation in the NumPy Python library. The S-FT ran on a physical Loihi boardand processed in a batch fashion, i.e., before the execution of the algorithm, a batch of data was converted into spike times, and it was processed by the S-FT in Loihi afterwards. Figure 6.4 shows the change in the performance of the algorithm when varying two different parameters: The simulation length t_s and the FT bin size N.

The results in fig. 6.4 show that the error of the S-FT decreases exponentially with the simulation length regardless of the number of bins in the generated frequency spectrum. This is the expected behaviour, aligning with the results in fig. 6.2. The simulation time t_s of the silent stage directly impacts the quantization of the frequency intensities, whereas the reference FFT always uses the same resolution of 64 bits. The invariance of the performance of the S-FT w.r.t the bin size is also expected, as a smaller N leads to a loss in the bin resolution of both the S-FT and FFT outputs, but not in the intensity resolution, i.e., the frequency

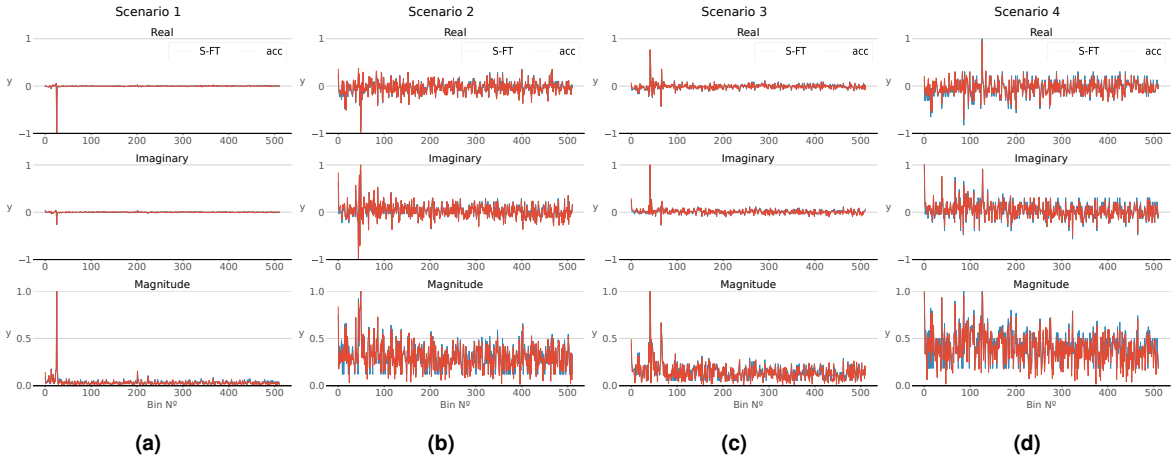


Figure 6.5: Output of an FFT accelerator (in blue) and the S-FT (in red) for the four static scenarios described in section 6.1.2. The plots include the real and imaginary components of the FT, as well as its absolute value. ©2022 IEEE

resolution is determined by (2.36), which only depends on N and thus is the same for the FFT and the S-FT.

The experiments also compared the error of the frequency spectrum generated by the S-FT to that of an FFT accelerator. The S-FT was simulated for 256 simulation steps, and the generated spectra contained 256 frequency bins each. Table 6.2 shows the average error for the different scenarios. The table shows the performance for a *discrete Fourier transform* (DFT) and an FFT spiking architectures, and the output of the FFT accelerator. Figure 6.5 and fig. 6.6 depict the output of the S-FT and the FFT accelerator for the aforementioned four static scenarios and for the 2D dynamic scenario, respectively. We observe that the S-FT provides a similar output pattern as the accelerator and that the S-FT plots do not show artifacts. When inspecting the quantitative results in table 6.2, we see larger errors for the S-FT, most likely due to quantization errors because of the smaller intensity resolution for the S-FT. Otherwise, we would expect anomalies in fig. 6.5. We also observe that the errors of the S-FFT architecture are similar to those of the S-DFT. This result suggests that the error generated in one layer does not accumulate and that a multi-layer structure does not lead to higher errors. The result of the 2D scene depicted in fig. 6.6 is analogous to the results for the static scenes and does not pose any particular challenges for the S-FT. In this case, the S-FT is applied to two consecutive layers to compute the range and Doppler dimensions.

Performance in neuromorphic hardware

To test the feasibility of accelerating the S-FT in neuromorphic hardware, the SNN was implemented on two different families of neuromorphic chips: Loihi and SpiNNaker. Both are digital chips, so the results should be analogous to simulations of the S-FT on a PC. Due to the limitations in the number of cores in the SpiNNaker 2 prototype board, these experiments

Table 6.2: RMSE of the *discrete Fourier transform* (DFT) and FFT architectures of the S-FT network, as well as an FFT accelerator used for comparison, on the four static scenarios introduced in section 6.1.2.

Architecture	S1	S2	S3	S4
S-DFT	0.004	0.041	0.009	0.030
S-FFT	0.006	0.026	0.007	0.028
Accelerator	0.0005	0.0005	0.0005	0.0005

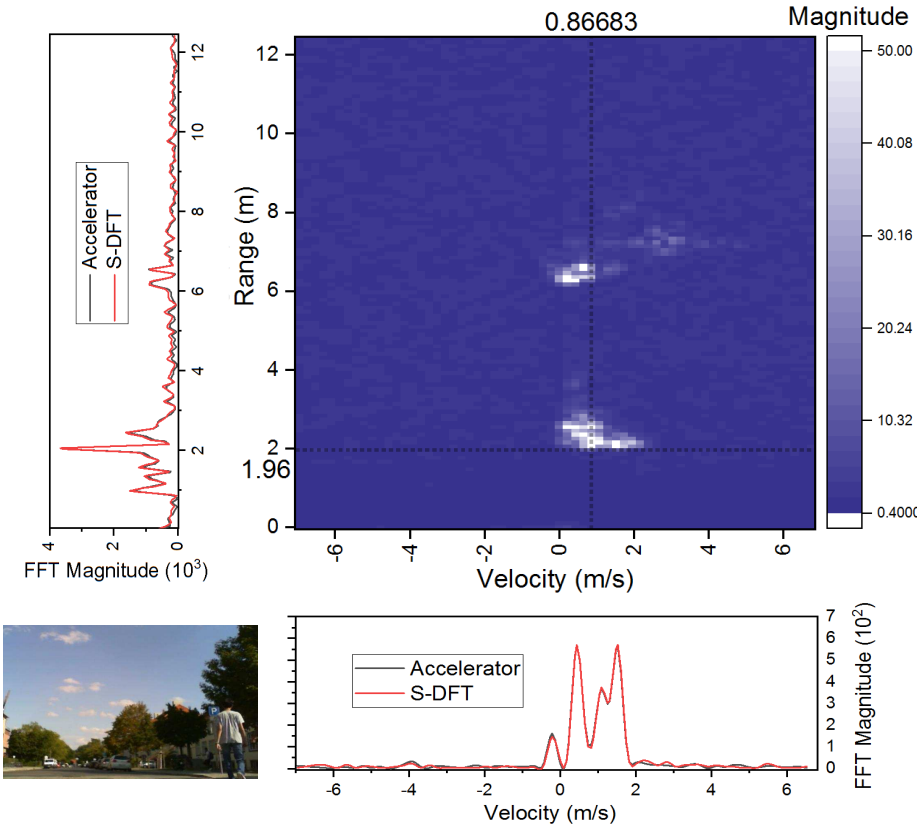


Figure 6.6: Output of an FFT accelerator (in blue) and the S-FT (in red) for the last scenario described in section 6.1.2. The plot on the left shows the result for all range bins for a specific velocity bin, and the plot at the bottom shows the result for all velocity bins for a specific range bin. ©2022 IEEE

ran the S-FT for a size of 256 input samples, as opposed to the 512 input size used in previous simulations in Loihi and PC. Figure 6.7 shows the error distribution when running the S-FT in Loihi 2 and SpiNNaker 2 chips, together with the error of the corresponding simulation in NumPy.

The results show that the three simulations hold a low error with similar deviations. This is especially the case for the SpiNNaker 2 experiment, as it yields a result identical to the simulation in NumPy, hinting the SpiNNaker 2 board can act as a signal processor with predictable behaviour. This predictability enables the simulation of SNNs in a traditional processor prior to deploying the neuromorphic chip for ease of development and debugging, speeding up the implementation time. The result of the Loihi 2 experiment shows a drift in the error, indicating that it may introduce some artifacts during the simulation.

Besides the accuracy of the S-FT for performing the frequency spectrum analysis, the knowledge of the resources it will consume is paramount. Table 6.3 summarizes the energy and time performance of the S-FT for an implementation in the Loihi 1 chip. The values in the table are an estimation based on the equations provided in [Dav+18]. Namely, the energy consumed E_{loihi} is calculated as

$$E_{\text{loihi}} = n_{\text{spikes}} \cdot 23.6\text{pJ} + n_{\text{steps}} \cdot n_{\text{neurons}} \cdot 52\text{pJ}\,, \tag{6.11}$$

where n_{spikes}, n_{steps}, and n_{neurons} are the number of spikes, simulation steps and instantiated neurons for the SNN simulation. The execution time T_{exec} is calculated as

$$T_{\text{exec}} = n_{\text{spikes}} \cdot 3.5\text{ns} + n_{\text{steps}} \cdot n_{\text{neurons}} \cdot 8.4\text{ns}\,. \tag{6.12}$$

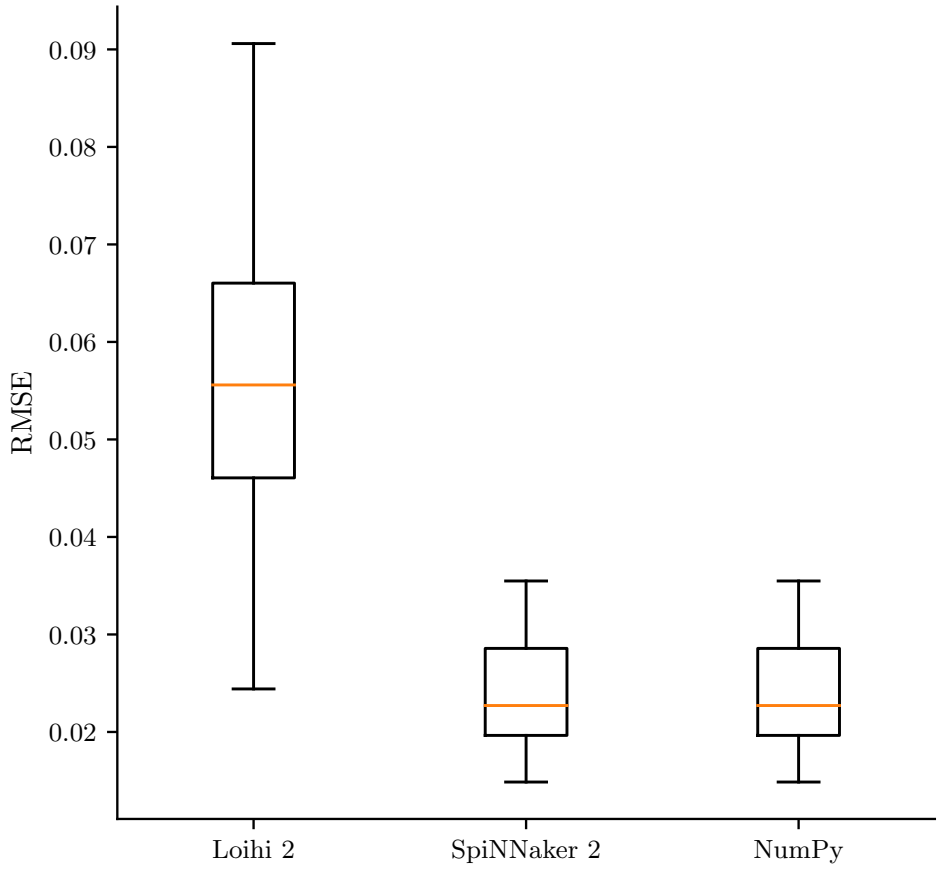


Figure 6.7: Distribution of the S-FT error when simulating it in the Loihi 2 board, the SpiNNaker board, and with NumPy library, for 30 scenarios simulated with the library from Infineon AG. Each S-FT was run for $t_s = 100$ timesteps, $N = 256$ samples, and the results ignored the first 12 bins of the spectrum. The distribution outliers are not represented in the plots.

The value of n_{spikes} in (6.11) and (6.12) refer to the total number of spike operations during the simulation, where a spike operation is the spike event between a source neuron and a destination neuron. This means that for each spike a neuron i emits, the number of spike operations equals the number of destination neurons connected to it,

$$n_{\text{spikes}_i} = \sum_{t=0}^{n_{\text{steps}}} s_i(t) n_{\text{out}}\,, \tag{6.13}$$

where n_{out} is the number of post-synaptic neurons connected to neuron i. T_{exec} defines the total time for processing a chirp and obtaining the spikes that define its frequency spectrum. Another important temporal metric is the chirp rate τ_{chirp}, which is the rate at which new chirps can be fed to the network. For multi-layer SNNs, these two values differ, as a new chirp can be sent to the SNN once the first layer has processed the previous one and before the final result has been obtained. This value depends on the latency T_{exec} and the total number of layers.

The current versions of Lava Library and Loihi 2 software stack include tools for measuring the energy consumed by an SNN simulation. However, this only works for native neuron models and specific simulations, so an accurate measurement of these parameters is not yet possible. Future versions of the software will likely enable this feature. Nevertheless, the values for Loihi 2 should exceed those of its first version, with improvements of at least 10× for the execution times[1]. The situation is similar for the SpiNNaker family: Researchers have already provided estimates of the simulation times for the SpiNNaker 1 board [Rho+18], and current research aims to provide measurement for the energy and time consumed in the SpiNNaker 2 board.

Table 6.3: Estimated performance indicators for the S-FT when processing a chirp with 1024 samples and 75 steps per simulation stage on a Loihi 1 chip. From top to bottom, the parameters refer to the number of neurons in the S-FT, number of spike events, energy consumed, execution time for one chirp, chirp input rate, and drained power.

Parameter	S-DFT	S-FFT
n_{neurons}	2048	10240
n_{spikes} $(\times 10^3)$	2100	84
$E(\mu J)$	65.5	49.9
$T_{\text{exec}}(\mu s)$	77.6	315
$\tau_{\text{chirp}}(\mu s)$	77.6	105
P (mW)	844	158

6.2 Analogue to spike encoder

This section covers the implementation and the experimental results of the ASE described in chapter 5. The implementation includes the electric circuit setup and the programming of the microcontroller that collected the spikes generated by the ASE. The first set of experiments assessed the encoding performance of the ASE for a given range of input values. These experiments analysed the impact of the different parameters on the performance of the ASE. Further experiments evaluated the ASE in a signal processing pipeline where the output of the ASE was used by a digital neuromorphic chip that computes the S-FT. In the latter case, the accuracy of the pipeline not only depends on the encoding performance but also on the compatibility of the encoded spikes with the following SNNs. Thus, the experiment compared the S-FT results with those of a standard digital pipeline that computes the FFT of an equivalent time series.

6.2.1 ASE implementation

After designing the encoder and establishing the behaviour of its input and output signals, a physical prototype of the ASE was constructed and tested with actual electric signals. The prototype was controlled by a microprocessor, which emulated the role of a digital neuromorphic chip that reads and processes the ASE spikes. Therefore, the programming of the microprocessor was also part of the implementation of the ASE prototype.

[1] https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf

Circuit construction

The construction of the ASE took place on a prototype board. Its main purpose was to validate the functionality of the ASE and to provide an estimation of the error produced by the ASE encoding when dealing with real signals. The chosen electric components had through-hole pins, and the auxiliary dsPIC microcontroller was mounted on the same board. As the consumption on a breadboard circuit is not comparable to that of a highly optimized VLSI circuit, the drained energy was not measured. Figure 6.8 shows a picture of the mounted prototype. Decoupling capacitors protected the different ASE components, and a potentiometer allowed the threshold voltage tuning. The communication between the microcontroller and the ASE consisted of two separate wires, carrying the spike events and the reset signal, respectively. The microcontroller communicated the collected spike times via UART protocol.

The supply voltage of the circuit was 5 V, and the input signals were on the range $[1, 5]$ V. A constant voltage power supply provided flat input voltages for testing the encoding performance of the ASE. Experiments assessed the performance of the ASE together with the S-FT by connecting the circuit to a function generator that provided sinusoidal electric signals. Appendix C contains a summary with the specific components used for constructing the ASE, and fig. C.1 shows a picture of the lab setup containing the ASE, the function generator, and an oscilloscope for validating the input signals.

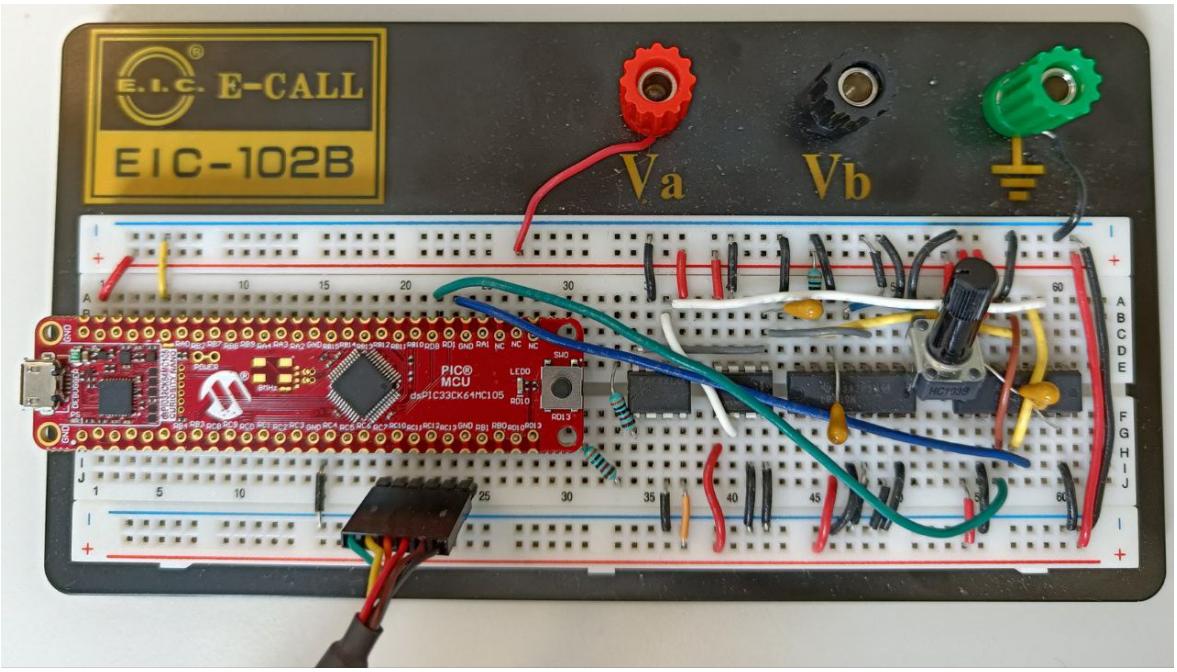


Figure 6.8: Prototype board for testing the ASE. The red board on the left contains the dsPIC33 microcontroller that collected the spikes and sent them via UART to an external PC. The microcontroller collected spike events and sent a reset signal to the ASE (green and blue wires, respectively). The dsPIC and the ASE had a common supply of 5 V connected to the red and green banana sockets. The circuit processed analog signals connected to the resistor pin on the left. The potentiometer provided the threshold voltage u_{th} of the ASE.

dsPIC microcontroller for data acquisition

The implementation of the ASE included not only constructing the circuit but also the programming of a microcontroller for interfacing the circuit. The function of the microcontroller can be split into the following tasks:

- Provide a clock signal CLK_S with period T_S that serves as a reference for data sampling.
- Generate an internal clock signal CLK_N that defines the time step resolution T_N of the spike times.
- Assess the presence of a spike event in the ASE on every time step and store the spike time t_s.
- Reset the spike event signal SPK after registering it.
- Reset the refractory period signal S_{refr} at the end of each sample period, T_S, by setting an output signal RST to 1.
- Send the recorded spike times after collecting N samples.

