Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://cas.tudelft.nl

CAS-2021-4607929

M.Sc. Thesis

Spiking CA-CFAR Implementation for Radar Target Detection

Bastiaan Johannes Antonie van Otterloo

Abstract

Radar systems have been used for decades to detect targets on the ground and in the air. The radar signal is transformed into a range-doppler image that distinguishes each detected object by range and velocity for further processing. A target detection algorithm is used to filter noise and clutter. Each target can be in a region with a different noise level; a simple threshold would yield false positives or miss detections depending on this value. To solve this problem, a Constant False Alarm Rate or CFAR is desirable. A CFAR detector estimates the noise surrounding each target and uses a dynamic threshold based on this.

Spiking Neural Networks are the third generation of Artificial Neural Networks where, instead of continuous signals, the input is encoded into trains of spikes over time. These networks have a potentially efficient hardware implementation instead of the older generation Artificial Neural Networks and could run directly at the sensor edge, lowering latency and power consumption.

This thesis will explore a Spiking Cell Averaging CFAR implementation and attempt to use its desirable properties like a temporal average over multiple radar frames, mimicking the non-coherent integration sometimes done in radar processing. It is shown that some configurations will behave similarly in a simulated environment with additive white gaussian noise.

Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Spiking CA-CFAR Implementation for Radar Target Detection

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Computer Engineering

by

Bastiaan Johannes Antonie van Otterloo
born in Purmerend, the Netherlands

This work was performed in:

Circuits and Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Spiking CA-CFAR Implementation for Radar Target Detection"** by **Bastiaan Johannes Antonie van Otterloo** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 30-11-2021

Chairman: ______________________
dr.ir. T.G.R.M. van Leuken

Advisor: ______________________
dr. E. Kalali

Committee Members: ______________________
dr.ing. R.K. Bishnoi

____
MSc. D. Maksimiuk

Abstract

Radar systems have been used for decades to detect targets on the ground and in the air. The radar signal is transformed into a range-doppler image that distinguishes each detected object by range and velocity for further processing. A target detection algorithm is used to filter noise and clutter. Each target can be in a region with a different noise level; a simple threshold would yield false positives or miss detections depending on this value. To solve this problem, a Constant False Alarm Rate or CFAR is desirable. A CFAR detector estimates the noise surrounding each target and uses a dynamic threshold based on this.

Spiking Neural Networks are the third generation of Artificial Neural Networks where, instead of continuous signals, the input is encoded into trains of spikes over time. These networks have a potentially efficient hardware implementation instead of the older generation Artificial Neural Networks and could run directly at the sensor edge, lowering latency and power consumption.

This thesis will explore a Spiking Cell Averaging CFAR implementation and attempt to use its desirable properties like a temporal average over multiple radar frames, mimicking the non-coherent integration sometimes done in radar processing. It is shown that some configurations will behave similarly in a simulated environment with additive white gaussian noise.

Acknowledgments

Thank you to everyone who supported me during the past few years and the patience of many people. Thanks to Rene in particular, who allowed me to work on this project unconditionally. Also, thanks to Darek, who inspired many of the ideas and evaluations used in this thesis. Special thanks to dr. Zhengyu Peng, for allowing me to use his Python radar simulator and DigitalOcean for supporting me to run the hundreds of simulations on their compute instances.

Bastiaan Johannes Antonie van Otterloo
Delft, The Netherlands
30-11-2021

Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. 2, 5, 13, 19, 20

CFAR Constant False Alarm Rate. xi–xiv, xvii, 2, 5, 10–12, 19–22, 25, 27, 31–36, 38, 39, 41, 43–45, 49, 53, 55–57, 59, 63, 64

CUT Cell Under Test. xii, xv, 11, 19–22, 33, 59

FMCW Frequency-Modulated Continuous Wave. 23, 36

IF Integrate-and-Fire. xiv, xv, 21, 26, 27, 32, 42, 45, 46, 48, 53–55, 57, 59, 60, 69–71, 73–75

LIF Leaky Integrate-and-Fire. xiv, xv, 21, 26, 27, 29, 32, 35, 42, 45–51, 54, 55, 57–59, 61–63, 67–69, 71–73

ODE Ordinary Differential Equation. xiii, 27, 28

RCS Radar Cross Section. 5, 57

ROC Receiver Operating Characteristic. xiii–xv, 9, 19, 36–38, 41, 44–51, 56–58, 67–75

SNN Spiking Neural Network. xii, xvii, 2, 5, 10, 16, 19–23, 25, 28, 32, 35, 39, 63, 64

SNR Signal-to-Noise Ratio. 8–11, 38, 39, 44, 45, 48, 56, 57, 63

Introduction 1

This chapter introduces the motivation and contributions of this thesis. During the writing of this thesis, many ideas have come and gone. All the relevant work is described, and the source code is available at Github[1].

1.1 Motivation

Target detection is one of the essential concepts in radar systems. Both in military and civilian use-cases, correctly detecting targets in radar images is critical. For example: in aviation, both on the ground and in the air, it is used for detecting (other) aeroplanes, and it has become an essential sensor on many cars for adaptive cruise control and collision avoidance. Automotive radars have evolved into advanced sensors that directly communicates each target's range and angle over the car's internal communication