Figure 6.9 contains two flow charts that describe the routine programmed on the dsPIC33 microcontroller. The main routine waits for an external new sample request, enabling the timer T1 afterwards with a period equivalent to the spike sampling time T_N and locking itself until the flag DONE is set. The T1 interrupt routine repeats the sampling N times, where N is the number of samples in a chirp. Thus, T1 runs N cycles and collects one spike time per cycle, storing them on the *spikes* array. Each cycle runs for a number of steps equal to the STEPS constant and records the step number at which the SPK flag is triggered. After receiving the spike, the flag RST, which is connected to the output pin that resets the ASE circuit, is set. The value in STEPS is fixed according to the configuration of the microcontroller and the sampling time T_N. At the end of each sample, the flag RST is deactivated. After collecting the N spike times, the timer is deactivated and the DONE flag is set, which commands the main routine to start the data sending via UART for post-processing. This is done by iterating N times over the *spikes* array and sending one spike time per iteration.

6.2.2 Experiment results

The experiments on the ASE prototype focused on validating the functionality of the circuit. The experiments analysis based the performance of the ASE on two metrics. The first metric is the error between the target output $y(x)$ and the estimated output $\hat{y}(x)$ for a given input signal x,

$$\varepsilon = \int_{x_{\min}}^{x_{\max}} |y(x) - \hat{y}(x)| dx \,, \tag{6.14}$$

where $x_{\min}$ and $x_{\max}$ are the limits of x. In the simplest case, $\hat{y}$ is the result of decoding the spikes of the ASE for reconstructing the original input signal, i.e., $y(x) = x$. In more complex cases $\hat{y}$ is the result of processing the spikes of the ASE with a given function F, i.e., $y = F(x)$. The second metric is the time ratio μ_T between the time range t_{spk} that can be used by the ASE for generating a spike, and the fastest reaction time t_{wait} of the ASE, i.e., the time it needs for encoding the highest possible input.

$$\mu_T = \frac{t_{\max} - t_{\min}}{t_{\min}} = \frac{t_{\text{spk}}}{t_{\text{wait}}} \,, \tag{6.15}$$

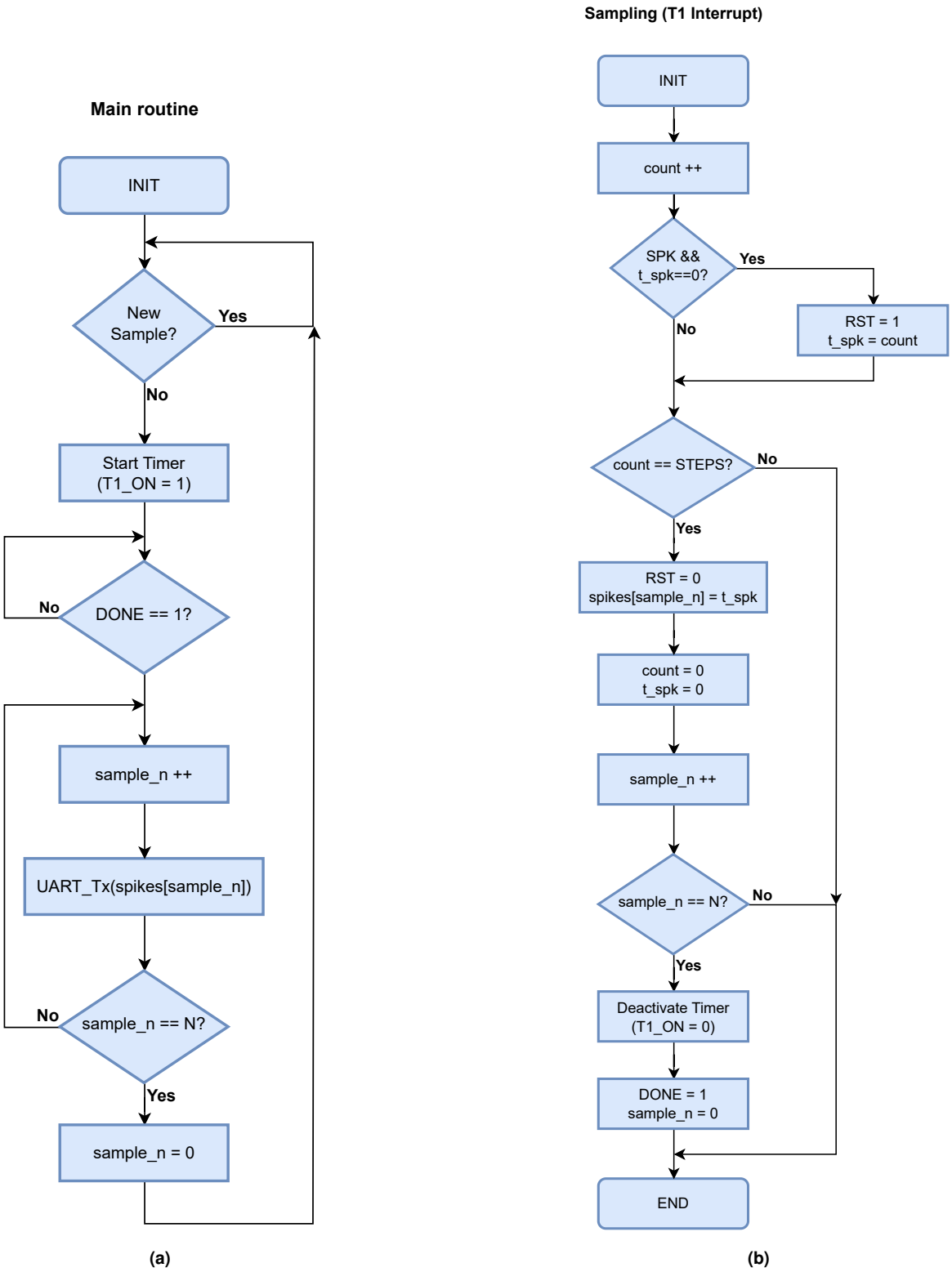


Figure 6.9: Flow charts of the dsPIC microcontroller routine for collecting spikes from the ASE and sending them via UART. Rounded rectangle blocks represent the start and end of the routines, rectangles with sharp edges represent instruction blocks, and diamonds represent conditional evaluations. Symbols in capital letters represent global boolean flags (DONE, T1_ON), constants (STEPS, N) and I/O signals (SPK, RST). Symbols on small letters represent local variables. (a) depicts the chart of the main routine, that checks for *new sample* requests and starts the sampling timer T1. After collecting N samples, the routine sends them via UART. (b) depicts the chart of the sampling routine, that is run when the T1 interrupt is triggered, i.e., after a time equivalent to the spike sampling period T_N. Every cycle increases the step count, checks if there is a spike event SPK present, and sends the reset signal RST to the ASE. The microcontroller repeats this process until N spikes are sampled.

where t_{min} and t_{max} are the limits of the ASE spike times. The existence of t_{wait} is due to the physical constraints of the ASE, as the capacitor needs to charge before generating a spike. As the optimization of the ASE parameters aims to maximize μ_T and minimize ε, we rephrase the generic loss function in (5.15) as

$$\mathcal{L} = \alpha\varepsilon - \mu_T\,, \tag{6.16}$$

where α is the weight of the error w.r.t the time ratio. Therefore, the main criteria for tuning the ASE is to minimize the loss $\mathcal{L}$.

The testing of the ASE was split into three use cases. The first use case assumed that the processor using the generated spikes is an *ideal decoder*, i.e., it extracts the information from the spike times using the inverse of the ASE's encoding function. The second use case assumed that the processor decodes the spike times using a *linear decoder*. The relevance of this scenario arises from the fact that the S-FT network employs integrate-and-fire neurons without a leak, so the combination of the ASE and the S-FT needs to minimise the errors in the encoding functions mismatch. Finally, the third use case implements the *Fourier transform* on the obtained spikes using the S-FT. By implementing the second and third cases separately, we can clearly understand the error introduced using different encoding functions and whether that error translates into an error in the resulting frequency spectrum.

Scenario 1: Ideal decoder

The first batch of experiments used constant voltage signals as input to the ASE and collected the resulting spike times of the ASE. The experiment assessed the encoding error by using an ideal decoder for reconstructing the original signal. Thus, for a constant input voltage u_{in}, the ASE provided spike times t_s according to the encoding function f, and an ideal decoder provided estimates of the input signal $\hat{u}_{\text{in}}$ by applying the inverse of the encoding function f^{-1},

$$u_{\text{in}} \xrightarrow{f} t_s \xrightarrow{f^{-1}} \hat{u}_{\text{in}}\,. \tag{6.17}$$

As the choice of τ_m does not have an impact in $\mathcal{L}$, the constraints of the physical system determined its value, e.g., working on a time range compatible with the microcontroller's maximum sampling rate and using choosing sensible electronic components for a breadboard environment. Thus, the ASE's time constant had a fixed value of $\tau_m = 3$ ms for all the experiments. For assessing the error of the ASE, we can adapt (6.14) to a discrete scenario, so we define the total encoding-decoding error as the difference between the input signal u_{in} and the reconstructed signal $\hat{u}_{\text{in}}$ for the whole input range $[u_{\text{min}}, u_{\text{max}}]$,

$$\varepsilon_u = \sum_{m}^{M} |u_{\text{in, m}} - \hat{u}_{\text{in, m}}|\,, \tag{6.18}$$

where $u_{\text{in, m}}$ is the mth sample of u_{in}, which is sampled M times at regular intervals between u_{min} and u_{max}. Let us also define the error in the spike times in a similar way,

$$\varepsilon_{t_s} = \sum_{m}^{M} |t_s - \hat{t}_s|\,, \tag{6.19}$$

where $\hat{t}_s$ is the recorded spike time from the ASE, and t_s is the ideal spike time that would take place if the ASE had an infinite output resolution. i.e., if the sampling period T_N was infinitely small, $T_N \approx 0$.

Part of the error is due to the accumulation of random noise sources that are hard to predict and quantify, e.g., environment interferences, imperfections on the different components, or coupling behaviours throughout the circuit. The most notable effect of this noise

are fluctuations in the membrane voltage, $u(t)+\psi_u$, that leads to an error on the spike time ψ_t. We can formally define this error as

$$t_s + \psi_t = -\tau \ln\left(1 - \frac{u_{\text{th}} - \psi_u}{u_{\text{in}}}\right). \tag{6.20}$$

Another portion of the error is due to the quantization error when converting analogue voltages into discrete spike times. This error is systematic, as it only depends on the parameters of the ASE. Assuming an ideal behaviour of the circuit, the quantization error δt_q when converting a voltage u_{in} into a spike time can be modelled as

$$\delta t_q = \frac{T_N}{2}, \tag{6.21}$$

where T_N is the neural sampling period for the spike times. (6.21) leads to an error δu_q in the decoded signal. The hyperbolic nature of the encoding function makes the impact of the quantization error larger for smaller values of u_{in}, due to the larger ratio between u_{in} for two consecutive spike time bins. The combination of (6.20) and (6.21) yield

$$\hat{t}_s = t_s + \delta t_q + \psi_t, \tag{6.22}$$

that leads to the decoded signal

$$\hat{u}_{\text{in}} = \frac{u_{\text{th}}}{1 - (1 - \frac{u_{\text{th}} \pm N_u}{u_{\text{in}}}) e^{-\delta t_q/\tau}}. \tag{6.23}$$

Figure 6.10 shows the decoding error when feeding to the ASE constant voltage signals in the range $[1,5]$ V for different threshold voltages u_{th}. The plots depict the errors for the estimated signal $\hat{u}_{\text{in}}$ and for the resulting spike times $\hat{t}_s$. The spike time errors disregard the quantization error, as the ideal spike time t_s already takes into account the granularity of the spike times. Thus, the bottom plots offer a picture of the random noise ψ_u for the different values of the input range. The noise is larger for smaller values of u_{in} This is due to the hyperbolic nature of the ASE charging, because when the membrane voltage of the ASE reaches u_{th} the charging slope decreases, and hence small oscillations in $u(t)$ lead to larger differences in the spike time. The plots of the total decoding error ε_u depict the impact of the random noise ψ_t together with the quantization error δt_q. Moreover, the light blue plots show the theoretical average quantization error for the different values of u_{in}. The effect of δt_q becomes larger for big values of u_{in}, and an initial conclusion when comparing the top and bottom plots is that the quantization error has a larger impact on the total decoding error. Table 6.4 shows the values of the different metrics for evaluating the decoding.

Figure 6.12a shows the result of decoding the spike times of the ASE with an ideal decoder for four different values of u_{th}. The results are in line with the error shown in fig. 6.10, as they are larger for big values of u_{in}. In any case, the error stays small relative to the input voltage. On the other hand, the time ratio μ_T increases for larger values of u_{th}. Therefore, the optimal solution for this scenario would be to choose the maximum possible value for the threshold voltage, $u_{\text{th}} = 0.9$ V.

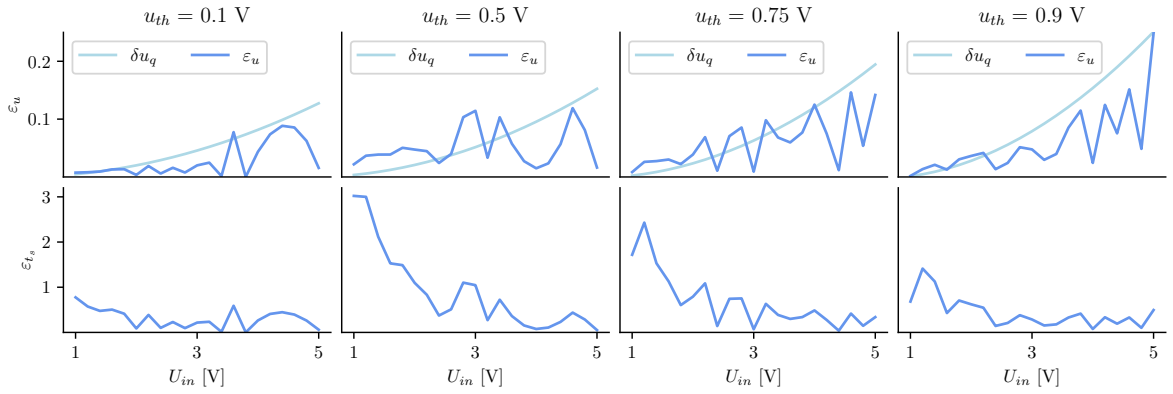


Figure 6.10: Decoding error of the ASE for input signals in the range $[1,5]$ V. The top plots show the total decoding error ε_u together with the theoretical quantization error δu_q (see (6.21)), in dark blue and light blue, respectively. The bottom plots show the error in the spike times. From left to right, the plots show the results of the experiment for u_{th} values of $0.1, 0.5, 0.75$, and 0.9 V, respectively [LRK23].

Table 6.4: Quantitative results when decoding ASE results with ideal decoder, using $\alpha = 100$

	$u_{\text{th}} = 0.1$ V	$u_{\text{th}} = 0.5$ V	$u_{\text{th}} = 0.75$ V	$u_{\text{th}} = 0.9$ V
ε_u	0.03	0.05	0.06	0.06
μ_T	4.15	5.4	7.6	9.9
$\mathcal{L}$	-1.15	-0.4	-1.6	-3.3

Scenario 2: Linear decoder

A second batch of experiments decoded the same spike times from the previous experiment with a linear decoding function g^{-1} with tunable parameters $t_{\text{lin, min}}$ and $t_{\text{lin, max}}$,

$$u_{\text{in}} \xrightarrow{f} t_s \xrightarrow{g^{-1}} \hat{u}_{\text{in}}\,. \tag{6.24}$$

Contrary to the experiments in the first scenario, this scenario does not assume an ideal decoding, so the decoding function is an approximation of the encoding. To obtain a good mapping, a differential evolution optimizer[2] performed the tuning of the linear decoding function by setting the time limits of the linear function g. They were tuned using the parameters k_1 and k_2,

$$t_{\text{lin, min}} = t_{\text{min}}(1 + k_1)\,, \tag{6.25}$$

and

$$t_{\text{lin, max}} = t_{\text{max}}(1 + k_2)\,. \tag{6.26}$$

As the spike times are positive real values, k_1 and k_2 are limited to the range $[-1, \infty]$. Based on this limitation, the boundaries during the optimization process were set to $k_1, k_2 \in [-1, 2]$. To penalize large errors, the fitting process used the RMSE metric (6.7) for measuring the error, $\varepsilon = E$, where the reference value is the original signal u_{in} and the estimate is the linear decoding $g^{-1}(f(u_{\text{in}}))$.

As the goal of the experiments was to fit a linear function to the ASE model, the experiments used fixed values for the ASE parameters (u_{th} and τ_m). Therefore, the time ratio μ_T was constant for each experiment and the loss function (6.16) became

$$\mathcal{L}' = E\,. \tag{6.27}$$

[2] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html

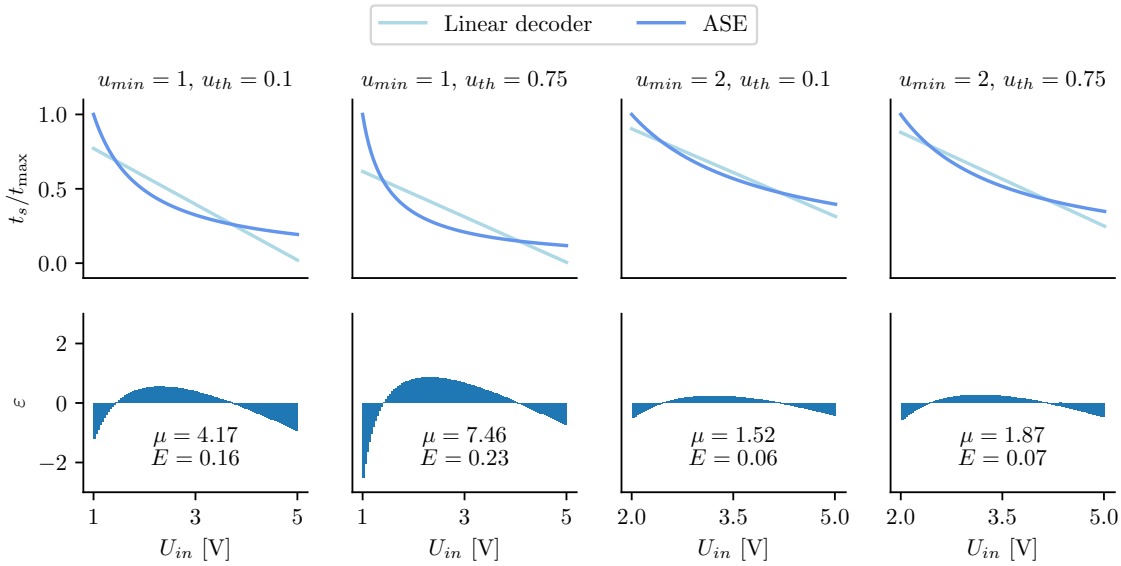


Figure 6.11: On top, the comparison between the output of the ASE and the fitted linear decoder $g^{-1}(t_s)$ for $u_{\text{th}} \in [0.1, 0.75]$ V and $u_{\text{min}} \in [1, 2]$ V. At the bottom, the error of the decoded voltages, $\hat{u}_{\text{in}} - u_{\text{in}}$ for each encoded voltage. The optimizer tuned the parameters assuming the ideal behaviour of the ASE, i.e., $\psi_u = 0$ [LRK23].

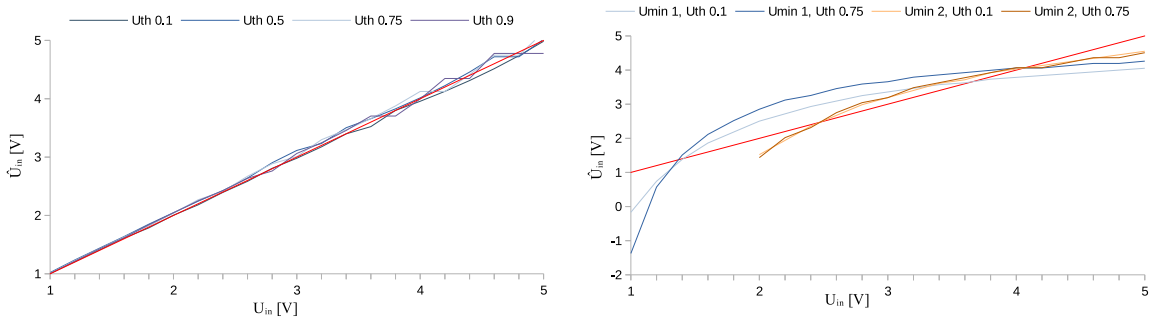
The Taylor expansion introduced in chapter 5 showed that the linear error of the encoding function is proportional to the ratio $u_{\text{th}}/u_{\text{min}}$. Therefore, the experiments also evaluated the impact of u_{min}. Namely, they included four different parameter setups of the ASE that combined two values of u_{th} with two values of u_{min}. Figure 6.11 shows on top the encoding and decoding curves of these four parameter setups in dark blue and light blue, respectively. The bottom plots show the distribution of the decoding error ε over the input voltage range for each setup. Moreover, fig. 6.12b shows the decoding of the ASE spike times when using an optimized linear decoder for the aforementioned combinations of u_{th} and u_{min}.

The results show that the choice of u_{min} has a big impact both in the error and in the time ratio. High values of u_{min} lead to lower errors as well as lower time ratios, so the chosen setup needs to be a compromise between both indicators. In other words, the choice of the ASE parameters is application-specific, as it depends on the admissible error in the decoding and the latency requirements.

The plots also show that the error distribution is not random. In fact, the decoding error tends to be larger near the range limits and in the middle of the encoding range. This characteristic could be exploited, and the optimizer could use criteria based on data heuristics to obtain better results, i.e., instead of using the absolute error in (6.27), the optimizer could use an error vector weighted with the density of information throughout the input range spectrum and transform (6.18) into

$$\varepsilon_h = \int_{x_{\text{min}}}^{x_{\text{max}}} w_h(x)|y(x) - \hat{y}(x)|dx\,, \tag{6.28}$$

where $w_h(x)$ is a heuristics coefficient that depends on the data distribution.



(a) (b)

Figure 6.12: Result of decoding the spike times provided by the ASE when (a) using an ideal decoder, i.e., $\hat{u_{\text{in}}} = f^{-1}(t_s)$, and (b) a linear decoder fitted with a differential evolution optimizer. The red line represents the target result and serves as a reference. The first experiment includes four results for $u_{\text{th}} \in [0.1, 0.5, 0.75, 0.9]$ V, respectively; and the second experiment includes results for the four combinations of $u_{\text{th}} \in [0.1, 0.75]$ V and $u_{\text{min}} \in [1, 2]$ V [LRK23].

Scenario 3: Fourier transform of ASE output

The third scenario sent the spikes from the ASE to the digital neuromorphic chip SpiNNaker 2 for computing the S-FT algorithm introduced in section 4.2. The experiments employed a function generator for producing a sinusoidal wave. Hence, the input to the ASE was a time-varying voltage signal $u_{\text{in}}(t)$ modeled according to

$$u_{\text{in}}(t) = A \sin 2\pi \nu t + B\,, \tag{6.29}$$

where A, ν, and B are the amplitude, frequency, and offset of the sinusoidal wave, respectively. Following the same notation as in the previous two experiments, in this case scenario, the ASE maps the input voltage u_{in} to spike times t_s, and after that a spiking algorithm A_s transforms the spike times into the output variable $\hat{y}$,

$$u_{\text{in}} \xrightarrow{f} t_s \mapsto A_s(t_s) \xrightarrow{g^{-1}} \hat{y}\,, \tag{6.30}$$

where A_s assumes that the input spike times are encoded with the function g. For assessing the performance of the whole pipeline in (6.30), the error function compares the estimated output $\hat{y}$ with the ideal output y obtained when using the non-spiking algorithm A on the analog input, $y = A(u_{\text{in}})$. For the specific case of the Fourier transform, the algorithm takes the amplitude of a signal over time and provides the frequency spectrum of the signal during a time window of width T_M. Its discrete form takes as input a time series of the signal formed by M samples and generates a frequency spectrum with M bins, where $M/2$ bins belong to the negative spectrum, and $M/2$ bins belong to the positive spectrum. For the given specifications, the input signal needs to be discretized with a sampling period T_S

$$T_S = \frac{T_M}{M}\,. \tag{6.31}$$

Section 2.3 provides a more detailed description of the algorithm. The S-FT implementation assumes that the amplitude of the input signal is represented by spike times. For obtaining a spike train resolution of N time steps per spike, the neural sampling period T_N needs to be

$$T_N = \frac{T_S}{N}\,. \tag{6.32}$$

The value of the neural sampling period T_N is a parameter of the digital chip that collects the spikes of the ASE, which in the case of these experiments was the dsPIC microcontroller. Hence, the limit value of this parameter is determined by the specifications of the digital chip.

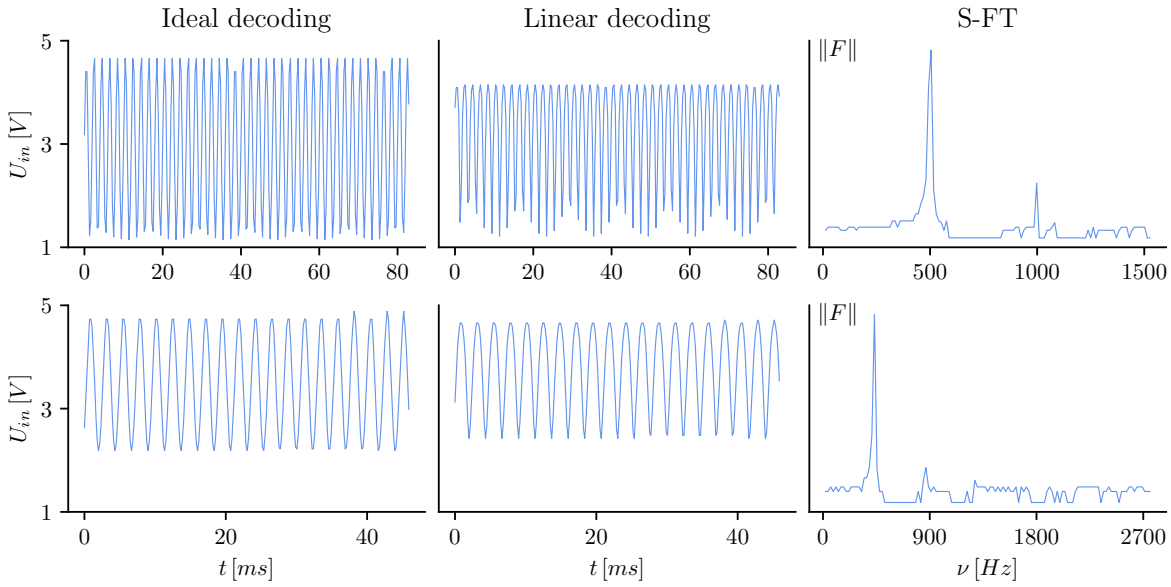


Figure 6.13: Result of applying the S-FT to the output of the ASE to sinusoidal waves with with frequency $\nu = 500$ Hz and amplitudes of 4 V (top) and 3 V (bottom). The left column shows the result of decoding the obtained spikes with an ideal decoder f^{-1}. The middle column shows the result of decoding the spikes with a linear decoder g^{-1}. The S-FT was implemented on the SpiNNaker 2 board [LRK23].

The focus of the experiment was to minimize the error, so the maximization of the time ratio μ_T was a secondary target. Based on the results from the previous experiment, the threshold voltage was fixed to the value of $u_{\text{th}} = 0.1$ V, and the time constant stayed unaltered, $\tau_m = 3$ ms, so the results between experiments are comparable. To assess the impact of the ratio $u_{\text{min}}/u_{\text{th}}$, the experiments included a batch of signals with a *wide* voltage range, $u_{\text{in}} \in [1,5]$ V, and a *narrow* voltage range, $u_{\text{in}} \in [2,5]$ V. These ranges were achieved by setting the wave parameters to $A = 2, B = 3$ for the *wide* voltage range, and $A = 1.5, B = 3.5$ for the *narrow* voltage range. The experiments included runs for input frequencies of $\nu = \{25, 50, 75, 100, 250, 500, 750, 1000\}$ Hz. The dsPIC microcontroller sampled the ASE spikes with a resolution of 100 time steps per spike window. Equation (5.3) yields the maximum spike time t_{max} that the ASE can produce for $u_{\text{in}} = u_{\text{min}}$. Assuming $T_S \approx t_{\text{max}}$, the neural step resolution T_N of the dsPIC is set using (6.32), $T_N = t_{\text{max}}/100$. The sampling parameters for the *wide* input range were $t_{\text{max}} = 320\mu$s and $T_N = 3.2\mu$s. For the *narrow* input range, they were $t_{\text{max}} = 155\mu$s and $T_N = 1.55\mu$s.

Figure 6.13 shows the result of computing the S-FT on the spikes produced by the ASE for an input voltage frequency of 500 Hz. The top and bottom rows of the figure show the results for *wide* and *narrow* voltage ranges, respectively. The difference in the output range of the frequency spectrum is due to the different values for the sampling time T_S, which is larger for the *wide* input range. The left and middle plots show the result of obtaining $\hat{u}_{\text{in}}$ using an ideal decoder and a linear decoder. The reconstruction with the ideal decoder shows a sinusoidal wave with minor sampling artifacts. The signal reconstructed with the linear decoder shows more significant artifacts, especially for the *wide* input range. Moreover, the decoding introduces a compression of the signal for low values of u_{in} and an expansion for high values of u_{in}. This is due to the hyperbolic nature of the ASE, as the spike resolution increases with u_{in}. This phenomenon is reflected in the S-FT result as a second harmonic in the frequency spectrum, most notably when using a *wide* input range. Although undesired, this effect could be mitigated by attenuating the bins corresponding to the harmonics of the main signals, as the intensity of the artifact can be easily calculated analytically. Moreover, the

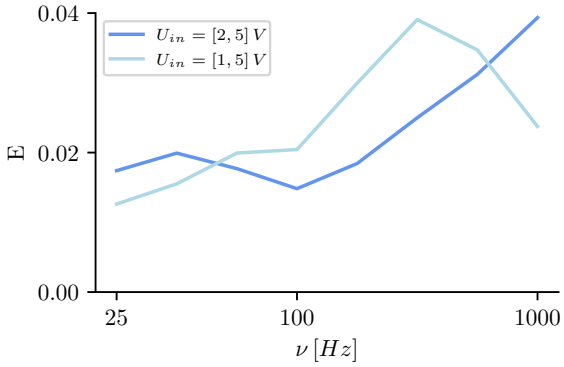


Figure 6.14: Error of the S-FT using as input the output spikes of the ASE. The experiment was performed on analogue signals with frequencies $\nu = \{25, 50, 75, 100, 250, 500, 750, 1000\}$, and the computed error metric was the root mean squared error (6.7). The plots show the measured error for $wide$ and $narrow$ input voltage ranges in light blue and dark blue, respectively [LRK23].

experiments used clean data obtained from a function generator. If the ASE model encodes sensor data, the harmonics lie below the sensor's noise level.

Figure 6.14 shows the total error of the S-FT for different input signal frequencies when using a *narrow* and *wide* input range. As expected, the error for the *wide* input range is larger for most cases, and the error seems to show exponential growth. This pattern reverses for the *wide* input range after $\nu = 500$ Hz. The decrease in the error after this point is due to the fact that the second harmonic in the frequency spectrum caused by the aforementioned hyperbolic distortion of the data lays outside of the frequency range of the S-FT for $\nu > 500$ Hz when using the setup for a *wide* input range. As the sampling frequency of the *narrow* input range is faster, the frequency limit for the S-FT is larger as well and the second harmonic is still present for the larger frequencies used in the experiment.

6.3 Spiking OS-CFAR

Section 2.5 introduced the concept of object detection in the context of radar signal processing, together with two popular variants of *constant false alarm rate* (CFAR) algorithms: the *cell-averaging CFAR* (CA-CFAR) and the OS-CFAR. This thesis has as a minor contribution a spiking version of the OS-CFAR. The S-OSCFAR algorithm is based on a winner-takes-all topology, where input neurons compete to determine the output spike train. The algorithm is described in appendix E. This section describes the implementation of the S-OSCFAR algorithm and the experiments that validated it, including the necessary preprocessing steps, the adjustment and encoding of input data into spikes, and the actual implementation in the neuromorphic chip SpiNNaker 2.

6.3.1 Implementation

The validation of the S-OSCFAR was performed through an implementation on a general-purpose processor, followed by further experiments that ported the algorithm to the SpiNNaker 2 board. From a computational perspective, the differences between both implementations are the maximum resolution that can be achieved for the synaptic weights and the maximum size of the input data.

The S-OSCFAR was implemented on the SpiNNaker 2 using a lightweight integrate-and-fire model, i.e., the implementation does not make use of delays, voltage decay over time, or short-term memory in the synapse, and the synaptic weights take two possible values: k for the cell-under-test and -1 for the neighbour cells. The weights had a 4-bit resolution for a 1D application with 10 neighbour cells. The implementation of the S-OSCFAR assumes the input to be an N-dimensional latency-encoded spike train, where the spike time of the ith input represents the intensity of the corresponding value in the original form. The encoding equation is a version of (4.3) with $t_{\min} = x_{\min} = 0$,

$$t_i = t_{\max} \frac{x_{\max} - x_i}{x_{\max}}, \tag{6.33}$$

where t_i and x_i takes values in the ranges $[0, t_{\max}]$ and $[0, x_{\max}]$, respectively. The membrane voltage $u(t)$ of the cell-under-test evolves according to integrate-and-fire dynamics,

$$u(t) = u(t-1) + \sum w_i \Theta(t_i). \tag{6.34}$$

It is important to note that the neuron described in (6.34) does not use an input current value and the decay constant is zero. This means that the influence of an input spike in the neuron is constant from the moment it occurs, hence only the specific order of the input spikes affects the output spike train.

6.3.2 Experiment results

Contrary to the ASE and S-FT, the S-OSCFAR employs data that is already preprocessed, so there is a larger availability of standard datasets for testing it. Namely, the validation experiments used data from the CARRADA dataset [Oua+21], which consists of $256x64$ range-Doppler maps from a 77 GHz FMCW radar with 4 GHz bandwidth, i.e., data already processed with an FT. The samples correspond to driving scenarios under different weather conditions, and they come together with data from LiDAR and camera sensors, and the scenes are partially labelled.

The experiments ran the S-OSCFAR for a total of 1000 randomly selected scenes and compared its results with those of a standard OS-CFAR. The accuracy of the algorithm was evaluated in terms of precision (6.9) and recall (6.10), taking as reference the output of the standard form of the algorithm. Moreover, the experiments compared the performance with the spiking CA-CFAR introduced in [Vog+22]. Both SNNs were simulated in software by mimicking the execution on a digital neuromorphic chip. A first batch of experiments tested the S-OSCFAR on input directly converted to spikes using (6.33). A second batch modified the input by applying the logarithmic conversion

$$x_{dB} = 20 \cdot \log_{10} x. \tag{6.35}$$

This change improves the resolution of the input for small intensities and helps the detection of low-amplitude targets. In this batch, the intensities of the neighbour cells were also modified by adding a small delay

$$t'_{x_N} = t_{x_N} + \Delta\tau, \tag{6.36}$$

where

$$\Delta\tau = \frac{\Delta t}{2}, \tag{6.37}$$

being Δt the time step resolution of the simulation, i.e., the spike time of the neighbour cells had a delay equivalent to half the resolution of the simulation time step. This modification

improves the mislabeling due to rounding errors when assigning the real spiking times t_i to discrete time steps.

Figure 6.15 compares the spiking CA-CFAR with the two versions of the S-OSCFAR for different numbers of simulation steps in terms of sensitivity and precision. Even though the basic version of the OS-CFAR shows perfect precision, this comes at the cost of lower sensitivity, i.e., all detected targets are correct, but not all targets are detected. The best results are obtained for the S-OSCFAR when converting the input to a logarithmic scale and adding a small delay to the neighbour cells. The figure shows that the performance for the latter approach is the same as the conventional OS-CFAR when using simulations with more than 100 steps. The spiking CA-CFAR reaches a performance on par with its counterpart too, but it requires a larger number of time steps to achieve such performance. Moreover, when the target is to minimize the employed resources, the S-OSCFAR outperforms the spiking CA-CFAR, as performance indicators for the former are over 90% for simulations with 10 time steps. Figure 6.16 shows the result of the spiking CA-CFAR and OS-CFAR for a specific frame from the CARRADA dataset.

The validation of the S-OSCFAR included the testing on the data produced in the KI-ASIC project, which consisted of real driving scenarios. As the algorithm does not require training, the validation was done on a small subset of the dataset. Figure 6.17 shows an example of

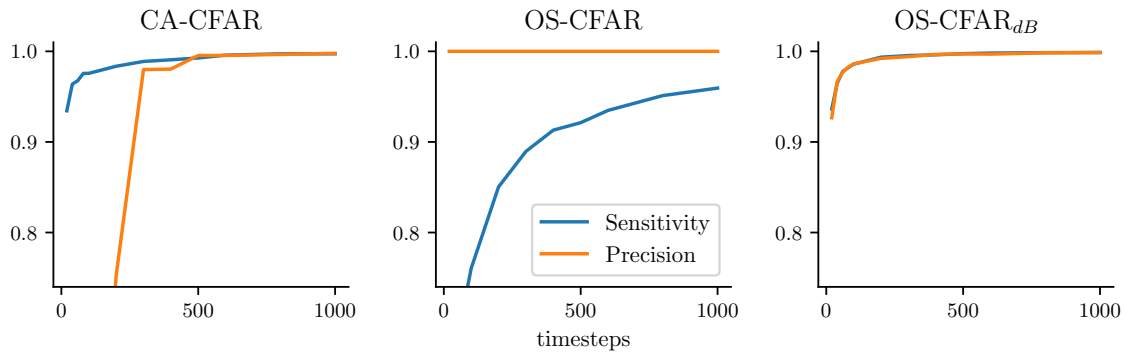


Figure 6.15: Performance of the spiking versions of the CA-CFAR (left) and OS-CFAR (middle and right), in terms of sensitivity and precision. The right plot corresponds to the evaluation of the S-OSCFAR when its input is previously converted to a logarithmic scale and the spikes to the neighbour cells arrive with a delay $\Delta\tau = \Delta t/2$.

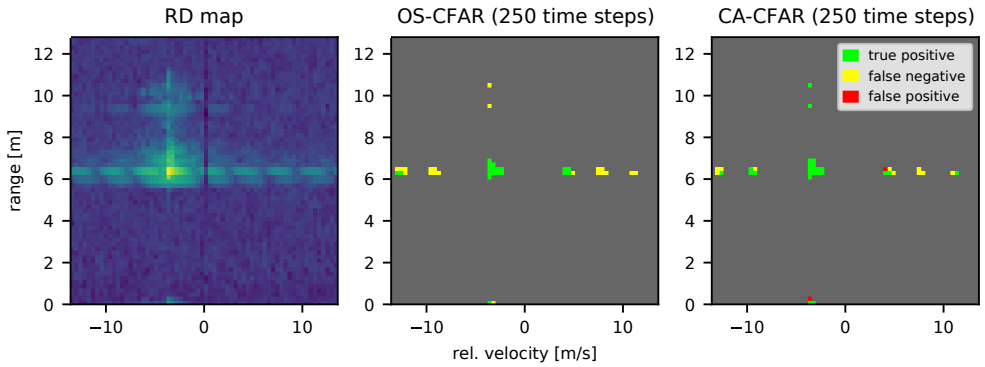


Figure 6.16: Result of processing the range-Doppler map on the left, which corresponds to a scene from the CARRADA dataset, with the S-OSCFAR (middle) and CA-CFAR (right), employing 250 simulation steps for each algorithm. The X and Y axis of the plot represent the range and Doppler dimensions. The true positive, false negative, and false positive detections are depicted in green, yellow, and red, respectively [Vog+22].

the processing pipeline on radar data, including the FFT and the OS-CFAR results The main purpose of this experiment was to test the implementation of the SNN on a SpiNNaker 2 board and deploy it in a real-world environment. A DSP pre-processed the signals from the sensor by computing an FFT. The SpiNNaker 2 board converted the FFT values to spike times and computed the OS-CFAR on those values.