[1] https://github.com/bas921/spiking-cfar

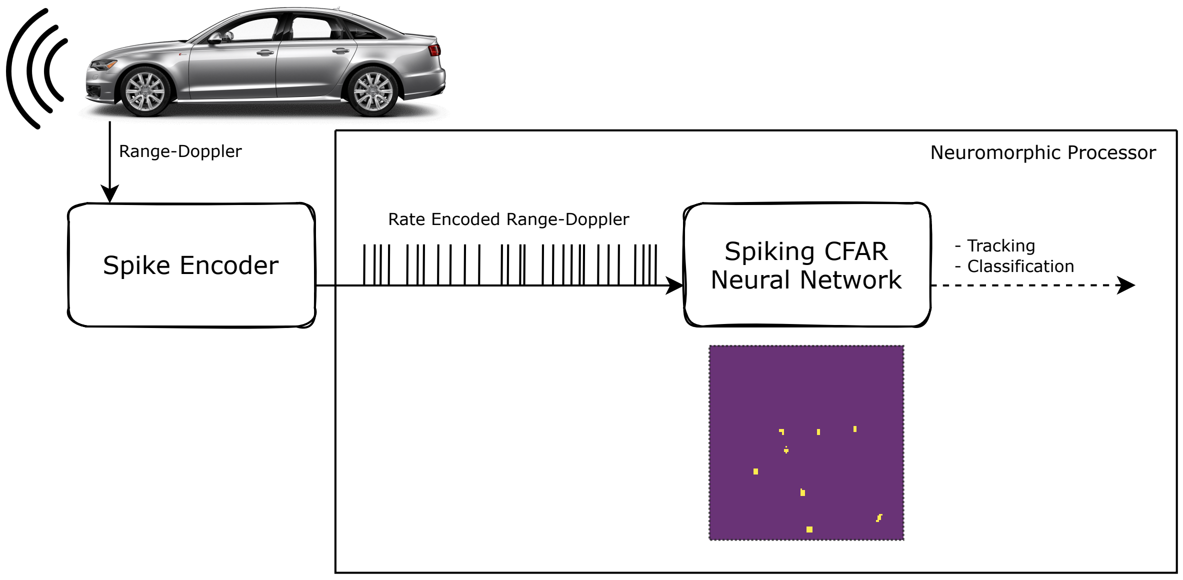


Figure 1.1: A simplified illustration of one of the use-cases. While optimisations are still possible, this is the most realistic use-case this thesis will explore; some (pre-)processing is still done outside the neuromorphic hardware. Ideally, the neuromorphic processor will be used at the edge inside each sensor's package and handle most, if not all, processing. The dashed arrow is meant to denote future work.

network.

A classical but practical class of target detection algorithms is Constant False Alarm Rate (CFAR). There are multiple implementations of CFAR, but they all attempt to estimate the background noise and compare based on this estimation. Recent studies into target detection algorithms look into using Artificial Neural Networks (ANNs) to implement more efficient target detection algorithms; many use (a part of) CFAR to do this [1, 24, 16].

ANNs are a set of flexible systems that can approximate many functions using learning based on the nervous systems of almost all animals. Spiking Neural Networks (SNNs) further try to match the brains' methods by introducing a temporal aspect in the form of spikes, giving them a low power footprint. Due to advances in computing power, conventional ANN and SNN have grown in popularity, and many professional and consumer products use them. SNNs are particularly interesting near the edge inside sensor hardware, allowing sensors to potentially improve the quality of information produced without significantly increasing power consumption. While most studies implement SNN algorithms only in simulation, a literature survey examining SNN hardware [22] confirms that interest in physically implementing these algorithms is growing, but more research is required.

This thesis will implement the CFAR pre-processing stage inside an SNN simulator to use the exciting properties of SNNs. This implementation is particularly interesting as a pre-processing stage when further processing is done in an SNN.

1.2 Contributions

- A simulated network architecture that replicates Cell-Averaging CFAR behaviour.
- A novel simulated recurrent network architecture that attempts to improve on the previous implementation.
- An analysis of several network components and performance of the several algorithms.
- A Python library (SPYDAR: Spiking Python Radar) for running, processing and analysing CA-CFAR and Spiking CA-CFAR simulations.
- A radar dataset generator that uses an existing simulator to generate radar frames.
- An optimised SNN simulation architecture to run many runs simultaneously on multiple compute cores.

1.3 Outline

This thesis consists of 7 chapters, including this introduction. In chapter 2, all the background information is provided to understand most concepts used in this thesis. Chapter 3 goes into the previous literature regarding the topic of this thesis. Chapter 4 shows all the steps taken to get the final result and how these results are evaluated.

Chapter 5 shows and describes the gathered results. Chapter 6 will discuss what the results indicate and what caveats are still there. In chapter 7, the work and results are combined into a conclusion taken from this thesis.

Background 2

This chapter introduces all the relevant background concepts on which this thesis will build. The first two sections introduce radar, and target detection, primarily Constant False Alarm Rate (CFAR), and the following section introduces conventional Artificial Neural Networks (ANNs) and Spiking Neural Networks (SNNs) and their differences.

2.1 Radar

Electromagnetic radiation, the fundamental principle of RAdio Detection And Ranging or radar, is a well-studied topic. Principle discoveries of electromagnetism started well before the beginning of the common era. Heinrich Hertz invented the basis of radio transmissions using the electromagnetic spectrum around 1887. In 1888, he discovered that some waves pass through certain materials (notably the lower frequencies) while others reflect (higher frequencies) [6]. Hertz himself ironically thought his discovery was useless. It was not until 1904 that German inventor Christian Hülsmeyer used this principle for ship detection to avoid collisions, one of the critical uses of the radar in modern times. Nowadays, many applications use radar.

2.1.1 Radar Principles

The radar equation determines the range characteristics of a radar, given in equation (2.1) [3]. The maximum range is determined by the transmission power P_s, the antenna gain G, wavelength λ, Radar Cross Section (RCS) σ and the minimum detectable receiving power $P_{e_{\min}}$. The equation shows that as components in the numerator increase, the maximum detectable range also increases.

$$P_e = \frac{P_s G^2 \lambda^2 \sigma}{(4\pi)^3 R^4} \implies R_{\max} = \sqrt[4]{\frac{P_s G^2 \lambda^2 \sigma}{P_{e_{\min}} (4\pi)^3}} \tag{2.1}$$

The most trivial example of a radar signal is a pulse, where a short sinusoidal signal is transmitted and reflected by a target. In this case, only the location of an object can be determined and is given by the round-trip time of the pulse in equation (2.2), with c being the speed of light and t the time between transmission and reception.

$$R = \frac{ct}{2} \tag{2.2}$$

Measuring velocity with radar is a bit more involved and uses the Doppler effect. The Doppler frequency is the frequency shift in the received signal. Humans also experience this effect with moving sirens on emergency vehicles, which introduces a frequency change dependant on the object's velocity. As can be seen in equation (2.3) [15], as

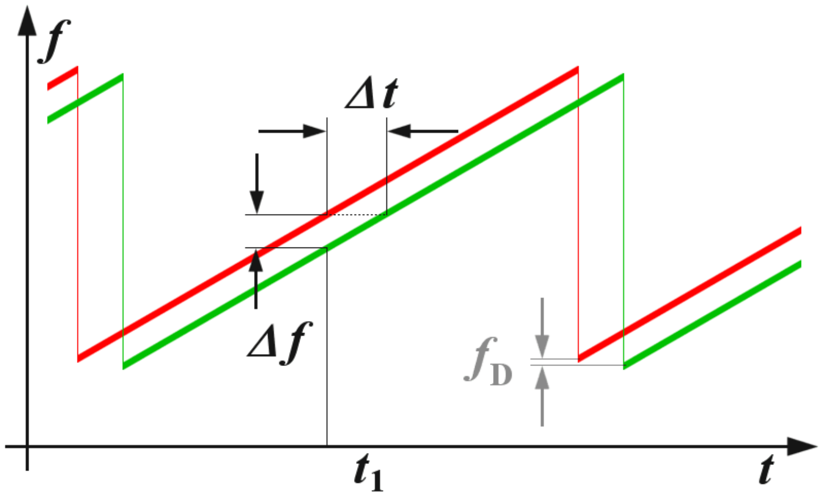


Figure 2.1: A chirp in the frequency spectrum. The difference in frequency is related to the distance. The height difference is the doppler frequency. (image from radartutorial.eu)

the relative velocity v between the object and observer increases, so does the Doppler frequency f_D. Thus something transmitting waves moving towards the observer will increase its frequency.

$$f_D = \frac{2v}{\lambda} \tag{2.3}$$

The single pulse radar is insufficient to measure the Doppler shift as the transmission stops before receiving the response. Contrary to this method, Continuous Wave (CW) radars transmit and receive at the same time. A continuous-wave radar will usually require two separate antennas for transmitting and receiving because of this. When transmitting an unmodulated signal, a continuous wave radar cannot distinguish targets nor measure distance. The reflection will also be continuous, and there is no way of knowing where it starts and stops. It is, however, possible to measure the Doppler shift (i.e. velocity) of a single target.

To be able to distinguish between targets and measure range, modulation of the signal is required. One possible modulation technique is Frequency Modulation (FM), modulating the signal's frequency over time. By modulating a high-frequency carrier wave over a specific bandwidth, the radar captures range and Doppler information as seen in figure 2.1. In radar, each rising edge is called an (up-)chirp. Accessing this information will require multiple chirps since it will be ambiguous whether the frequency difference is from the delay or Doppler shift.

Velocity is given by the phase difference ω between two chirps (equation (2.4)) [14]. The quality (max. distinguishable and resolution) of the velocity information is determined by the time of each chirp (T_c) and the number for chirps i.e. frame time (T_f) (equation (2.5) and equation (2.6)).

$$v = \frac{\lambda\omega}{4\pi T_c} \quad (2.4) \qquad v_{\max} = \frac{\lambda}{4T_c} \quad (2.5) \qquad v_{\text{res}} = \frac{\lambda}{2T_f} \quad (2.6)$$