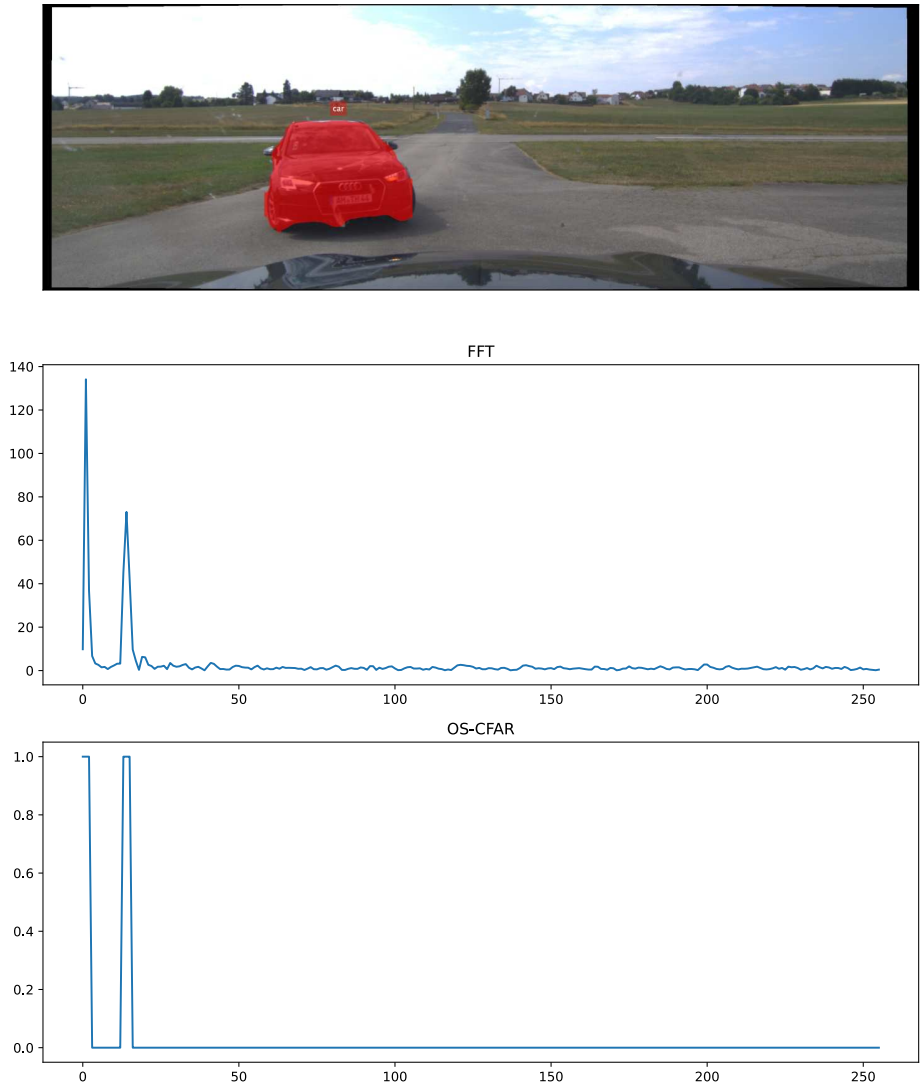


Figure 6.17: Sample result of the image object dectection, FFT and the OS-CFAR on one scenario from the KI-ASIC dataset.

6.4 Discussion

This thesis introduces SNN algorithms and an ASE circuit for processing FMCW radar data. The experiments were conducted on synthetic and real data with increasing noise levels and target occlusion. All blocks use latency-encoded spikes, so the data format across the generated pipeline remains consistent and does not generate incompatibilities.

Spiking Fourier transform

Section 6.1 describes the implementation and experiments on the S-FT. The architecture and parameters of the SNN are mathematically derived from the FT algorithm, so the expected output coincides with that of a standard FT. Moreover, contrary to black-box models, the absence of learning algorithms makes the algorithm predictable, so errors can be easily analysed and tracked. The experiments ran the S-FT and compared it with a software-based FFT, resulting in frequency spectra with high similarity. The obtained errors are partly due to the low precision of the data types used for the weights and the internal state of the neuron, and mainly to the number of simulation steps, as this value determines the output dynamic range. For example, the experiments depicted in fig. 6.5 correspond to an S-FT simulated for a spiking stage with 256 steps, i.e., the output can take 256 different values; whereas the reference FFT has 64-bit precision and thus a much larger dynamic range. Remarkably, this big difference in the output range does not lead to substantial errors in the frequency spectra. Moreover, this error barely propagates further in the post-processing of the S-FT in terms of a loss in the accuracy of the full pipeline, e.g., in the precision of an object detection task, such as in fig. 6.3. This behaviour is expected, as most of the information in the frequency spectrum of an FMCW radar signal is stored in the value of the dominant frequencies, not in the precise intensities of such frequencies. The intensities in a radar frequency spectrum fluctuate over time and are not a reliable source of information. When analyzing the frequency range of the S-FT, we observe that it is identical to the FFT used for comparison, because the number of frequencies only depends on the number of neurons used to implement the S-FT.

Besides the number of simulation steps, tuning the S-FT also involves setting a value for u_{th}. A high value decreases the sensitivity to low-intensity frequencies, and a low value caps the maximum intensity a frequency component can yield. Even though a careful study of the sensor and the environment it will scan would lead to a better-suited value for u_{th}, we can make a well-educated guess without prior knowledge of the problem due to the nature of the FT, as it is a bounded function, i.e., there is a range of values it cannot reach in real-world problems (see section 6.1.1). Moreover, capping high intensities is generally not a problem when generating a frequency spectrum, as it can improve the sensitivity to low-intensity frequencies (see fig. 6.1). The experiments with the S-FT not only tested the SNN with a single-layer, but also multidimensional architectures, like the SNN used for computing a range-Doppler map (fig. 6.6) and the SNN used for computing a radix-4 FFT architecture (fig. 6.3). The corresponding results do not show a noticeable difference in the error introduced by multi-layer structures.

The implementation of the S-FT was performed on boards from the Loihi and SpiNNaker families. Figure 6.7 shows a comparison in the performance of the SNN in the Loihi 2 and SpiNNaker 2 boards. Whereas the results of the SpiNNaker 2 implementation are identical to the simulation made on a PC, the results with Loihi 2 show a deviation from the reference. Even though this deviation is noticeable relative to the magnitude of the error, it is small in absolute terms. At the point of writing this document, it was not possible to determine the source of this misalignment, so a future study should find the reason and, if possible, improve the implementation. It is also important to note that the size of the S-FT was limited by the number of cores available in the chips. The state of the software libraries for the SpiNNaker 2 and Loihi 2 boards did not allow conducting a detailed measurement of the energy consumption for the S-FT implementations. Instead, table 6.3 offers energy and time estimations based on the study in [Dav+18] for the Loihi 1 chip. These energy and latency footprints of the S-FT are worse than the conventional FFT accelerators summarized in table 2.1. The close resemblance of the S-FT to the original FT, together with having a predictable output, makes it a potential candidate for computing the FT more efficiently and processing FMCW radar signals with neuromorphic hardware. The suitability of such implementation would

depend on the final footprint of the system and its compatibility with the rest of the processing chain. The early stage in the development of neuromorphic hardware technology leaves a big margin for improvement to achieve a competitive performance.

Analogue-to-spike encoder

The successful experiments with the S-FT motivated the design of the ASE introduced in chapter 5 as a means of overcoming the need for an ADC and having a circuit that could directly convert analogue signals to spikes compatible with the S-FT. Section 6.2 covers the implementation steps for the ASE, including the elaboration of the circuit prototype and the programming of the microcontroller used to collect the circuit's output. The circuit was tested on different electric signals obtained in an electronics lab. The conducted experiments let us assess the behaviour of the circuit and its feasibility for use as a replacement for an ADC.

The experiments were split into three batches aimed at evaluating the encoding accuracy at different voltage levels, the encoding accuracy when using a linear decoder, and the FT performance when running the S-FT afterward. The first two experiments yielded low encoding error, which led to a correct reconstruction of the original signal. The last experiment showed that the ASE is compatible with the S-FT: The output spikes of the ASE were directly fed to a Loihi 2 chip with any further modification, and the chip was able to compute the S-FT with the provided data, as shown in fig. 6.13. The S-FT plots also show artifacts introduced due to the mismatch in the signal encoding between the ASE and the S-FT. This mismatch is inherent to the nature of the two models. A straightforward strategy for minimizing this phenomenon is to tune the circuit so that the ratio $u_{\text{th}}/u_{\text{min}}$ is as small as possible. However, this strategy has limitations, as the values for u_{th} and u_{min} have physical boundaries, and increasing u_{min} leads to a worse dynamic range and latency response in the ASE. A second strategy is to predict where the artifacts will occur and denoise the resulting signal accordingly. Although feasible, this approach would require additional processing and would add computational burden to the pipeline. A third alternative would be to modify either the S-FT or the ASE models so the encoding of signals into spikes is the same for both. One interesting approach in this regard would be to use an ASE based on a voltage-to-rate encoder that linearly maps voltages into an instantaneous spike rate. Thus, the measurement of the *inter-spike interval* (ISI) would reflect the input value at any time, and the model would resemble the encoding in the S-FT. The resulting ASE would resemble the initial stage of a $\Delta - \Sigma$ ADC architecture, namely the Δ circuit.

The ASE transforms analogue signals into a single stream of spikes, where each spike represents the value of the signal during a specific sampling time. For N samples, the input layer of the S-FT SNN introduced in section 4.2 is formed by N neurons, where each neuron has N input synapses, from which the nth input synapse provides one spike containing information about the nth sample. The original design of the S-FT assumes that data from the N input samples arrives in parallel, after the sampling and digitisation that takes place at the ADC. Thus, the interface of the ASE to the S-FT needs to multiplex the spikes provided by the ASE over time and combine each spike with its corresponding weight.

Object detection and pipeline ensemble

The final set of experiments, described in section 6.3, assessed the performance of the S-OSCFAR. As phase-encoded spikes representing higher values arrive earlier , applying a method based on ordered statistics comes naturally for the output of the S-FT. Thus, the S-OSCFAR can be easily implemented on a digital neuromorphic chip, with a connection grid and weight magnitudes considerably simpler than those of the S-FT. The SNN was tested on automotive data, and the results showed an accuracy equivalent to that of a standard

OS-CFAR (see fig. 2.7). When analyzing these results together with the full-pipeline results for the S-FT in fig. 6.3, we observe that both algorithms yield an accuracy similar to the equivalent conventional pipelines. Moreover, when performing a qualitative evaluation of the results of a specific scene, like the one in fig. 6.16, we can observe that the missed detections tend to correspond to outlier points or weak reflections that are typically disregarded when running a clustering algorithm.

The results described in this chapter for the ASE, along with the S-FT and the S-OSCFAR, open the door to the creation of end-to-end neuromorphic pipelines for processing FMCW radar data. Altogether, the experiments show that the three stages are compatible with each other and that they yield results similar to their original counterparts. To evaluate whether the proposed models are a competitive alternative, further research in neuromorphic hardware should enable real-time processing of the spikes generated by the ASE by creating a low-level interface that can read and process the spike train. Moreover, the ASE needs to be integrated into a more efficient physical form that allows for smaller components and thus faster latencies. The energy and latency measurements on such final assembly would be crucial to determine the real potential of the ASE together with the S-FT. One current limitation of the interface between the S-FT and the S-OSCFAR is the need for switching from the polar dimensions of the S-FT to the absolute values typically employed by CFAR algorithms. Moreover, the magnitude values would have to be mapped to a logarithmic scale for optimizing the performance of the S-OSCFAR. The best solution would be to develop an intermediate stage that performs these two operations on the spikes arriving from the S-FT. Such hybrid computation is possible in chips like Loihi 2 or SpiNNaker 2, as they are actually prepared for combining spiking operations with conventional floating point arithmetic.

The application of a logarithmic transform to the input of the S-OSCFAR is particularly interesting because of its similarity to the way biological systems apply scaling maps to stimuli to enhance the dynamic range. For instance, retinal cells in the human eye provide a logarithmic response Ψ to retinal illumination I, following the equation $\Psi = k \log(I/I_0)$, known as *Fechner law*, where k is a proportionality constant and I_0 is the neural noise level under dark light conditions. The auditory system applies a similar form of logarithmic compression through the hair cells in the cochlea, which convert sound vibrations into electrical signals. The effect of this scaling can be observed in fig. 6.15, which shows how switching to a logarithmic mapping of the input significantly improves the S-OSCFAR accuracy. In contrast, the output of the S-FT follows a linear relationship with the input signal. Applying a logarithmic transformation requires an arithmetic unit to perform the operation. Achieving an intrinsic logarithmic response requires modifying the neuron model. The adaptive photoreceptor circuit in event-based sensors may serve as inspiration for this redesign [LPD08].

The primary motivation of this work was to contribute to the development of an end-to-end neuromorphic pipeline that takes sensor voltage as input and provides a high-end output such as target clusters or object labels. To reach that end, the proposed processing elements would have to be deployed in the same system to process data in real-time. Figure 6.18 represents how data would be transmitted for such system across the different processing elements over the time dimension for a signal chirp: The ASE encodes voltage data in real-time into a spike train split into N identical time intervals. The spikes are buffered and processed in parallel by an S-FT with N input channels, resulting in N spikes representing the frequency spectrum. Finally, the S-OSCFAR produces a spike for the bins where a peak is detected, staying silent otherwise. When this work was conducted, these blocks could not be implemented together, as neuromorphic chips were limited in size and lacked interfaces for acquiring low-level signals and processing them in real-time, i.e., a single-bit port for collecting the spikes generated by the ASE. Figure 6.18 shows a snapshot of how a final implementation with all introduced processing stages would look. One attractive property of

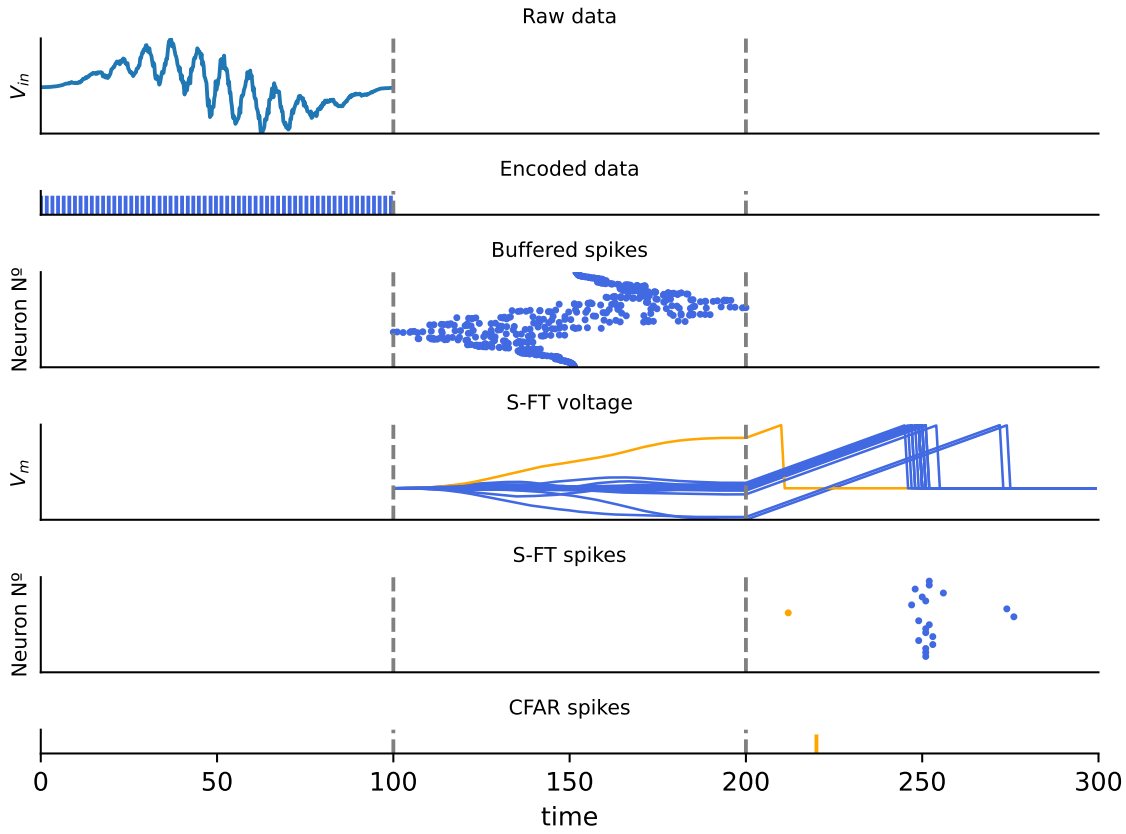


Figure 6.18: Simulation of the different stages of the spiking pipeline for FMCW radar. On top is the sensor's raw data for one chirp with 512 samples. The second plot represents the output of the ASE after feeding the sensor voltage. The third plot shows the spikes generated by the ASE after being buffered and forwarded in a parallel fashion. The fourth and fifth plots show the evolution of the S-FT's membrane voltage and spike output, respectively. The fifth plot shows the output spike train of the S-OSCFAR. The S-FT and S-OSCFAR plots are shown for a single CFAR window with one CUT and 20 neighbour cells, represented in orange and blue, respectively.

this pipeline is the structure in independent layers and the sequential processing of arriving data, which is a generalization of the multi-layer architecture property of the S-FT described in section 4.1.3. An important implication is that after a pipeline stage computes data from one chirp, it can start computing data from the next chirp before the following pipeline stages finish their task with previous chirps. The pipeline is not limited to the described algorithms, and it could be modified so it can provide a more sophisticated output, including tasks like clustering, classification, detection of the angle-of-arrival, etc. The study of these algorithms and their compatibility with the introduced SNNs is outside the scope of this dissertation.

The neuromorphic pipeline shown in fig. 6.18 is showcased for FMCW radar data and the experiments for the different stages ran with data from simulated or real radar sensors, or electric signals with a similar nature. In any case, the application of the algorithms presented in this work is not limited to the radar case, as they can be adapted to different sensors and input signals. The specific algorithms and architectures used are entirely dependent on the signal-processing problem at hand. For some sensors, like FMCW lidar, the transfer of the algorithms would require adjusting to the data acquisition rates and signal intensities. Moreover, the same sensor may require a different tuning of the algorithms, depending on the specific tasks to be solved, the number of dimensions that are relevant to the problem, and the characteristics of the scenarios that will be sensed.

7

Conclusion

Sensor signal processing is an appealing application area for neuromorphic computing. Many sensors generate raw data in the form of electric currents, where information is encoded in their changes over time. Moreover, data often arrives from multiple channels, requiring parallel processing. As these applications are frequently implemented in embedded or autonomous systems with limited energy resources, optimising their efficiency is crucial. Neuromorphic computing algorithms provide competitive solutions by taking advantage of the parallel, asynchronous, and low-power characteristics of neuromorphic hardware. Automotive radar is an appealing sensor for processing with neuromorphic algorithms, due to the highly parallel and temporal nature of its data, as well as the significant need to reduce the energy footprint in automotive systems.

This work introduces an end-to-end neuromorphic computing pipeline for processing the raw signal of *frequency-modulated continuous-wave* (FMCW) radar sensors. The components designed within this pipeline address the crucial tasks of analogue-to-digital conversion, frequency-domain representation, and object detection, representing data with the precise times of spikes. The *analogue-to-spike encoder* (ASE) replaces the analogue-to-digital conversion by encoding analogue signals into a stream of spikes, where each spike corresponds to the voltage intensity during the equivalent sampling time. The *spiking Fourier transform* (S-FT) loads in parallel time-encoded spikes representing entire radar frames and computes their frequency spectrum into time-encoded spikes. For object detection, the *spiking OSCFAR* (S-OSCFAR) segments data from the S-FT using a winner-takes-all neuron model, where arriving spikes compete for being selected as targets based on their occurrence times. All elements were implemented in hardware and tested with actual data, i.e., the ASE was implemented as an analogue circuit, and the rest of the pipeline was formed by *spiking neural networks* (SNNs) that were implemented in the digital neuromorphic chips SpiNNaker 2 and Loihi 2. The experiments assessed the performance of these components under varying levels of difficulty using synthetic data and electric signals generated in the lab. Results showed accuracy comparable to well-established methods. A final batch of experiments validated the performance on data recorded on a sensor used on real-driving scenarios.

The scope of the presented methods is not limited to radar sensors: the ubiquity of the Fourier transform and the similar computing principle of FMCW radar with other sensors facilitate the migration of this work to other processing chains like LiDAR and audio sensors, or medical imaging signals like electrocardiograms. Moreover, the *temporal charge before spike* (TCBS) neuron model for computing the S-FT also serves as a lossless time-based model for converting *deep convolutional neural networks* (DCNNs) to SNNs and accelerating the inference in neuromorphic hardware (see appendix D).

This work pioneers processing FMCW radar sensors with neuromorphic computing methods, aiming to spark the design of novel event-based circuits and algorithms that can accelerate edge applications with small footprints and low latencies. The S-FT proves, for the first time, that the frequency spectrum can be generated using an SNN and sets a reference point for future research on frequency domain representation using neuromorphic computing. Additionally, the experiments with the ASE show that the signal processing with the S-FT does not require an *analog-to-digital converter* (ADC). Instead, this dedicated component can directly switch from the analogue to the spiking realms. This work also contributes to establishing time-based spike encoding as a solid alternative to rate encoding representations, which is crucial for unleashing the full potential of neuromorphic hardware. Finally, one key aspect of these methods is that they do not use any learning or optimisation technique, so their behaviour is deterministic and do not pose the inherent risk in machine learning methods due to the lack of knowledge on the learnt models. This characteristic is paramount for safety-critical applications where the operation cannot rely on black-box models that yield performances within a probability range and cannot predict the scenarios where they will fail.

The impact of this work is threefold: it shows it is possible to compute the frequency-domain representation using spikes, provides an end-to-end neuromorphic processing chain for FMCW for the first time, and stimulates the application of sparse, time-coded SNNs. The introduced models and implementations are not end products but proofs of concept that aim to encourage researchers to optimise neuromorphic systems further to provide a greener and more efficient means of processing the ever-increasing amount of data generated by more intelligent and widespread applications.

7.1 Challenges and limitations

Implementing the ASE and the S-FT presented several limitations, mainly due to constraints imposed by the hardware. The specifications of the neuromorphic chips determined the precision of the weights and the maximum size of the S-FT, which in turn limited testing to one SNN at a time, i.e., the S-FT and S-OSCFAR were compiled separately, with the output of the S-FT recorded and used as input of the S-OSCFAR. Furthermore, the chips' interfaces lack ports for directly interfacing low-level signals, requiring the S-FT to be executed in a batch fashion. In practice, this meant that the input spikes were pre-recorded and sent offline from a host PC to the SNN via a high-level ethernet interface, preventing real-time testing of the pipeline. Another limitation was the missing capability of conducting an energy profiling of the SNNs with the current chip versions. These limitations are not unique to this work but a recurring gap in the literature. Neuromorphic research often focuses on detailed, in-lab studies of small-scale scenarios that are not realistic or easily transferable to real world applications, or in proof-of-concept examples where accuracy and performance cannot be carefully evaluated. Regarding the ASE, even though circuit simulations show a potential improvement in energy consumption compared to that of ADCs, the prototyping nature of the lab experiments does not clarify whether this component can replace an ADC for the high sampling frequencies that radar chips require.

From a computational perspective, an essential limitation was the accuracy of the measurements. The resolution of the ASE and the S-FT is determined by the time-step precision of the implementation. In contrast with the floating point precision of traditional ADCs and *Fourier transform* (FT) accelerators, the neuron models encode the information at the precise timing of the spikes. Even though experiments show that this drastic reduction in the resolution does not impact key performance indicators like precision and recall, future im-

plementations of these models must carefully evaluate these metrics under realistic scenarios that address the challenges of the application at hand. One potential alternative would be implementing the computation of the S-FT neurons using analogue circuits that do not rely on discrete time steps. In such an approach, the resolution limits would be determined by the sampling capacities of the communications interface with the following processes. The presented models aim to improve the time and energy performance of the signal processing pipeline at the cost of small accuracy losses. Even though these losses are negligible for many applications, tasks requiring precise measurements should not rely on these methods in their current state. Although the experimental results show an accuracy up to standard for processing radar signals, the lack of a full picture in terms of energy consumption and computing latency makes it difficult to state whether the proposed neuromorphic pipeline can improve the performance of traditional signal processing approaches.

7.2 Future work

While this work demonstrates the feasibility of radar signal processing using neuromorphic computing approaches, future research should focus on refining some aspects and clarifying open questions. The current research does not provide a definitive answer to whether neuromorphic computing can beat traditional approaches in radar signal processing. The conducted experiments took place on general-purpose digital neuromorphic chips, which offer a suitable platform for the prototyping and validation of the algorithms' accuracy. However, to measure the actual impact in terms of energy performance and time efficiency it is crucial to implement a precise and detailed benchmark to evaluate the capabilities of neuromorphic systems against traditional signal processing methods. Moreover, a comprehensive evaluation should also include the analysis of a full-pipeline implementation instead of the evaluation of isolated blocks. This benchmarking should also focus on the assessment under real-conditions, by performing test-driving scenarios and assessing real-time performance and robustness towards interferences and outliers present on real-world, unconstrained experiments.

Regarding the computational paradigm, while the current research focused on spiking neurons, the best solution for a full radar pipeline is not necessarily fully spiking. Future research should explore hybrid approaches that combine the benefits of spiking and neural dynamics with the flexibility and robustness of some traditional signal processing operations with floating point arithmetic. A clear example is the benefit of applying a logarithmic conversion to the output spikes of the S-FT before feeding them to the S-OSCFAR, as shown in fig. 6.15. Neuromorphic digital chips like SpiNNaker and Loihi offer the flexibility to implement such hybrid systems by incorporating non-spiking operations between neural blocks. Furthermore, while proposed algorithms replicate existing solutions using spikes, a future iteration of this work should exploit neural dynamics for yielding faster and more efficient solutions. The work in [Ree+25] explores this line of research by employing the *resonate-and-fire* (RF) model from Izhikevich [Izh01] for obtaining early estimations of the frequency spectrum. In the proposed network, each neuron is sensitive to a specific frequency and gets excited before the full frame is processed. Moreover, the RF neuron model has the advantage that it can be directly applied to voltage signals, so the conversion to spikes happens naturally within the neuron circuit. Additionally, these neural dynamics can be leveraged for extracting other important information, such as the angle of arrival *angle of arrival* (AoA) of radar signals, offering an alternative to the beamforming operation. Alternatively, further work on the S-FT should focus on increasing the sparsity of the model by limiting the neurons that

can produce spikes within a single frame, similar to the concept of sparse Fourier transforms (See section 2.3). A similar result could be obtained using a winner-takes-all approach where only the first neurons to get excited are allowed to spike, or making the neurons' threshold adaptive over time. The TCBS neuron model could also be applied for mapping algorithms other than the FT, such as the wavelet transform.

Regarding the ASE, future work should modify the encoder to map voltages to spikes employing a spiking rule that is fully compatible with the S-FT, so to avoid the artifacts in the frequency spectrum due to the mismatch between the encoder and the S-FT model. A promising approach would be the removal of the leak from the neuron using a frequency-to-spike converter similar to the sigma stage in sigma-delta ADCs. Additionally, future research should design and implement the ASE as an integrated circuit, with the final goal of achieving very-large-scale integration. Such an implementation would make possible to fully evaluate its potential in comparison to using conventional ADCs.

Finally, the scope of future work should also focus on the following stages in the processing pipeline and extend beyond radar signal processing. The development of alternatives for computing higher-level tasks such as clustering, tracking, and classification is fundamental for obtaining full neuromorphic pipelines that can offer a competitive alternative to traditional signal processing pipelines. While radar systems have been the focus of this research, neuromorphic approaches have also shown potential for processing data from other types of sensors such as event-based cameras and LiDAR. Combining data from multiple sensor modalities using neuromorphic architectures could result in a robust and efficient alternative to sensor fusion.

Bibliography

[Abb99] Abbott, L. F. “Lapicque’s introduction of the integrate-and-fire model neuron (1907)”. In: *Brain research bulletin* 50.5-6 (1999), pp. 303–304.

[AM27] Adrian, E. D. and Matthews, R. “The action of light on the eye: Part I. The discharge of impulses in the optic nerve and its relation to the electric changes in the retina”. In: *The Journal of Physiology* 63.4 (1927), p. 378.

[Afs+20] Afshar, S., Hamilton, T. J., Davis, L., Van Schaik, A., and Delic, D. “Event-based processing of single photon avalanche diode sensors”. In: *IEEE Sensors Journal* 20.14 (2020), pp. 7677–7691.

[Aim+22] Aimone, J., Date, P., Fonseca-Guerra, G., Hamilton, K., Henke, K., Kay, B., Kenyon, G., Kulkarni, S., Mniszewski, S., Parsa, M., et al. “A review of non-cognitive applications for neuromorphic computing”. In: *Neuromorphic Computing and Engineering* (2022).

[Aug+21a] Auge, D., Hille, J., Kreutz, F., Mueller, E., and Knoll, A. “End-to-end spiking neural network for speech recognition using resonating input neurons”. In: *International Conference on Artificial Neural Networks*. Springer. 2021, pp. 245–256.

[Aug+21b] Auge, D., Hille, J., Mueller, E., and Knoll, A. “A survey of encoding techniques for signal processing in spiking neural networks”. In: *Neural Processing Letters* 53.6 (2021), pp. 4693–4710.

[BK96] Bair, W. and Koch, C. “Temporal Precision of Spike Trains in Extrastriate Cortex of the Behaving Macaque Monkey”. In: *Neural Computation* 8.6 (1996), pp. 1185–1202. ISSN: 0899-7667. DOI: 10.1162/neco.1996.8.6.1185.

[Bao+22] Bao, H., Zhou, H., Li, J., Pei, H., Tian, J., Yang, L., Ren, S., Tong, S., Li, Y., He, Y., et al. “Toward memristive in-memory computing: principles and applications”. In: *Frontiers of Optoelectronics* 15.1 (2022), p. 23.

[BP12] Barth, A. L. and Poulet, J. F. “Experimental evidence for sparse firing in the neocortex”. In: *Trends in neurosciences* 35.6 (2012), pp. 345–355.

[Bas+22] Basu, A., Deng, L., Frenkel, C., and Zhang, X. “Spiking neural network integrated circuits: A review of trends and future directions”. In: *2022 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2022, pp. 1–8.

[Beh+17] Behroozpour, B., Sandborn, P. A., Wu, M. C., and Boser, B. E. “Lidar system architectures and circuits”. In: *IEEE Communications Magazine* 55.10 (2017), pp. 135–142.

[Ben+23] Bensimon, M., Hadad, Y., Ben-Shimol, Y., and Greenberg, S. “Time-Frequency Analysis for Feature Extraction Using Spiking Neural Network”. In: (2023).

[BF10] Bhattacharya, B. S. and Furber, S. B. “Biologically inspired means for rank-order encoding images: A quantitative analysis”. In: *IEEE transactions on neural networks* 21.7 (2010), pp. 1087–1099.

[Bia+89] Bialek, W., Rieke, F., Steveninck, R. van, and Warland, D. “Reading a neural code”. In: *Advances in neural information processing systems* 2 (1989).

[Bin+18] Bing, Z., Meschede, C., Röhrbein, F., Huang, K., and Knoll, A. C. “A survey of robotics control based on learning-inspired spiking neural networks”. In: *Frontiers in neurorobotics* 12 (2018), p. 35.

[Blu+17] Blum, K. P., Lamotte D’Incamps, B., Zytnicki, D., and Ting, L. H. “Force encoding in muscle spindles during stretch of passive muscle”. In: *PLoS computational biology* 13.9 (2017), e1005767.

[BT99] Borst, A. and Theunissen, F. E. “Information theory and neural coding”. In: *Nature neuroscience* 2.11 (1999), pp. 947–957.

[Bra+12] Brasselet, R., Panzeri, S., Logothetis, N. K., and Kayser, C. “Neurons with Stereotyped and Rapid Responses Provide a Reference Frame for Relative Temporal Coding in Primate Auditory Cortex”. In: *The Journal of Neuroscience* 32.9 (2012), pp. 2998–3008. ISSN: 0270-6474. DOI: 10.1523/jneurosci.5435-11.2012.

[Bre15] Brette, R. “Philosophy of the spike: rate-based vs. spike-based theories of the brain”. In: *Frontiers in systems neuroscience* 9 (2015), p. 151.

[BG05] Brette, R. and Gerstner, W. “Adaptive exponential integrate-and-fire model as an effective description of neuronal activity”. In: *Journal of neurophysiology* 94.5 (2005), pp. 3637–3642.

[Bri20] Briggs, F. “Role of feedback connections in central visual processing”. In: *Annual review of vision science* 6 (2020), pp. 313–334.

[BS12] Bullmore, E. and Sporns, O. “The economy of brain network organization”. In: *Nature reviews neuroscience* 13.5 (2012), pp. 336–349.

[CF00] Carandini, M. and Ferster, D. “Membrane potential and firing rate in cat primary visual cortex”. In: *Journal of Neuroscience* 20.1 (2000), pp. 470–484.

[CK90] Carr, C. and Konishi, M. “A circuit for detection of interaural time differences in the brain stem of the barn owl”. In: *Journal of Neuroscience* 10.10 (1990), pp. 3227–3246.

[CLS07] Chan, V., Liu, S.-C., and Schaik, A. van. “AER EAR: A matched silicon cochlea pair with address event representation interface”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.1 (2007), pp. 48–59.

[Che+20] Chen, G., Cao, H., Conradt, J., Tang, H., Rohrbein, F., and Knoll, A. “Event-based neuromorphic vision for autonomous driving: A paradigm shift for bio-inspired visual sensing and perception”. In: *IEEE Signal Processing Magazine* 37.4 (2020), pp. 34–49.

[Che+18] Chen, X., Lei, Y., Lu, Z., and Chen, S. “A variable-size FFT hardware accelerator based on matrix transposition”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.10 (2018), pp. 1953–1966.

[Chi+14] Chicca, E., Stefanini, F., Bartolozzi, C., and Indiveri, G. “Neuromorphic electronic circuits for building autonomous cognitive systems”. In: *Proceedings of the IEEE* 102.9 (2014), pp. 1367–1388.

[Chk04] Chklovskii, D. B. “Synaptic connectivity and neuronal morphology: two sides of the same coin”. In: *Neuron* 43.5 (2004), pp. 609–617.

[Chr+22] Christensen, D. V., Dittmann, R., Linares-Barranco, B., Sebastian, A., Le Gallo, M., Redaelli, A., Slesazeck, S., Mikolajick, T., Spiga, S., Menzel, S., et al. "2022 roadmap on neuromorphic computing and engineering". In: *Neuromorphic Computing and Engineering* 2.2 (2022), p. 022501.

[CH06] Clark, B. and Häusser, M. "Neural coding: hybrid analog and digital signalling in axons". In: *Current biology* 16.15 (2006), R585–R588.

[Com+20] Comsa, I. M., Potempa, K., Versari, L., Fischbacher, T., Gesmundo, A., and Alakuijala, J. "Temporal coding in spiking neural networks with alpha synaptic function". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 8529–8533.

[CT65] Cooley, J. W. and Tukey, J. W. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[CI15] Corradi, F. and Indiveri, G. "A neuromorphic event-based neural recording system for smart brain-machine-interfaces". In: *IEEE transactions on biomedical circuits and systems* 9.5 (2015), pp. 699–709.

[DJ22] D'Angelo, E. and Jirsa, V. "The quest for multiscale brain modeling". In: *Trends in neurosciences* (2022).

[DF21] Davidson, S. and Furber, S. B. "Comparison of artificial and spiking neural networks on digital hardware". In: *Frontiers in Neuroscience* 15 (2021), p. 651141.

[Dav+18] Davies, M., Srinivasa, N., Lin, T.-H., Chinya, G., Cao, Y., Choday, S. H., Dimou, G., Joshi, P., Imam, N., Jain, S., et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[Dav+21] Davies, M., Wild, A., Orchard, G., Sandamirskaya, Y., Guerra, G. A. F., Joshi, P., Plank, P., and Risbud, S. R. "Advancing neuromorphic computing with loihi: A survey of results and outlook". In: *Proceedings of the IEEE* 109.5 (2021), pp. 911–934.

[DA01] Dayan, P. and Abbott, L. F. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. Computational Neuroscience Series, 2001.

[DLR21] De Zeeuw, C. I., Lisberger, S. G., and Raymond, J. L. "Diversity and dynamism in the cerebellum". In: *Nature neuroscience* 24.2 (2021), pp. 160–167.

[deC98] deCharms, R. C. "Information coding in the cortex by independent or coordinated populations". In: *Proceedings of the National Academy of Sciences* 95.26 (1998), pp. 15166–15168.

[Dee+20] Deep, Y., Held, P., Ram, S. S., Steinhauser, D., Gupta, A., Gruson, F., Koch, A., and Roy, A. "Radar cross-sections of pedestrians at automotive radar frequencies using ray tracing and point scatterer modelling". In: *IET Radar, Sonar & Navigation* 14.6 (2020), pp. 833–844.

[DA23] Delic, D. and Afshar, S. "Neuromorphic Computing for Compact LiDAR Systems". In: *More-than-Moore Devices and Integration for Semiconductors*. Springer, 2023, pp. 191–240.

[Den+20] Deng, L., Wang, G., Li, G., Li, S., Liang, L., Zhu, M., Wu, Y., Yang, Z., Zou, Z., Pei, J., et al. "Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation". In: *IEEE Journal of Solid-State Circuits* 55.8 (2020), pp. 2228–2246.

[Die+16] Diehl, P. U., Zarrella, G., Cassidy, A., Pedroni, B. U., and Neftci, E. "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware". In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE. 2016, pp. 1–8.

[Dom+18] Dominguez-Morales, J. P., Liu, Q., James, R., Gutierrez-Galan, D., Jimenez-Fernandez, A., Davidson, S., and Furber, S. "Deep spiking neural network model for time-variant signals classification: a real-time speech recognition approach". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[DHX18] Dong, M., Huang, X., and Xu, B. "Unsupervised speech recognition through spike-timing-dependent plasticity in a convolutional spiking neural network". In: *PloS one* 13.11 (2018), e0204596.

[DS00] Dongarra, J. and Sullivan, F. "Guest Editors Introduction to the top 10 algorithms". In: *Computing in Science & Engineering* 2.01 (2000), pp. 22–23.

[DMM95] Douglas, R., Mahowald, M., and Mead, C. "Neuromorphic analogue VLSI". In: *Annual review of neuroscience* 18.1 (1995), pp. 255–281.

[Eck+18] Eckhardt, J. M., Joram, N., Figueroa, A., Lindner, B., and Ellinger, F. "FMCW multiple-input multiple-output radar with iterative adaptive beamforming". In: *IET Radar, Sonar & Navigation* 12.11 (2018), pp. 1187–1195.

[Est+96] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[Fit61] FitzHugh, R. "Impulses and physiological states in theoretical models of nerve membrane". In: *Biophysical journal* 1.6 (1961), pp. 445–466.

[For+22] Forno, E., Fra, V., Pignari, R., Macii, E., and Urgese, G. "Spike encoding techniques for IoT time-varying signals benchmarked on a neuromorphic classification task". In: *Frontiers in Neuroscience* 16 (2022), p. 999029.

[FF10] Fraden, J. and Fraden, J. *Handbook of modern sensors: physics, designs, and applications*. Vol. 3. Springer, 2010.