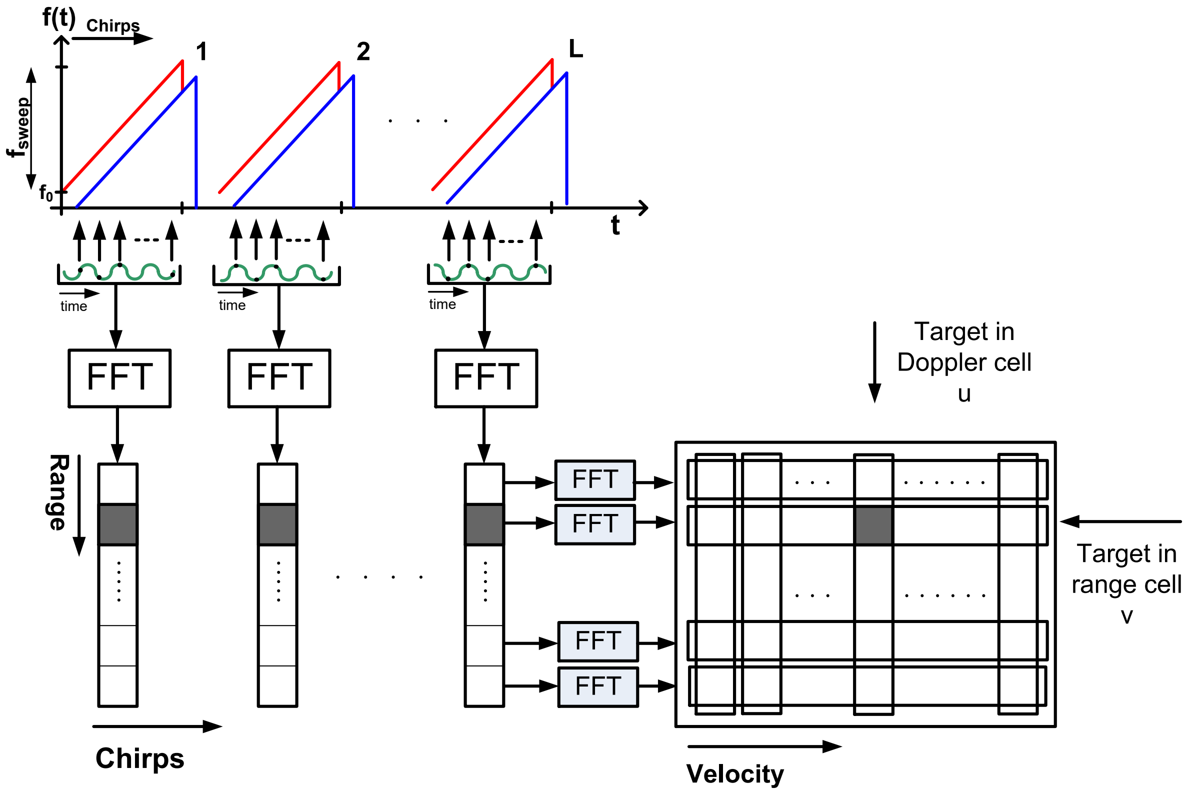


Figure 2.2: Range-Doppler processing requires two sequential FFT arrays. First over range, then over chirps. (image from C. Schroeder et al. [21] © 2011 IEEE)

2.1.2 Radar Processing

In radar hardware using FMCW, a mixer stage mixes the transmitted signal with the received signal. The resulting signal is called a beat or intermediate signal. The range of each target is directly related to the frequency in the beat signal: a low-frequency component means a nearby target. This range information contains a small error for non-stationary objects because of the offset caused by the Doppler frequency. The phase information over multiple chirps is required to get the Doppler frequency. When doing Fourier transforms over each chirp (also called fast time), and consecutively over all the chirps per range bin (also called slow time), the resulting 2D image is called a Range-Doppler map or image (see figure 2.2). On one axis is the range of each object, and on the other, the velocity. The Range-Doppler map can distinguish equidistant objects with different relative velocities; equidistant objects with the same velocity are still ambiguous in range-doppler images.

2.2 Target Detection

Determining whether something is a target is a non-trivial problem. One approach would be to set a simple threshold. In that case, a change in the noise level could change the radar characteristics and underperform, which is undesirable: the detection

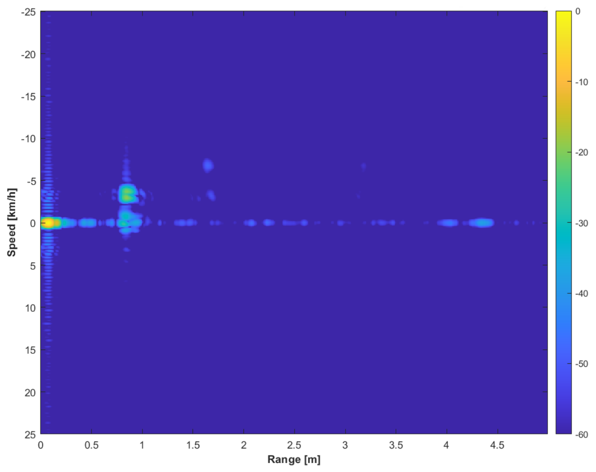


Figure 2.3: Example of a Range-Doppler map of a rotating metal plate at a 1m distance. The target moves around a circle as it rotates, and the targets at 0m are cross-talk between antennas and usually are filtered out.

rate could be higher with a lower threshold. When deciding on a value, there are four distinct outcomes:

1. The decision rule decides there is a target, and there indeed is (detection or true positive).
2. The decision rule decides there is no target, but there is (miss or false negative).
3. The decision rule decides there is a target, but there is not (false alarm or false positive).
4. The decision rule decides there is no target, and there is not (correct rejection or true negative).

The probability of a false alarm is an important design constraint, as the signal processing (or human) behind it needs to handle this number of false detections. For example, a pulse radar transmitting 15000 pulses per second and 400 range bins would require (15000)(400) is six million decisions per second. With a probability of false alarm $P_{\text{FA}} = 10^{-8}$, it would trigger a false alarm about once every 17 seconds, while a $P_{\text{FA}} = 10^{-6}$ would trigger a false alarm about six times per second. If the false alarm rate is constrained, it can be set to a fixed number of false alarms per hour that the system can feasibly process.

2.2.1 Probability

Radar antennas receive both the noise and signal. The essence of target detection is determining whether a part of the measurement contains noise and the signal.

Both the noise and signal have their probability distribution dependant on the noise power and radar characteristics. The only way to statistically increase the probability of detection without increasing the false alarm rate is by increasing the Signal-to-Noise

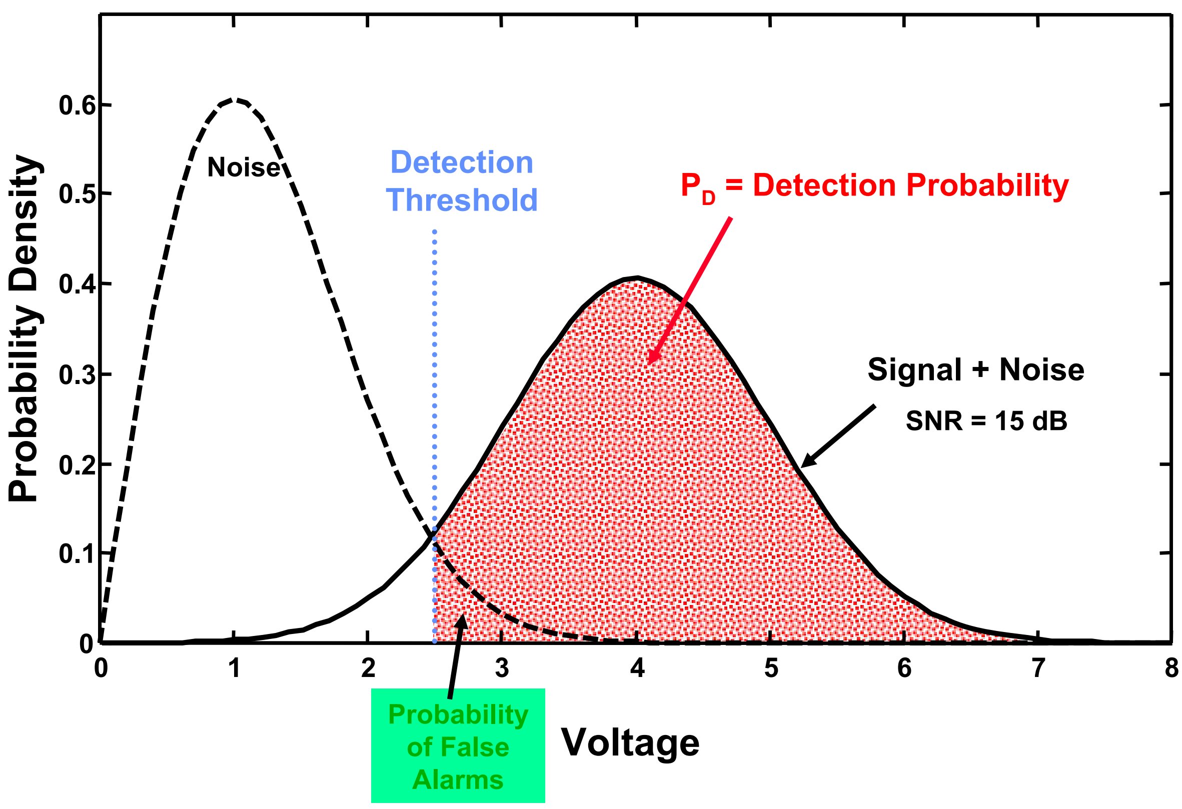


Figure 2.4: The signal always overlaps to some degree with the noise. This means that there is always a probability of a false alarm. Ideally, the probability of a false alarm is constrained. (image from MIT Lincoln Lab)

Ratio (SNR). In this case, the whole distribution shifts to the right, decreasing the noise and signal area overlap. Similarly, decreasing the probability of false alarm will result in a lower probability of detection, as can be seen in figure 2.4. The Neyman-Pearson criterion maximises the P_D under a set constraint for P_{FA}.

A Receiver Operating Characteristic (ROC) curve can be used to show the performance of each decision rule or algorithm. It shows the relation between P_{D} and P_{FA} (see figure 2.5). The linear line in the middle represents the performance expected from a random binary classifier. Ideally, a perfect classification never produces a false-positive result. Realistically, this is never the case, and there is always a probability of a false alarm. As the probability of detection approaches 1, the probability of a false alarm will as well.

Note that P_D and P_{FA} are related to $P_{\text{miss}} = 1 - P_D$ and $P_{\text{TN}} = 1 - P_{\text{FA}}$ respectively. As such, the ROC curve covers the whole confusion matrix and can be used to characterise the complete performance of the classifier.

2.2.2 Range-Doppler Integration

To effectively increase the SNR of a radar system, (non-)coherent integration can be used [20]. By integrating multiple range-doppler frames, the system essentially averages

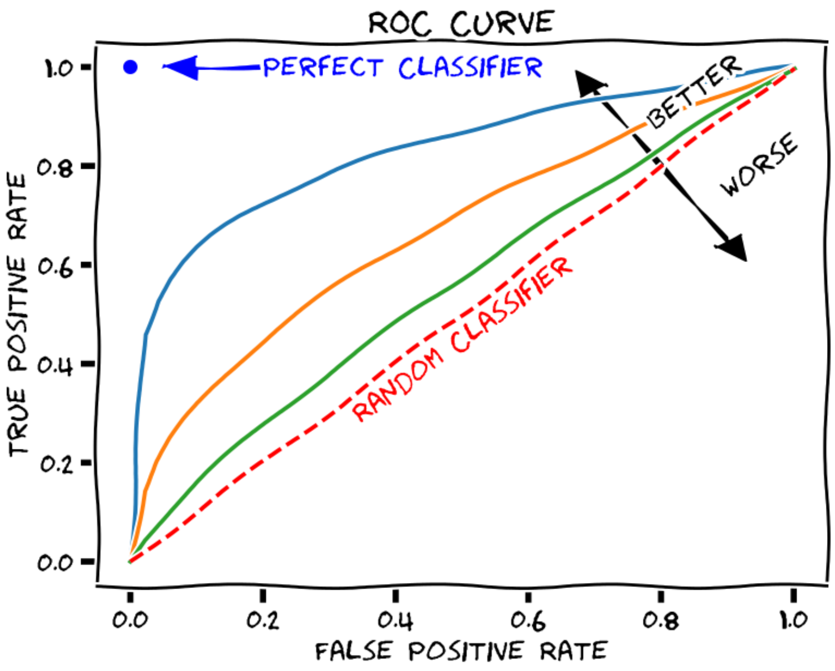


Figure 2.5: The ROC curve shows the performance of a decision rule by plotting the relation between P_D and P_{FA}.

out the noise. When this is done in the complex range-doppler image, it is called coherent integration. After passing through the (magnitude) square law detector, it is called non-coherent integration. Non-coherent integration cannot use the phase information and, because of that, is less effective than coherent integration. Typically, integration is an expensive computational operation, but with SNNs, this is partially built-in since each neuron is an integrator.

The gain is dependant on the number of frames to be integrated. With two SNRs, SNR_1 and SNR_N, the SNR of a single frame and N integrated frames, respectively, the non-coherent integration gain is defined in equation (2.7). Sometimes non-coherent integration gain is also called the integration improvement factor.

$$G_{\mathrm{nc}} = \frac{\mathrm{SNR}_1}{\mathrm{SNR}_N} \tag{2.7}$$

While the non-coherent integration gain is usually better, the gain for simple point targets used in this thesis varies between N at best and $\sqrt{N}$ at worst.