[FBI21] Frenkel, C., Bol, D., and Indiveri, G. "Bottom-Up and Top-Down Neural Processing Systems Design: Neuromorphic Intelligence as the Convergence of Natural and Artificial Intelligence". In: *arXiv preprint arXiv:2106.01288* (2021).

[Fre+18] Frenkel, C., Lefebvre, M., Legat, J.-D., and Bol, D. "A 0.086-mm2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS". In: *IEEE transactions on biomedical circuits and systems* 13.1 (2018), pp. 145–158.

[Fur+04] Furber, S. B., Bainbridge, W. J., Cumpstey, J. M., and Temple, S. "Sparse distributed memory using N-of-M codes". In: *Neural Networks* 17.10 (2004), pp. 1437–1451.

[Fur+14] Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. "The spinnaker project". In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[Ger+22] Gerhards, P., Kreutz, F., Knobloch, K., and Mayr, C. G. "Radar-Based Gesture Recognition with Spiking Neural Networks". In: *2022 7th International Conference on Frontiers of Signal Processing (ICFSP)*. IEEE. 2022, pp. 40–44.

[Ger+14] Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. *Neuronal Dynamics - From Single Neurons to Networks and Models of Cognition*. Cambridge: Cambridge University Press, 2014. ISBN: 978-1-107-63519-7.

[Gil+14] Gilbert, A. C., Indyk, P., Iwen, M., and Schmidt, L. "Recent developments in the sparse Fourier transform: A compressed Fourier transform for big data". In: *IEEE Signal Processing Magazine* 31.5 (2014), pp. 91–100.

[GM08] Gollisch, T. and Meister, M. "Rapid neural coding in the retina with relative spike latencies". In: *science* 319.5866 (2008), pp. 1108–1111.

[Göl+21] Göltz, J., Kriener, L., Sabado, V., and Petrovici, M. A. "Fast and Energy-efficient deep Neuromorphic Learning". In: *Brain-inspired Computing* (2021), p. 17.

[Gon+21] Gonzalez, H. A., Kelber, F., Stolba, M., Liu, C., Vogginger, B., Hänzsche, S., Scholze, S., Höppner, S., and Mayr, C. "Ultra-high compression of twiddle factor ROMs in multi-core DSP for FMCW radars". In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2021, pp. 1–5.

[Gra95] Graps, A. "An introduction to wavelets". In: *IEEE computational science and engineering* 2.2 (1995), pp. 50–61.

[Guo+14] Guo, L., Tang, Y., Lei, Y., Dou, Y., and Zhou, J. "Transpose-free variable-size FFT accelerator based on-chip SRAM". In: *IEICE Electronics Express* 11.15 (2014), pp. 20140171–20140171.

[GQH07] Guo, X., Qi, X., and Harris, J. G. "A time-to-first-spike CMOS image sensor". In: *IEEE Sensors Journal* 7.8 (2007), pp. 1165–1175.

[HG18] Haegens, S. and Golumbic, E. Z. "Rhythmic facilitation of sensory processing: A critical review". In: *Neuroscience & Biobehavioral Reviews* 86 (2018), pp. 150–165.

[Has+12] Hasch, J., Topak, E., Schnabel, R., Zwick, T., Weigel, R., and Waldschmidt, C. "Millimeter-wave technology for automotive radar sensors in the 77 GHz frequency band". In: *IEEE Transactions on Microwave Theory and Techniques* 60.3 (2012), pp. 845–860.

[HN98] Hawkes, M. and Nehorai, A. "Acoustic vector-sensor beamforming and Capon direction estimation". In: *IEEE transactions on signal processing* 46.9 (1998), pp. 2291–2304.

[HA16] Hawkins, J. and Ahmad, S. "Why neurons have thousands of synapses, a theory of sequence memory in neocortex". In: *Frontiers in neural circuits* (2016), p. 23.

[Her+06] Herz, A. V., Gollisch, T., Machens, C. K., and Jaeger, D. "Modeling single-neuron dynamics and computations: a balance of detail and abstraction". In: *science* 314.5796 (2006), pp. 80–85.

[HSK19] Heuvel, M. P. van den, Scholtens, L. H., and Kahn, R. S. "Multiscale neuroscience of psychiatric disorders". In: *Biological Psychiatry* 86.7 (2019), pp. 512–522.

[HK98] Hinneburg, A. and Keim, D. A. "An efficient approach to clustering in large multimedia databases with noise". In: *Knowledge Discovery and Datamining (KDD'98)*. 1998, pp. 58–65.

[HS97] Hochreiter, S. and Schmidhuber, J. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[Hol+03] Holmgren, C., Harkany, T., Svennenfors, B., and Zilberter, Y. “Pyramidal cell communication within local networks in layer 2/3 of rat neocortex”. In: *The Journal of physiology* 551.1 (2003), pp. 139–153.

[HSS16] Horne, C. D., Sumner, C. J., and Seeber, B. U. “A phenomenological model of the electrically stimulated auditory nerve fiber: temporal and biphasic response properties”. In: *Frontiers in computational neuroscience* 10 (2016), p. 8.

[HE16] Hunsberger, E. and Eliasmith, C. “Training spiking deep networks for neuromorphic hardware”. In: *arXiv preprint arXiv:1611.05141* (2016).

[IEA24] IEA. “Electricity 2024”. In: (2024).

[Ind21] Indiveri, G. “Introducing ‘neuromorphic computing and engineering’”. In: *Neuromorphic Computing and Engineering* 1.1 (2021), p. 010401.

[IH11] Indiveri, G. and Horiuchi, T. K. *Frontiers in neuromorphic engineering*. 2011.

[Ind+11] Indiveri, G., Linares-Barranco, B., Hamilton, T. J., Schaik, A. v., Etienne-Cummings, R., Delbruck, T., Liu, S.-C., Dudek, P., Häfliger, P., Renaud, S., et al. “Neuromorphic silicon neuron circuits”. In: *Frontiers in neuroscience* 5 (2011), p. 73.

[Iñi23] Iñiguez, B. “Flexible and Printed Electronics”. In: *More-than-Moore Devices and Integration for Semiconductors*. Springer, 2023, pp. 105–125.

[Int21] Intelligence, Y. “Neuromorphic Computing and Sensing 2021”. In: (2021).

[Izh01] Izhikevich, E. M. “Resonate-and-fire neurons”. In: *Neural networks* 14.6-7 (2001), pp. 883–894.

[Izh03] Izhikevich, E. M. “Simple model of spiking neurons”. In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.

[Izh+03] Izhikevich, E. M., Desai, N. S., Walcott, E. C., and Hoppensteadt, F. C. “Bursts as a unit of neural information: selective communication via resonance”. In: *Trends in neurosciences* 26.3 (2003), pp. 161–167.

[JYB16] Jalil, A., Yousaf, H., and Baig, M. I. “Analysis of CFAR techniques”. In: *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE. 2016, pp. 654–659.

[Jan18] Jankiraman, M. *FMCW radar design*. Artech House, 2018.

[Jim+16] Jiménez-Fernández, A., Cerezuela-Escudero, E., Miró-Amarante, L., Domínguez-Morales, M. J., Asís Gómez-Rodríguez, F. de, Linares-Barranco, A., and Jiménez-Moreno, G. “A binaural neuromorphic auditory sensor for FPGA: A spike signal processing approach”. In: *IEEE transactions on neural networks and learning systems* 28.4 (2016), pp. 804–818.

[Kai+22] Kaiser, K., Daugalas, J., López-Randulfe, J., Knoll, A., Weigel, R., and Lurz, F. “Complex-Valued Neural Networks for Millimeter Wave FMCW-Radar Angle Estimations”. In: *2022 19th European Radar Conference (EuRAD)*. IEEE. 2022, pp. 145–148.

[KPP17] Kamel, E. B., Peden, A., and Pajusco, P. “RCS modeling and measurements for automotive radar applications in the W band”. In: *2017 11th European Conference on Antennas and Propagation (EUCAP)*. IEEE. 2017, pp. 2445–2449.

[Kay+09] Kayser, C., Montemurro, M. A., Logothetis, N. K., and Panzeri, S. “Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns”. In: *Neuron* 61.4 (2009), pp. 597–608.

[Khe+18] Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., and Masquelier, T. “STDP-based spiking deep convolutional neural networks for object recognition”. In: *Neural Networks* 99 (2018), pp. 56–67.

[KM20] Kheradpisheh, S. R. and Masquelier, T. “Temporal backpropagation for spiking neural networks with one spike per neuron”. In: *International Journal of Neural Systems* 30.06 (2020), p. 2050027.

[Kim+18] Kim, J., Kim, H., Huh, S., Lee, J., and Choi, K. “Deep neural networks with weighted spikes”. In: *Neurocomputing* 311 (2018), pp. 373–386.

[KF16] Knight, J. C. and Furber, S. B. “Synapse-centric mapping of cortical models to the SpiNNaker neuromorphic architecture”. In: *Frontiers in neuroscience* 10 (2016), p. 213660.

[KGV14] Kösem, A., Gramfort, A., and Van Wassenhove, V. “Encoding of event timing in the phase of neural oscillations”. In: *Neuroimage* 92 (2014), pp. 274–284.

[KSH17] Krizhevsky, A., Sutskever, I., and Hinton, G. E. “Imagenet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[KS15] Kutty, S. and Sen, D. “Beamforming for millimeter wave communications: An inclusive survey”. In: *IEEE communications surveys & tutorials* 18.2 (2015), pp. 949–973.

[Lap07] Lapicque, L. “Recherches quantitatives sur l’excitation electrique des nerfs”. In: *J Physiol Paris* 9 (1907), pp. 620–635.

[Leh+23] Lehmann, H. M., Hille, J., Grassmann, C., and Issakov, V. “Direct Signal Encoding with Analog Resonate-and-Fire Neurons”. In: *IEEE Access* (2023).

[Lei+20] Leiserson, C. E., Thompson, N. C., Emer, J. S., Kuszmaul, B. C., Lampson, B. W., Sanchez, D., and Schardl, T. B. “There’s plenty of room at the Top: What will drive computer performance after Moore’s law?” In: *Science* 368.6495 (2020), eaam9744.

[LSW03] Li, J., Stoica, P., and Wang, Z. “On robust Capon beamforming and diagonal loading”. In: *IEEE transactions on signal processing* 51.7 (2003), pp. 1702–1715.

[LPD08] Lichtsteiner, P., Posch, C., and Delbruck, T. “A 128× 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor”. In: *IEEE journal of solid-state circuits* 43.2 (2008), pp. 566–576.

[Lin+18] Lin, S.-C., Zhang, Y., Hsu, C.-H., Skach, M., Haque, M. E., Tang, L., and Mars, J. “The architectural implications of autonomous driving: Constraints and acceleration”. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 2018, pp. 751–766.

[LI09] Livi, P. and Indiveri, G. “A current-mode conductance-based silicon neuron for address-event neuromorphic systems”. In: *2009 IEEE international symposium on circuits and systems*. IEEE. 2009, pp. 2898–2901.

[Lob+20] Lobo, J. L., Del Ser, J., Bifet, A., and Kasabov, N. “Spiking neural networks and online learning: An overview and perspectives”. In: *Neural Networks* 121 (2020), pp. 88–100.

[LRK22] Lopez-Randulfe, J., Reeb, N., and Knoll, A. “Conversion of ConvNets to Spiking Neural Networks With Less Than One Spike per Neuron”. In: *2022 Conference on Cognitive Computational Neuroscience*. 2022, pp. 553–555.

[LRK23] Lopez-Randulfe, J., Reeb, N., and Knoll, A. “Integrate-and-fire circuit for converting analog signals to spikes using phase encoding”. In: *Neuromorphic Computing and Engineering* 3.4 (2023), p. 044002.

[Lóp+21] López-Randulfe, J., Duswald, T., Bing, Z., and Knoll, A. “Spiking neural network for fourier transform and object detection for automotive radar”. In: *Frontiers in Neurorobotics* 15 (2021), p. 69.

[Lóp+22] López-Randulfe, J., Reeb, N., Karimi, N., Liu, C., Gonzalez, H., Dietrich, R., Vogginger, B., Mayr, C., and Knoll, A. “Time-Coded Spiking Fourier Transform in Neuromorphic Hardware”. In: *IEEE Transactions on Computers* (2022).

[Maa97] Maass, W. “Networks of spiking neurons: the third generation of neural network models”. In: *Neural networks* 10.9 (1997), pp. 1659–1671.

[Mar+20] Marković, D., Mizrahi, A., Querlioz, D., and Grollier, J. “Physics for neuromorphic computing”. In: *Nature Reviews Physics* 2.9 (2020), pp. 499–510.

[Mar+18] Martin, J. G., Davis, C. E., Riesenhuber, M., and Thorpe, S. J. “Zapping 500 faces in less than 100 seconds: evidence for extremely fast and sustained continuous visual search”. In: *Scientific reports* 8.1 (2018), pp. 1–12.

[MEL03] Mathie, A., E. Kennard, L., and L. Veale, E. “Neuronal ion channels and their sensitivity to extremely low frequency weak electric field effects”. In: *Radiation Protection Dosimetry* 106.4 (2003), pp. 311–315.

[MHF19] Mayr, C., Hoeppner, S., and Furber, S. “Spinnaker 2: A 10 million core processor system for brain simulation and machine learning”. In: *arXiv preprint arXiv:1911.02385* (2019).

[McK10] McKeown, M. “FFT Implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs”. In: *Texas Instruments Incorporated, White Paper SPRABB6B* (2010).

[Mea90] Mead, C. “Neuromorphic electronic systems”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636.

[Mea20] Mead, C. “How we created neuromorphic engineering”. In: *Nature Electronics* 3.7 (2020), pp. 434–435.

[MM88] Mead, C. A. and Mahowald, M. A. “A silicon model of early visual processing”. In: *Neural networks* 1.1 (1988), pp. 91–97.

[Mer+14] Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N., Guo, C., Nakamura, Y., et al. “A million spiking-neuron integrated circuit with a scalable communication network and interface”. In: *Science* 345.6197 (2014), pp. 668–673.

[Mod+23] Modha, D. S., Akopyan, F., Andreopoulos, A., Appuswamy, R., Arthur, J. V., Cassidy, A. S., Datta, P., DeBole, M. V., Esser, S. K., Otero, C. O., et al. “Neural inference at the frontier of energy, space, and time”. In: *Science* 382.6668 (2023), pp. 329–335.

[Mor+17] Moradi, S., Qiao, N., Stefanini, F., and Indiveri, G. “A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)”. In: *IEEE transactions on biomedical circuits and systems* 12.1 (2017), pp. 106–122.

[ML81] Morris, C. and Lecar, H. “Voltage oscillations in the barnacle giant muscle fiber”. In: *Biophysical journal* 35.1 (1981), pp. 193–213.

[MS03] Moss, C. F. and Sinha, S. R. “Neurobiology of echolocation in bats”. In: *Current opinion in Neurobiology* 13.6 (2003), pp. 751–758.

[Mos17] Mostafa, H. “Supervised learning based on temporal coding in spiking neural networks”. In: *IEEE transactions on neural networks and learning systems* 29.7 (2017), pp. 3227–3235.

[Mur13] Murmann, B. “Energy limits in A/D converters”. In: *2013 IEEE Faible Tension Faible Consommation* (2013), pp. 1–4.

[Mur15] Murmann, B. “The race for the extra decibel: A brief review of current ADC performance trajectories”. In: *IEEE Solid-State Circuits Magazine* 7.3 (2015), pp. 58–66.

[Nec+18] Neckar, A., Fok, S., Benjamin, B. V., Stewart, T. C., Oza, N. N., Voelker, A. R., Eliasmith, C., Manohar, R., and Boahen, K. “Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model”. In: *Proceedings of the IEEE* 107.1 (2018), pp. 144–164.

[Orc+21] Orchard, G., Frady, E. P., Rubin, D. B. D., Sanborn, S., Shrestha, S. B., Sommer, F. T., and Davies, M. “Efficient neuromorphic signal processing with loihi 2”. In: *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE. 2021, pp. 254–259.

[Oua+21] Ouaknine, A., Newson, A., Rebut, J., Tupin, F., and Pérez, P. “Carrada dataset: Camera and automotive radar with range-angle-doppler annotations”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 5068–5075.

[PAR20] Panda, P., Aketi, S. A., and Roy, K. “Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization”. In: *Frontiers in Neuroscience* 14 (2020), p. 653.

[Pas+12] Pasley, B. N., David, S. V., Mesgarani, N., Flinker, A., Shamma, S. A., Crone, N. E., Knight, R. T., and Chang, E. F. “Reconstructing speech from human auditory cortex”. In: *PLoS biology* 10.1 (2012), e1001251.

[Pat+17] Patole, S. M., Torlak, M., Wang, D., and Ali, M. “Automotive radars: A review of signal processing techniques”. In: *IEEE Signal Processing Magazine* 34.2 (2017), pp. 22–35.

[PKK19] Petro, B., Kasabov, N., and Kiss, R. M. “Selection and optimization of temporal spike encoding methods for spiking neural networks”. In: *IEEE transactions on neural networks and learning systems* 31.2 (2019), pp. 358–370.

[Pfe+13] Pfeil, T., Scherzer, A.-C., Schemmel, J., and Meier, K. “Neuromorphic learning towards nano second precision”. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2013, pp. 1–5.

[PU21] Pitcher, D. and Ungerleider, L. G. “Evidence for a third visual pathway specialized for social perception”. In: *Trends in Cognitive Sciences* 25.2 (2021), pp. 100–110.

[Qia+15] Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., and Indiveri, G. “A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses”. In: *Frontiers in neuroscience* 9 (2015), p. 141.

[Rad+18] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. “Improving language understanding by generative pre-training”. In: (2018).

[RDP18] Rameez, M., Dahl, M., and Pettersson, M. I. “Adaptive digital beamforming for interference suppression in automotive FMCW radars”. In: *2018 IEEE Radar Conference (RadarConf18)*. IEEE. 2018, pp. 0252–0256.

[Ram04] Ramón y Cajal, S. *Textura del sistema nervioso del hombre y de los vertebrados: estudios sobre el plan estructural y composición histológica de los centros nerviosos adicionados de consideraciones fisiológicas fundadas en los nuevos descubrimientos*. Vol. 2. N. Moya, 1904.

[Rat+13] Ratté, S., Hong, S., De Schutter, E., and Prescott, S. A. “Impact of neuronal properties on network coding: roles of spike initiation dynamics and robust synchrony transfer”. In: *Neuron* 78.5 (2013), pp. 758–772.

[Ree+25] Reeb, N., Lopez-Randulfe, J., Dietrich, R., and Knoll, A. C. “Range and Angle Estimation with Spiking Neural Resonators for FMCW Radar”. In: *arXiv preprint arXiv:2503.00898* (2025).

[Rei+00] Reich, D. S., Mechler, F., Purpura, K. P., and Victor, J. D. “Interspike Intervals, Receptive Fields, and Information Encoding in Primary Visual Cortex”. In: *The Journal of Neuroscience* 20.5 (2000), pp. 1964–1974. ISSN: 0270-6474. DOI: 10.1523/jneurosci.20-05-01964.2000.

[RG19] Reverter, F. and Gasulla, M. “Experimental characterization of the energy consumption of ADC embedded into microcontrollers operating in low power”. In: *2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE. 2019, pp. 1–5.

[Rey+19] Reyes, B. T., Biolato, L., Galetto, A. C., Passetti, L., Solis, F., and Hueda, M. R. “An energy-efficient hierarchical architecture for time-interleaved SAR ADC”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66.6 (2019), pp. 2064–2076.

[Rho+18] Rhodes, O., Bogdan, P. A., Brenninkmeijer, C., Davidson, S., Fellows, D., Gait, A., Lester, D. R., Mikaitis, M., Plana, L. A., Rowley, A. G., et al. “sPyNNaker: a software package for running PyNN simulations on SpiNNaker”. In: *Frontiers in neuroscience* 12 (2018), p. 816.