2.2.3 Constant False Alarm Rate

The class of detectors that aim to maintain a constant false alarm rate are called CFAR detectors. Contrary to simple detectors, CFAR detectors have a dynamic threshold that is dependant on an estimation of the noise power ($\hat{\sigma_w^2}$) and some factor α (equation (2.8)) [20]. The additional α factor compensates for the fact that the noise power is an

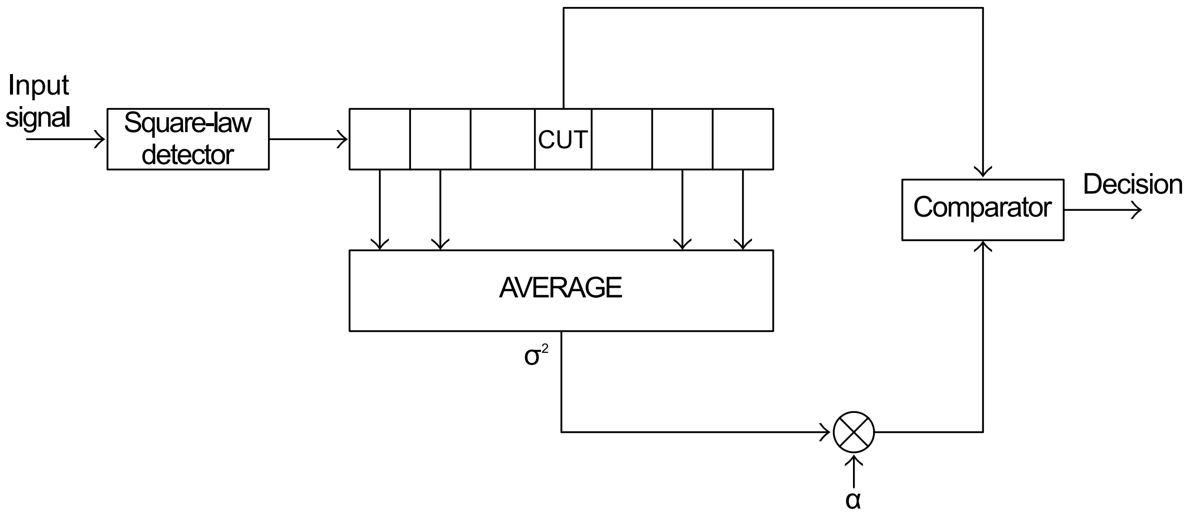


Figure 2.6: A schematic of a CA-CFAR detector. CFAR uses neighbouring cells to calculate some noise estimation used as an input for the threshold comparator. Note that the cells directly adjacent to the CUT are guard cells and not used in the average. (original image from Hong et al. [13])

estimation. In this case, it is assumed the interference is Gaussian and thus exponential distributed due to the square-law detector. Using maximum likelihood estimation, the noise power $\hat{\sigma_w^2}$ can be defined as the average of N-cells around the evaluated cell x_i called the Cell Under Test (CUT) (equation (2.9)). Intuitively, this makes sense; taking the average cells around a potential target has a high chance of accurately estimating the noise. This type of CFAR is called Cell Averaging CFAR or CA-CFAR. This methodology has two problems: (1) When other targets are nearby, they might inadvertently raise the threshold (target masking). The cell with a lower SNR is no longer detected. (2) Heterogeneous interference may produce false alarms or misses on edges between different noise floors [8, 4]. An example of heterogeneous interference caused by clutter is an open road with buildings or trees on the side. The open road will likely have a lower noise power than the buildings where echoes can bounce around multiple times. The edge between these two different noise power is called the clutter edge and the false alarm caused by this are considered a shortcoming of CA-CFAR.

$$\hat{T} = \alpha\hat{\sigma_w^2} \tag{2.8}$$

$$\hat{\sigma_w^2} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.9}$$

CA-CFAR is the most used CFAR algorithm. In terminology, the CUT has both guard cells and reference cells surrounding the CUT. The guard cells are ignored and in place to partially solve the interference problem caused by itself or nearby targets, and the references cells are the cells averaged to estimate the noise power. When solving for α, the noise is assumed to be Additive White Gaussian Noise. When modeled in a exponential distribution (due to the square-law detector), α ends up being dependant only on P_{FA} and N therefore being a constant false alarm detector (equation (2.10)).

Ordered Statistics (OS)-CFAR is another variation that aims to solve the problems in heterogeneous noise. The algorithm sorts reference cells and selects the k-th rank

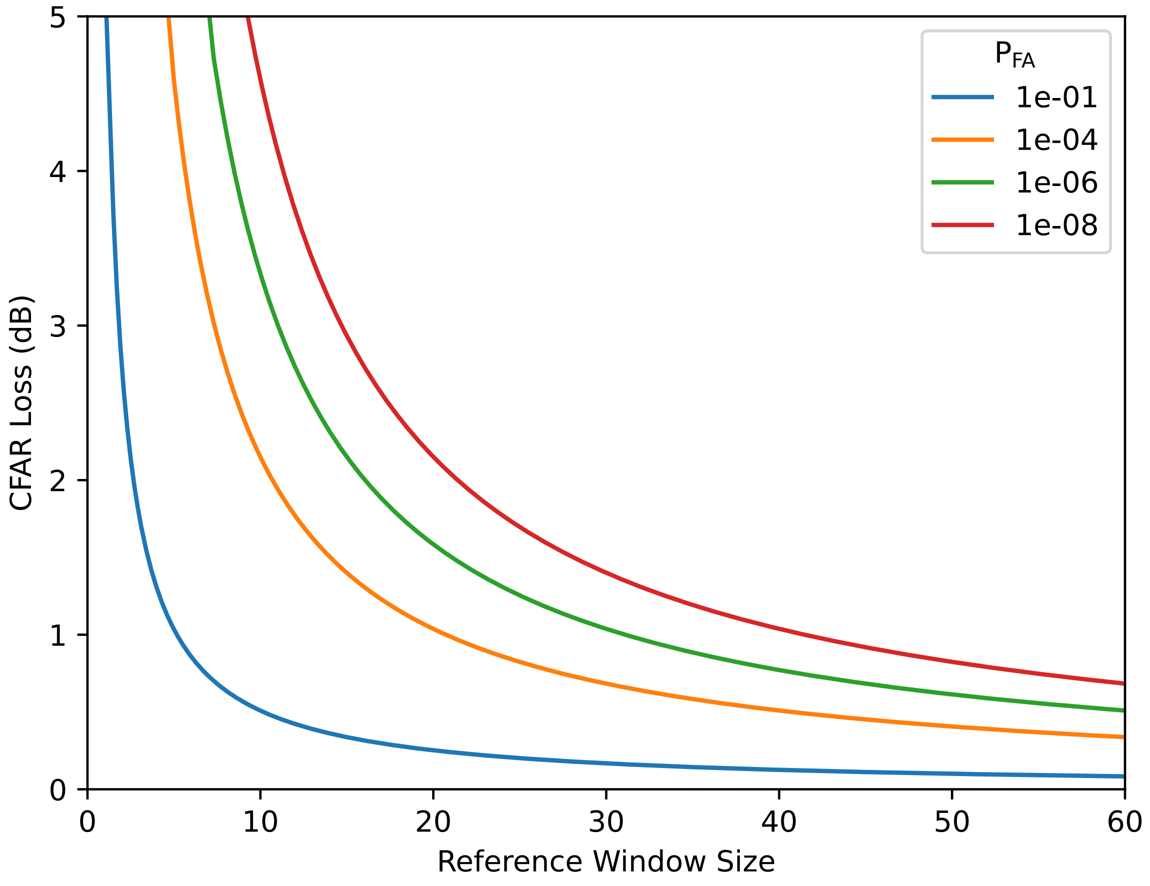


Figure 2.7: The CFAR loss plotted for several P_{FA}. As the number of reference cells increases, the noise estimation gets more accurate, and the loss will be lower. The lower desired probability of false alarms rely heavily on the noise estimation and, as such, will have a larger CFAR loss.

order value (e.g. the median) as noise estimation instead of averaging. There is again a constant α that determines the P_{FA} that is multiplied with the chosen value.

$$P_{\text{FA}} = (1 + \frac{\alpha}{N})^{-N} \iff \alpha = N(P_{\text{FA}}^{-1/N} - 1) \tag{2.10}$$

Because the noise is estimated, a lower number of the reference cells that estimate the noise will result in a larger error. This error causes some signals to be lost as the noise will be over-estimated; this loss is called the CFAR loss and indicates what is a a good number of reference cells [20]. As can be seen in figure 2.7, as the P_{FA} decreases for a given P_{D} (in this case 0.9), the loss becomes bigger. Below ten reference cells, This is the case because, as the desired P_{FA} gets lower, the CA-CFAR relies more on its reference cells. When these reference cells do not give a good estimation because there are too few, the noise is overestimated, and detections will be missed. Note that this loss does not consider the shortcomings of real-world losses, like target masking and clutter edges.

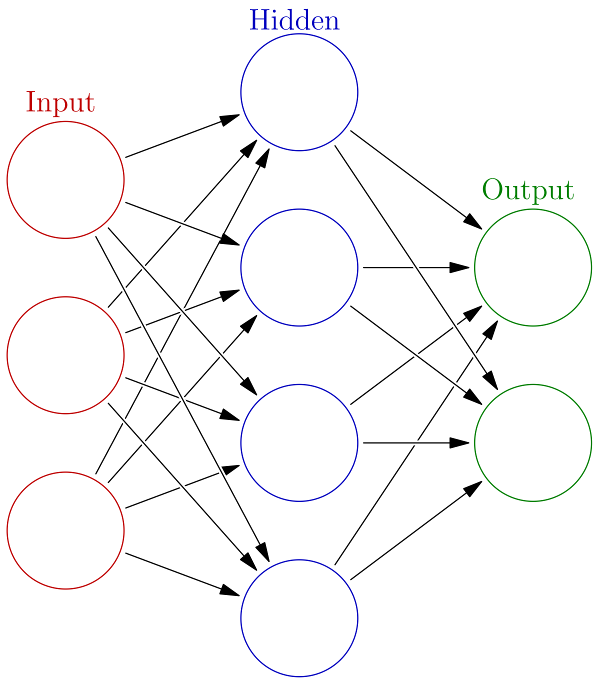


Figure 2.8: The network as a whole acts as a function that is determined by its connections and weights. The number of hidden layers characterises the depth of a neural network. (image by Glosser.ca at English Wikipedia)

2.3 Artificial Neural Networks

Traditionally, scientists modelled Artificial Neural Networks (ANNs) after the nervous system of almost all animals, including humans. The nervous system is responsible for almost all information processing, ranging from hundreds to billions of neurons and trillions of synapses. This highly complex system is well-studied, yet many details are still unknown. The exponential growth in computing power reignited the interest in implementing ANNs, and large homogeneous compute structures can do the necessary computation relatively efficiently and fast. Nowadays, deep neural networks achieve results in fields like image and speech recognition, previously only privy to humans.

2.3.1 Neural Network Principles

Neural Networks consist of two elementary components: neurons and synapses. The neurons are the nodes of a network, while synapses are the edges connecting the neurons. Neurons take the inputs from synapses and produce an output related to their input. The depth of a network is characterised by the number of layers apart from the in-, and output layers also called the hidden layers (figure 2.8) [10].