[Rho+20] Rhodes, O., Peres, L., Rowley, A. G., Gait, A., Plana, L. A., Brenninkmeijer, C., and Furber, S. B. “Real-time cortical simulation on neuromorphic hardware”. In: *Philosophical Transactions of the Royal Society A* 378.2164 (2020), p. 20190160.

[Rie+99] Rieke, F., Warland, D., Van Steveninck, R. d. R., and Bialek, W. *Spikes: exploring the neural code*. MIT press, 1999.

[Rob+20] Robertson, J., Hejda, M., Bueno, J., and Hurtado, A. “Ultrafast optical integration and pattern classification for neuromorphic photonics based on spiking VCSEL neurons”. In: *Scientific reports* 10.1 (2020), p. 6098.

[Roh83] Rohling, H. “Radar CFAR Thresholding in Clutter and Multiple Target Situations”. In: *IEEE Transactions on Aerospace and Electronic Systems* AES-19.4 (1983), pp. 608–621. DOI: 10.1109/TAES.1983.309350.

[Rot+22] Roth, F., Bidoul, N., Rosca, T., Dörpinghaus, M., Flandre, D., Ionescu, A. M., and Fettweis, G. “Spike-based sensing and communication for highly energy-efficient sensor edge nodes”. In: *2022 2nd IEEE International Symposium on Joint Communications & Sensing (JC&S)*. IEEE. 2022, pp. 1–6.

[RJP19] Roy, K., Jaiswal, A., and Panda, P. “Towards spike-based machine intelligence with neuromorphic computing”. In: *Nature* 575.7784 (2019), pp. 607–617.

[RL18] Rueckauer, B. and Liu, S.-C. “Conversion of analog to spiking neural networks using sparse temporal coding”. In: *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2018, pp. 1–5.

[Rue+17] Rueckauer, B., Lungu, I.-A., Hu, Y., Pfeiffer, M., and Liu, S.-C. “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification”. In: *Frontiers in neuroscience* 11 (2017), p. 682.

[Sar97] Sarwar, A. “Cmos power consumption and cpd calculation”. In: *Proceeding: Design Considerations for Logic Products* (1997).

[SH21] Schmidgall, S. and Hays, J. “Stable Lifelong Learning: Spiking neurons as a solution to instability in plastic neural networks”. In: *arXiv preprint arXiv:2111.04113* (2021).

[SH22] Schmidgall, S. and Hays, J. “Stable lifelong learning: Spiking neurons as a solution to instability in plastic neural networks”. In: *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*. 2022, pp. 1–7.

[Sch22] Schmidhuber, J. “Annotated History of Modern AI and Deep Learning”. In: *arXiv preprint arXiv:2212.11279* (2022).

[Sch+17] Schmitt, S., Klähn, J., Bellec, G., Grübl, A., Guettler, M., Hartel, A., Hartmann, S., Husmann, D., Husmann, K., Jeltsch, S., et al. “Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system”. In: *2017 international joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 2227–2234.

[Sch+22] Schuman, C. D., Kulkarni, S. R., Parsa, M., Mitchell, J. P., Kay, B., et al. “Opportunities for neuromorphic computing algorithms and applications”. In: *Nature Computational Science* 2.1 (2022), pp. 10–19.

[Sen+19] Sengupta, A., Ye, Y., Wang, R., and Roy, K. “Going deeper in spiking neural networks: VGG and residual architectures”. In: *Frontiers in neuroscience* 13 (2019), p. 425055.

[Sha+21a] Sharifshazileh, M., Burelo, K., Sarnthein, J., and Indiveri, G. “An electronic neuromorphic system for real-time detection of high frequency oscillations (HFO) in intracranial EEG”. In: *Nature communications* 12.1 (2021), p. 3095.

[Sha+21b] Shastri, B. J., Tait, A. N., Ferreira de Lima, T., Pernice, W. H., Bhaskaran, H., Wright, C. D., and Prucnal, P. R. “Photonics for artificial intelligence and neuromorphic computing”. In: *Nature Photonics* 15.2 (2021), pp. 102–114.

[Sha+20] Shawkat, M. S. A., Sayyarparaju, S., McFarlane, N., and Rose, G. S. “Single photon avalanche diode based vision sensor with on-chip memristive spiking neuromorphic processing”. In: *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE. 2020, pp. 377–380.

[SO18] Shrestha, S. B. and Orchard, G. “Slayer: Spike layer error reassignment in time”. In: *Advances in neural information processing systems* 31 (2018).

[SIA09] Sifuzzaman, M., Islam, M. R., and Ali, M. Z. “Application of wavelet transform and its advantages compared to Fourier transform”. In: *Vidyasagar University Journal of Physical Sciences* 13 (2009), pp. 121–134.

[Sou+17] Sourikopoulos, I., Hedayat, S., Loyez, C., Danneville, F., Hoel, V., Mercier, E., and Cappy, A. “A 4-fJ/spike artificial neuron in 65 nm CMOS technology”. In: *Frontiers in neuroscience* 11 (2017), p. 247370.

[Sta+20] Stagsted, R., Vitale, A., Binz, J., Bonde Larsen, L., Sandamirskaya, Y., et al. "Towards neuromorphic control: A spiking neural network based PID controller for UAV". In: RSS. 2020.

[Sta13] Stanley, G. B. "Reading and writing the neural code". In: *Nature neuroscience* 16.3 (2013), pp. 259–263.

[SM21] Stöckl, C. and Maass, W. "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes". In: *Nature Machine Intelligence* 3.3 (2021), pp. 230–238.

[Stu+21] Stuijt, J., Sifalakis, M., Yousefzadeh, A., and Corradi, F. "μBrain: An event-driven and fully synthesizable architecture for spiking neural networks". In: *Frontiers in neuroscience* 15 (2021), p. 664208.

[Tan+13] Tang, W., Osman, A., Kim, D., Goldstein, B., Huang, C., Martini, B., Pieribone, V. A., and Culurciello, E. "Continuous time level crossing sampling ADC for bio-potential recording systems". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 60.6 (2013), pp. 1407–1418.

[Tha+18] Thakur, C. S., Molin, J. L., Cauwenberghs, G., Indiveri, G., Kumar, K., Qiao, N., Schemmel, J., Wang, R., Chicca, E., Olson Hasler, J., et al. "Large-scale neuromorphic spiking array processors: A quest to mimic the brain". In: *Frontiers in neuroscience* 12 (2018), p. 891.

[TG98] Thorpe, S. and Gautrais, J. "Rank order coding". In: *Computational Neuroscience: Trends in Research, 1998*. Springer, 1998, pp. 113–118.

[Tim+23] Timcheck, J., Shrestha, S. B., Rubin, D. B. D., Kupryjanow, A., Orchard, G., Pindor, L., Shea, T., and Davies, M. "The intel neuromorphic DNS challenge". In: *Neuromorphic Computing and Engineering* 3.3 (2023), p. 034005.

[Tor+17] Torrejon, J., Riou, M., Araujo, F. A., Tsunegi, S., Khalsa, G., Querlioz, D., Bortolotti, P., Cros, V., Yakushiji, K., Fukushima, A., et al. "Neuromorphic computing with nanoscale spintronic oscillators". In: *Nature* 547.7664 (2017), pp. 428–431.

[TC98] Torrence, C. and Compo, G. P. "A practical guide to wavelet analysis". In: *Bulletin of the American Meteorological society* 79.1 (1998), pp. 61–78.

[Tsa+21] Tsang, I. J., Corradi, F., Sifalakis, M., Van Leekwijck, W., and Latre, S. "Radar-based hand gesture recognition using spiking neural networks". In: *Electronics* 10.12 (2021), p. 1405.

[Tur50] Turing, A. "Machines informatiques et intelligence". In: *Mind* 49 (1950), pp. 433–460.

[Van+18] Van Albada, S. J., Rowley, A. G., Senk, J., Hopkins, M., Schmidt, M., Stokes, A. B., Lester, D. R., Diesmann, M., and Furber, S. B. "Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model". In: *Frontiers in neuroscience* 12 (2018), p. 309524.

[Van01] Van Schaik, A. "Building blocks for electronic spiking neural networks". In: *Neural networks* 14.6-7 (2001), pp. 617–628.

[VB88] Van Veen, B. D. and Buckley, K. M. "Beamforming: A versatile approach to spatial filtering". In: *IEEE assp magazine* 5.2 (1988), pp. 4–24.

[Vas+17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[Via+21] Viale, A., Marchisio, A., Martina, M., Masera, G., and Shafique, M. “Carsnn: An efficient spiking neural network for event-based autonomous cars on the loihi neuromorphic research processor”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–10.

[Vog+22] Vogginger, B., Kreutz, F., López-Randulfe, J., Liu, C., Dietrich, R., Gonzalez, H. A., Scholz, D., Reeb, N., Auge, D., Hille, J., et al. “Automotive Radar Processing With Spiking Neural Networks: Concepts and Challenges”. In: *Frontiers in neuroscience* 16 (2022).

[Vog+24] Vogginger, B., Rostami, A., Jain, V., Arfa, S., Hantsch, A., Kappel, D., Schäfer, M., Faltings, U., Gonzalez, H. A., Liu, C., et al. “Neuromorphic hardware for sustainable AI data centers”. In: *arXiv preprint arXiv:2402.02521* (2024).

[WHM21] Waldschmidt, C., Hasch, J., and Menzel, W. “Automotive radar—From first efforts to future systems”. In: *IEEE Journal of Microwaves* 1.1 (2021), pp. 135–148.

[Wan+23] Wang, D., Tang, R., Lin, H., Liu, L., Xu, N., Sun, Y., Zhao, X., Wang, Z., Wang, D., Mai, Z., et al. “Spintronic leaky-integrate-fire spiking neurons with self-reset and winner-takes-all for neuromorphic computing”. In: *Nature Communications* 14.1 (2023), p. 1068.

[Wan+20] Wang, W., Zhou, S., Li, J., Li, X., Yuan, J., and Jin, Z. “Temporal pulses driven spiking neural network for fast object recognition in autonomous driving”. In: *arXiv preprint arXiv:2001.09220* (2020).

[WD08] Wijekoon, J. H. and Dudek, P. “Compact silicon neuron circuit with spiking and bursting behaviour”. In: *Neural Networks* 21.2-3 (2008), pp. 524–534.

[Win+14] Winner, H., Hakuli, S., Lotz, F., and Singer, C. *Handbook of driver assistance systems*. Springer International Publishing, 2014.

[Yan+21] Yan, Y., Stewart, T. C., Choo, X., Vogginger, B., Partzsch, J., Höppner, S., Kelber, F., Eliasmith, C., Furber, S., and Mayr, C. “Comparing Loihi with a SpiNNaker 2 prototype on low-latency keyword spotting and adaptive robotic control”. In: *Neuromorphic Computing and Engineering* 1.1 (2021), p. 014002.

[ZD19] Zbili, M. and Debanne, D. “Past and future of analog-digital modulation of synaptic transmission”. In: *Frontiers in cellular neuroscience* 13 (2019), p. 160.

[Zha+17] Zhao, C., Yi, Y., Li, J., Fu, X., and Liu, L. “Interspike-interval-based analog spike-time-dependent encoder for neuromorphic processors”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.8 (2017), pp. 2193–2205.

[Zhe+23] Zheng, K., Qian, K., Woodford, T., and Zhang, X. “NeuroRadar: A Neuromorphic Radar Sensor for Low-Power IoT Systems”. In: (2023).

A

Hebbian learning

For a pre-synaptic neuron and a post-synaptic neuron with firing rates u_b and v, respectively, we can model the firing dynamics of the post-synaptic neuron as

$$v = \sum_{b=1}^{N_u} w_b \cdot u_b \,, \tag{A.1}$$

where N_u is the amount of pre-synaptic neurons, and w_b is the synaptic weight between the bth pre-synaptic neuron and the post-synaptic neuron.

The simplest Hebb rule takes the form of

$$\tau_w \frac{d\boldsymbol{w}}{dt} = v\boldsymbol{u} \,, \tag{A.2}$$

where τ_w is a time constant for regulating the rate of change of the synaptic weights.

We can replace v in (A.2) by using (A.1), and obtain the so called *correlation-based plasticity rule*

$$\tau_w \frac{d\boldsymbol{w}}{dt} = Q\boldsymbol{w} \,, \tag{A.3}$$

where Q is the input correlation matrix. Equation eq. (A.3) only works if the weights matrix is initialized to a non-zero matrix. If we average over several input patterns, Q is calculated as

$$Q = \langle \boldsymbol{u}, \boldsymbol{u} \rangle \,. \tag{A.4}$$

The previous form of the Hebb rule introduces LTP. To obtain LTD, we introduce a threshold term in the equation that can either be applied to $\boldsymbol{u}$ or to v. In the first case, (A.2) takes the form of

$$\tau_w \frac{d\boldsymbol{w}}{dt} = v(\boldsymbol{u} - \boldsymbol{\theta_u}), \tag{A.5}$$

where $\boldsymbol{\theta_u}$ is the threshold applied to the pre-synaptic firing rate. A suitable choice for $\boldsymbol{\theta_u}$ is the average firing rate of $\boldsymbol{u}$. If this is the case, the equation takes the form of

$$\tau_w \frac{d\boldsymbol{w}}{dt} = vC \,, \tag{A.6}$$

where C is the covariance matrix of the input. Therefore, (A.6) is called *covariance rule* and C can be calculated as

$$C = \langle (\boldsymbol{u} - \langle \boldsymbol{u} \rangle)(\boldsymbol{u} - \langle \boldsymbol{u} \rangle) \rangle \,. \tag{A.7}$$

A.1 BCM rule

It stands for Bienenstock, Cooper, and Munro, the authors that in 1982 proposed this synaptic rule.

$$\tau_w \frac{d\boldsymbol{w}}{dt} = v\boldsymbol{u}(v - \theta_v). \tag{A.8}$$

The main difference with the covariance rule (A.6) is that the term θ_v is not fixed. By allowing the threshold to vary with certain constraints, the equation becomes stable and does not grow to infinity. The critical condition for reaching stability is that the threshold grows faster than v. One option is as follows,

$$\tau_\theta \frac{d\theta_v}{dt} = v^2 - \theta_v, \tag{A.9}$$

where τ_θ is the time constant of the rate of change of θ_v. This value is typically bigger than τ_w. (A.9) is a version of the BCM rule that uses a sliding window. Besides being stable, this rule also introduces competition between the different pre-synaptic variables i.e. when weights of specific inputs increase, the post-synaptic neuron increases its firing rate, and therefore increases the threshold θ_v, reducing the increase of the weights from the other pre-synaptic neurons.

A.2 Synaptic normalization and Oja rule

A different alternative for achieving a stable Hebb rule is to normalize the synaptic weights i.e. bound the weights values. If the weights can only take positive values, the normalization is typically implemented by limited the total weights sum. In case they can take negative values as well, the squared sum is a more adequate option. Moreover, the normalization may be applied all the time, or dynamically at the end of the learning process.

One approach for implementing synaptic normalization is to by subtraction. Namely, the weights evolution is constrained by the sum of all the input weights. It can represented as

$$\tau_w \frac{d\boldsymbol{w}}{dt} = v\boldsymbol{u} - \frac{v(\boldsymbol{n}\boldsymbol{u})\boldsymbol{n}}{N_u}, \tag{A.10}$$

where N_u is the total amount of pre-synaptic neurons, and $\boldsymbol{n}$ is a unit vector of size N_u. This rule subtracts from every weight the same amount, based on the total sum of all weights. Therefore, it is non-local and therefore biologically non-plausible.

Oja rule is known as a multiplicative normalization rule, since it imposes a constraint on the sum of the squares of the weights. This rule can be expressed as

$$\tau_w \frac{d\boldsymbol{w}}{dt} = v\boldsymbol{u} - \alpha v^2 \boldsymbol{w}, \tag{A.11}$$

where α is a positive constant. Contrary to (A.10) , Oja rule only uses local synaptic information.

A.3 Spiking timing-dependent plasticity (STDP)

The usage of specific spike times for adapting the weights of the synapses is perhaps the most popular version of Hebbian learning. This type of learning is called STDP, and many works have recently adapted this rule for different applications and scopes [SH21; Khe+18].

B

Leaky integrate-and-fire differential equation

This appendix introduces two versions of the leaky integrate-and-fire (LIF) neuron model, one driven by an input current and another one driven by a voltage source. fig. B.1 contain the diagrams of the equivalent circuits for both models. The following sections describe how to reach the differential equations for both models, and solve them for the specific case of a constant input signal during the integration period.

B.1 Current-driven LIF neuron

The LIF neuron is typically represented by an electric circuit with an input current $I(t)$ that feeds two parallel branches, one containing the capacitance C and the second formed by a resistance R. Moreover, a voltage source represents the resting potential u_{rest}. fig. B.1a depicts a diagram for the equivalent electric circuit. The total input current is split according to

$$I(t) = I_C + I_R\,, \tag{B.1}$$

where I_C and I_R are the currents circulating through C and R, respectively. The current I_C is solved by applying the general differential equation of a capacitor,

$$I_C(t) = \frac{du_C}{dt}C\,, \tag{B.2}$$

where C and u_C are the capacitor's capacitance and potential difference, respectively. The current I_R is calculated by using Ohm's law,

$$I_R = \frac{u - u_{\text{rest}}}{R}\,, \tag{B.3}$$

where u and u_{rest} are the membrane and resting potentials, respectively. By using the equality $u = u_C + u_{\text{rest}}$ and combining the three previous equations, we obtain the ordinary differential equation

$$\frac{du_C}{dt}RC = RI(t) - u_C\,. \tag{B.4}$$

From now on, we will denote the product RC as the membrane time constant τ_m. We rearrange (B.4) to separate the variables t and u_C,

$$\frac{du_C}{RI(t) - u_C} = \frac{dt}{\tau_m}\,. \tag{B.5}$$

B.2 Solving for a constant input current I_0

We solve now (B.5) for the case of a constant input current $I(t) = I_0$ and assuming $u(0) = u_{\text{rest}}$. We integrate both sides of the equation

$$\int \frac{du}{I_0 R - u_C} = \frac{1}{\tau_m} \int dt\,. \tag{B.6}$$

Integrating for $t \in [0, t]$ and $u_C \in [0, u_{C,t}]$ yields

$$-\ln\left(\frac{I_0 R - u_{C,t}}{I_0 R}\right) = \frac{t}{\tau_m} + k_1\,. \tag{B.7}$$

With the contour condition $u_{C,t} = 0$ if $t = 0$, then $k_1 = 0$. Applying the transformation $u_c = u - u_{\text{rest}}$, we obtain

$$-\ln\left(\frac{I_0 R + u_{\text{rest}} - u}{I_0 R}\right) = -\frac{t}{\tau_m}\,. \tag{B.8}$$

Finally, we solve the ODE by applying exponentials for removing the logarithms, that yields

$$u = u_{\text{rest}} + I_0 R(1 - e^{t/\tau_m})\,. \tag{B.9}$$

B.3 Voltage-driven LIF neuron

The previous section describes a neuron model that assumes the information provided to the neuron arrives in the form of a time-varying current. Alternatively, we can model the LIF neuron as a circuit that receives information in the form of a time-varying voltage signal. We do that by placing the resistor and the capacitor in series, so the membrane potential is represented by the voltage of the node between the two components,

$$u_{\text{in}}(t) = u_C + u_R + u_{\text{rest}}\,, \tag{B.10}$$

where u_C and u_R is the potential difference in the capacitor and the resistor, respectively, and we can obtain their values using (B.2) and (B.3), respectively. fig. B.1b depicts a diagram of the equivalent electric circuit As the current passing through the capacitor and the resistor is the same, $I_C = I_R$, we can turn (B.10) in

$$\frac{du_C}{dt} C = \frac{u_{\text{in}}(t) - u_{\text{rest}} - u_C}{R}\,. \tag{B.11}$$

Same as for the current-based model, we can replace $\tau_m = RC$ and separate the variables t and u,

$$\frac{du_C}{u_{\text{in}}(t) - u_{\text{rest}} - u_C} = \frac{dt}{\tau_m}\,. \tag{B.12}$$

B.4 Solving for a constant input voltage U_0

We can simplify the ODE in section B.3 by assuming that the input voltage stays constant for the integration time, $u_{\text{in}}(t) = U_0$. This is a reasonable assumption if the neuron dynamics are faster than the input voltage dynamics. The integration of both sides of (B.12) yields

$$\int \frac{du}{U_0 - u_{\text{rest}} - u_C} = \frac{1}{\tau_m} \int dt\,. \tag{B.13}$$

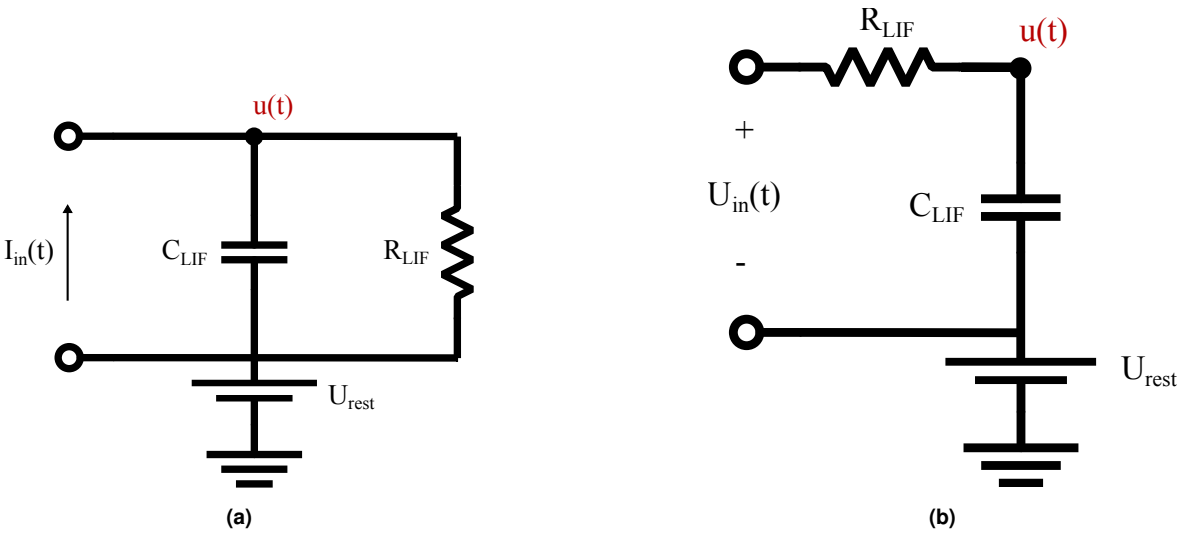


Figure B.1: Equivalent electric diagrams for the leaky integrate-and-fire neuron model. (a) represents a current-based LIF, and (b) represents a voltage-based LIF.

Integrating for $t \in [0, t]$ and $u_C \in [0, u_{C,t}]$ yields

$$-\ln\left(\frac{U_0 - u_{\text{rest}} - u_{C,t}}{U_0 - u_{\text{rest}}}\right) = \frac{t}{\tau_m} + k_2\,. \tag{B.14}$$

If we assume $u_{C,t} = 0$ for $t = 0$, then $k_2 = 0$. Applying the transformation $u_C = u - u_{\text{rest}}$, we can then solve the ODE as

$$u(t) = U_0(1 - e^{-t/\tau_m}) + u_{\text{rest}} e^{-t/\tau_m}\,. \tag{B.15}$$

C

Implementation of analogue-to-spike encoder

A prototype of the analogue-to-spike encoder introduced in 5 was implemented for validating the properties of the circuit. section 6.2 contains a summary of the experiment results and their corresponding discussion. This section covers details of the implementation that are necessary for replicating the experiments, as well as the formal proof for the choice of certain components.

Table C.1 contains a list of the specific components that were used for implementing the circuit, together with their corresponding reference. The choice of the components was made based on the availability in the market and suitability for easy and fast deployment on a prototype board. All active components of the circuit were protected with decoupling capacitors of value $C_{dec} = 100nF$. The table does not include auxiliary components necessary for implementing the circuit, such as wires, pin headers, or bread-boards. The experiments also required external equipment for generating the input signals, namely a DC voltage supply and a signal generator.

fig. C.1 Shows the ASE prototype on the bottom, connected to a function generator that provided sinusoidal signals as input. The experiments employed signals with frequencies in the range [25Hz, 1kHz]. The oscilloscope on top validated the frequencies of the input signal. A UART connection between the microcontroller and a PC allowed to store the spike times after the experiments.

Table C.1: List of the commercial components used for implementing the ASE described in 5, together with their magnitude, reference, retailer, and retailer reference.

Component	Mag.	Reference	Retailer	Ret. ref.
C_{LIF}	$100nF$	RS 181-4752	RS	263-3034
R_{LIF}	$30k\Omega$	LR1F30K	RS	683-5580
R_{div}	$10k\Omega$	RK09K1130A0H	RS	263-3034
$R_{pull-up}$	$30k\Omega$	LR1F30K	RS	148-843
C_{dec}	$100nF$	RS 181-4752	RS	181-4752
Comparator	-	LP339N	Mouser	595-LP339N
D flip-flop	-	SN74HC74NE4	Mouser	595-SN74HC74NE4
Analog switch	-	MAX317CPA	Mouser	700-MAX317CPA
Microc. board	-	EV88G73A	Mouser	579-EV88G73A

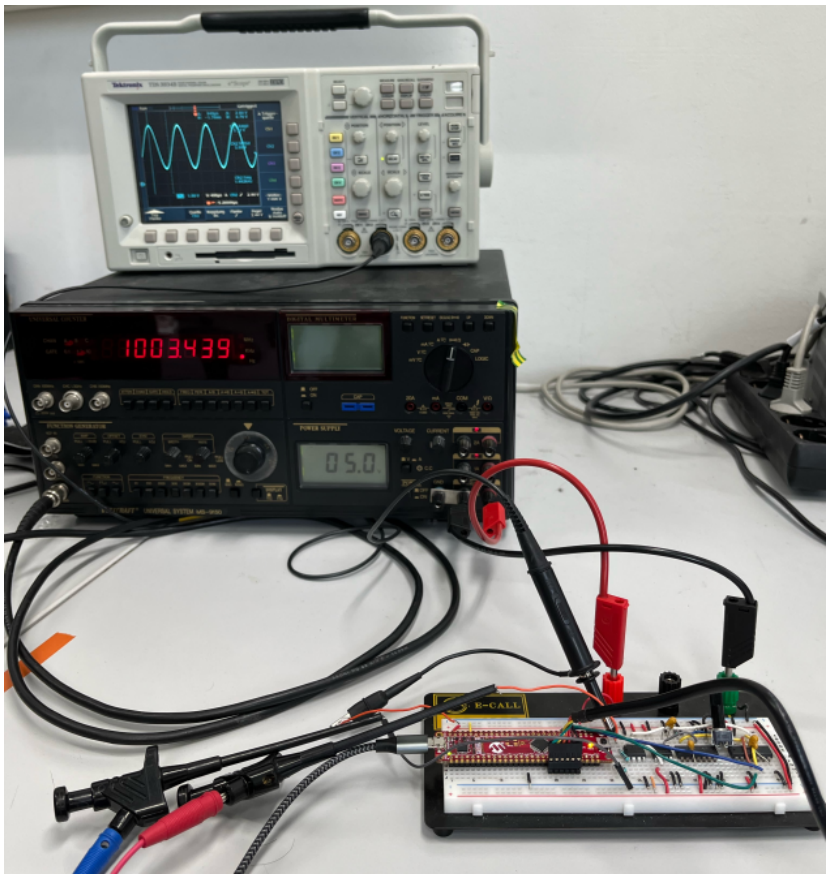


Figure C.1: Lab setup for sampling sinusoidal wave signals with the ASE. On top, an oscilloscope for validating the input signals. In the middle, the function generator that provided the wave signals. At the bottom, the ASE prototype board.

C.1 LIF temporal dynamics

Equation (5.1) yields the time constant τ_m of the ASE by fixing $u(t) = u_{\min}$ and $t = t_{\max}$ to the limits imposed by the electronic components. For the values of C_{LIF} and R_{LIF} indicated in table C.1, $\tau_m = 100\text{nF} \times 30k\Omega = 3$ ms. The spike time of the ASE is obtained with (5.3).

C.2 Discharge resistor for C_{LIF} switch

When the membrane voltage of the encoder reaches the threshold voltage u_{th}, the circuit enters a reset phase. This includes the activation of the switch that discharges the encoder capacitor C_{LIF}. In order to avoid the destruction of the two components involved (the capacitor and the switch), it is necessary to include a resistor that absorbs the energy of the capacitor. On the other hand, the discharge resistance needs to be as low as possible, in order to minimize the discharge time for the capacitor.

When the switch activates, the discharge takes place through a simple $R-C$ circuit with an initial voltage V_{th}. During discharge time, the circuit follows the $R-C$ dynamics

$$C\frac{du}{dt} = \frac{u(t)}{R_S}, \tag{C.1}$$

where $C = C_{LIF}$ is the capacitance of the capacitor, and R_S is the value of the discharge

resistor. The initial condition for the membrane voltage is $u_0 = u_{th}$, and the voltage at time t is obtained by solving (C.1), which yields

$$u(t) = e^{t/\tau} u_0 \,, \tag{C.2}$$

where $\tau = R_S C_{LIF}$ is the time constant of the capacitor. The instantaneous current circulating through R_S is given by Ohm's law,

$$I(t) = \frac{u(t)}{R_S} \,. \tag{C.3}$$

The peak current through the circuit takes place at $t = t_0$, when the voltage takes the maximum value $u_0 = u_{th}$. For the chosen analog switch, MAX317, R_L is given by its ON resistance, which is $R_{ON} = 20\Omega$. In absence of additional resistors, and using a security coefficient $\alpha = 2$, the peak current would thus be $I_{MAX} = \alpha \cdot u_{th}/R_{ON} = 2 \cdot 0.5/20 = 50\text{mA}$. This value lays below the peak current $I_{peak} = 100\text{mA}$ that the switch can tolerate. Therefore, the design does not need to include an additional protection resistor.

C.3 Interface with digital acquisition of spikes

The spikes generated by the circuit need to be collected and stored so they can be processed by a neuromorphic chip. In the prototype implementation a microcontroller was used for this purpose, namely the EV88G73A developer board from Microchip.

The main functionalities of the microcontroller are the following:

1. Generate the sampling clock signal CLK_S at a fixed rate
2. Listen to the SPK digital input and generate an interrupt when a HI level is detected
3. Record the time when SPK takes place
4. Send the reset flag CLK_N to the circuit when $SPK = HI$
5. Send the collected spike times to the main controller of the experiment for a posterior analysis of the data

D

Conversion with the TCBS neuron model

The *temporal charge before spike* (TCBS) neuron model described in chapter 4 can be applied to the conversion of *artificial neural network*s (ANNs) to time-coded *spiking neural network*s (SNNs) for the inference stage. The neuron model's temporal nature and linear behaviour permit an easy conversion of crucial operations like max pooling or the ReLU activation function. A LeNet *deep convolutional neural network* (DCNN) was trained for digit classification with the MNIST dataset to test the model. Specifically, the network comprised two convolutional layers with 32 kernels each, and two dense layers with 100 and 10 neurons, respectively (see fig. D.1). The network was trained using a dropout of 0.2 and applying data augmentation to the dataset using rotations, translations, and shifts to the original images.

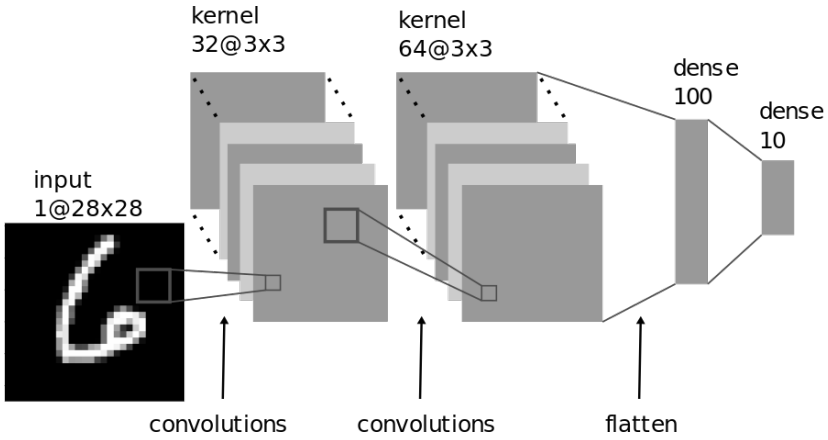


Figure D.1: LeNet architecture used for classifying digits from the MNIST dataset. After training, the network was converted to an SNN using the TCBS neuron model

The converted SNN was simulated with input data converted to time-coded spikes, according to the *time to first spike* (TTFS) encoding defined by (4.3). All neurons on each layer were run in parallel with the same times for their silent and spiking stages, which were referenced to the start of the silent stage, i.e., $t_{\min} = t_0 = 0$. Moreover, all neurons run for the same number of time steps $t_{\max}$, which was a tunable parameter. A second parameter was the membrane voltage u_{th}. Even though it is possible to calculate a theoretical maximum value for u_{th} based on the synaptic weights of each neuron, such value is never reached when working with actual data, as it is randomly distributed across the input spectrum and the addition of all inputs is far from the theoretical maximum. Thus, experiments assessed the impact of both $t_{\max}$ and u_{th} on the SNN error. fig. D.2 depicts the *root-mean-square error* (RMSE) of all neurons in the SNN when compared to the real number they should represent.

Table D.1: Comparison of the inference results for a LeNet DCNN architecture for the MNIST dataset with the results of the same network converted to SNN using the TCBS model.

	FLOPs	synaptic ops.	Acc.
ANN	14316496	-	99.56
SNN	-	1015015	99.44

table D.1 compares the original ANN with the converted network for the optimal parameters from fig. D.2. We can observe that the accuracy after conversion drops 0.12%, while the number of synaptic operations is considerably smaller than that of floating point operations in the ANN. This is due to two reasons: Firstly, whereas computing the input data in an ANN requires multiplication and addition for calculating $y = W^T x$, SNNs only requires the addition of the weights due to the binary nature of spikes. Secondly, SNNs do not need to compute zero values in the vector-matrix multiplication, so all outputs that the ReLU function turns to zero are ignored. This resulted in the converted SNN emitting, on average, 0.142 spikes per neuron.

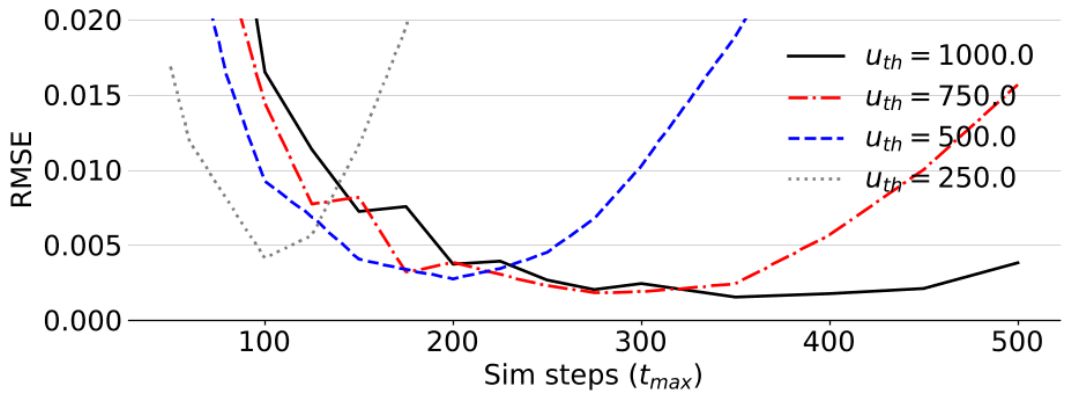


Figure D.2: Error of the converted LeNet DCNN to an SNN when varying the number of simulation steps and the membrane voltage.

E

Spiking OS-CFAR

The *spiking OSCFAR* (S-OSCFAR) is based on a winner-takes-all topology, where input neurons compete to determine the output spiking behaviour. Namely, the neuron takes latency-encoded spikes representing the neighbour values X_N and the CUT value X_C, connected with synaptic weights $w_N = -1$ and $w_C = k$, respectively. This architecture is represented in the left side of Fig. E.1.

For generating each input spike train S_i, the corresponding input value X_i is mapped to a single spike time t_i by using latency encoding (4.5), i.e., $t_i = t(X_i)$. In the special case of the CUT value, the spike time is computed after modifying X_C with the scaling factor α

$$t_C = t(\alpha X_C)\,. \tag{E.1}$$

On each simulation step, the S-OSCFAR neuron iterates over the CUT value and the N neighbour cells, increasing the membrane voltage u when a spike is present

$$u(t+1) = u(t) + \sum_{i \in N+1} S_i(t) w_i\,. \tag{E.2}$$

The update rule (E.2) corresponds to that of an integrate and fire neuron without leak current. This means that the membrane voltage u accumulates all the information that arrived in previous simulation steps and the information stays unaltered over time.

When the membrane voltage $u(t)$ reaches the threshold voltage $u_{th} = 1$, the neuron generates a spike $O(t) = 1$

$$O(t) = \begin{cases} 1, & \text{if } u(t) > u_{th} \\ 0, & \text{otherwise} \end{cases} \tag{E.3}$$

After generating a spike, the membrane voltage is reset, and the neuron stays in a refractory state until the end of the simulation. The introduced neuron model establishes a competition between the CUT value X_C and the neighbour values X_N, as X_C will generate a spike if it arrives before k neighbour cell spikes. The neuron behaviour is depicted in Figure E.1 for a window with six neighbour cells.

Object detection algorithms in radar pipelines are typically applied after the Fourier transform to filter background and noise from the actual objects in the scene.

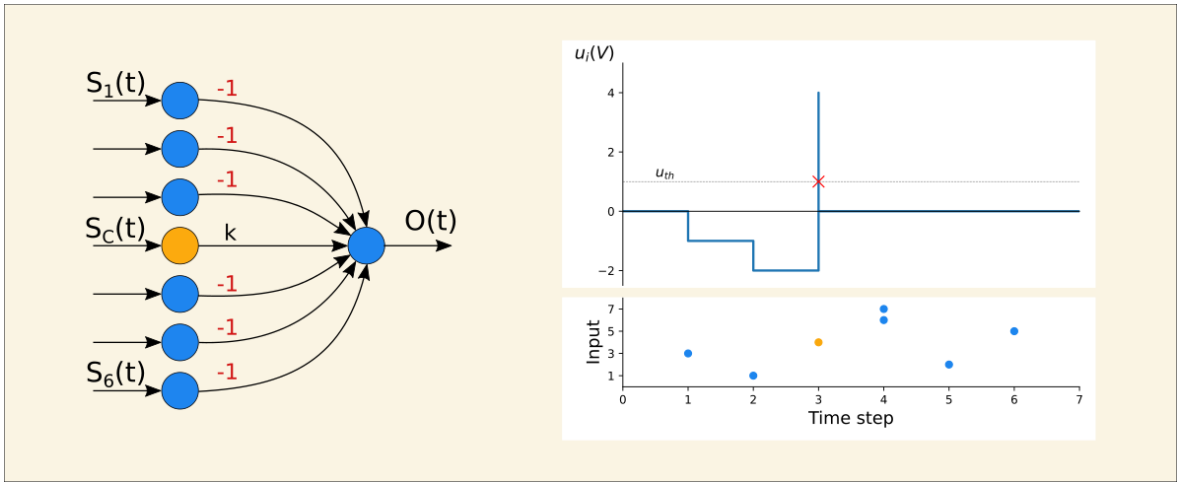


Figure E.1: On the left, a diagram of an S-OSCFAR neuron taking as input the spike trains from 6 neighbour cells and the CUT value, represented in blue and orange, respectively. The values on top of the connections represent the synaptic weights. On the right, a plot of the membrane potential of the neuron and a scatter plot of the input spikes at the top and at the bottom, respectively. Neighbour cell spikes are represented in blue, and the CUT spike is represented in orange. The neuron starts at a resting potential $u(0) = 0$ and has a threshold potential of $u_{th} = 1$. After generating a spike, the neuron stays in a refractory state.