Modern Artificial Neural Networks usually work as a black box in which the connectivity determines what function the network approximates. Each connection between neurons has an associated weight that determines how meaningful the relationship between these neurons are for that function. The change of this weight facilitates learning and is usually done using backpropagation; the learning algorithm can use the continuous nature of these networks to calculate partial derivatives of the error between the input and expected output. This propagates back through every layer, starting at the output going back to the input. When repeating this process many times, the network is trained to a particular function that hopefully approximates the desired function. A complex deep neural network can have many (hidden) layers through which the input propagates towards the last output layer. Generally, a higher number of layers constitutes a higher complexity of the function when training it properly.

In biology, most of these connections are chemical processes that consist of channels triggered by neurotransmitters. The signals between neurons are no longer continuous but spikes, introducing a temporal element to the network. A spike occurs when the membrane potential, i.e. the voltage inside a neuron, exceeds a threshold. This sets off a chain reaction that causes a spike to travel along the axons (figure 2.9) and in turn causes the synapse to release neurotransmitters, increasing the membrane potential in the connected neurons. During the spike generation and shortly after, the neuron can not generate another spike. The period in which this is the case is called the refractory period and limits the rate at which neurons can fire. After the refractory period, the neuron's membrane potential returns to its resting or reset voltage. Many factors can influence whether the neuron will generate a spike and how neurons are connected. Donald Hebb explained so-called synaptic plasticity in the Hebbian theory and can be summarised as "Cells that fire together wire together", stating that a pre-synaptic (input) neuron that spiked just before the neuron did the neuron will become more sensitive to its input. Although this is thought to be the primary mechanism driving learning and memories, it is still poorly understood.

2.3.2 Neuron Models

Simulating neural networks requires modelling the individual elements of a network. Neurons have been studied intensively across animals. Alan Hodgkin and Andrew Huxley described a model by looking at a giant squid axon, which made experimentation easier due to its large size. This Hodgkin-Huxley model described the membrane potential as capacitance and multiple channels as a conductance, essentially having a capacitor as the cell membrane and (variable) resistors as channels [12, 9]. While this model is complete and accurate, the mathematical model contains multiple non-linear differential equations making it hard to analyse.

To reduce complexity both in simulation and hardware, neuron models have been simplified into an integrate-and-fire model, which can be described in a single linear ordinary differential equation (equation (2.11)). The applied input current, which in biological neurons would be the channels opening in response to neurotransmitters, charges the capacitor, which increases the membrane potential. A spike is fired after crossing a set threshold, after which the membrane potential is reset to the resting voltage.

$$I(t) = C\frac{du(t)}{dt} \tag{2.11}$$

Biological neurons constantly lose charge such that they return to their resting

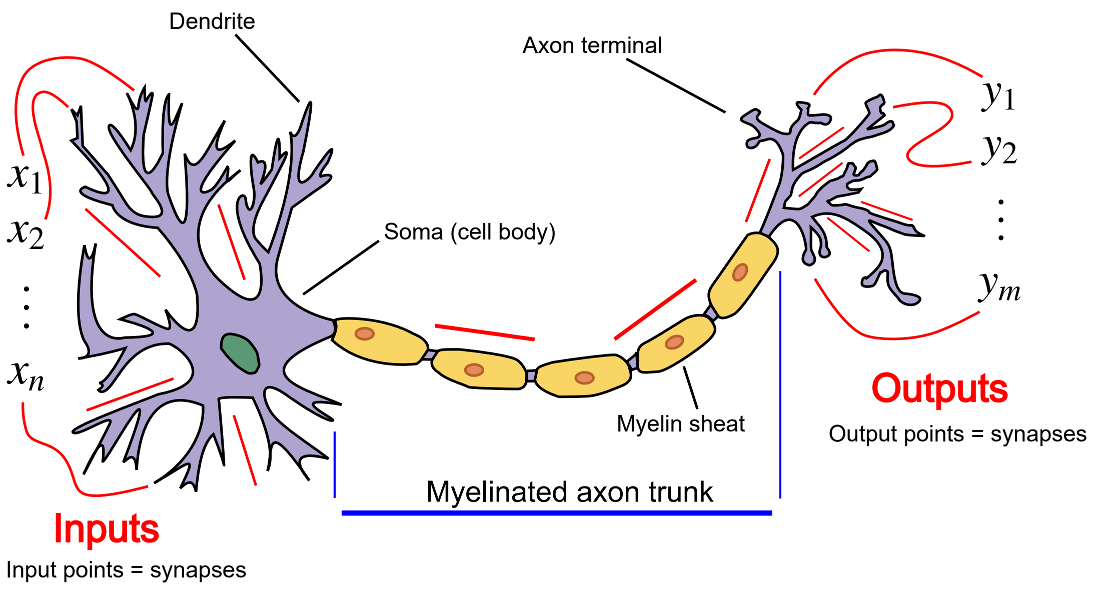


Figure 2.9: Synapses connect with other neurons. As a neuron receives inputs on its dendrites from other synapses, it will increase the membrane potential inside the neuron and spike at a certain threshold. (image by Egm4313.s12 at English Wikipedia)

potential after input spikes. This leaky behaviour is modelled in the Leaky Integrate-and-Fire neuron model (equation (2.12)) by adding a resistor parallel to the capacitor [9]. In this equation the input current $I(t)$ is reduced by the leaky current $(u(t)-u_0)/R$, where u_0 is the resting membrane potential. The constant RC can be expressed as the time-constant τ, it takes approximately 5τ for the capacitor to discharge back to 0V. In many neurons, the time constant is in the order of 10 milliseconds, but from a hardware point of view, this can be anything as hardware permits. For example, due to the area, the time constant might be constrained to a few nanoseconds. These mathematical models do not include the firing itself or reset; this is solved in the exponential (leaky) integrate-and-fire but is unnecessary for simulation purposes. Furthermore, in simulation, the input current is replaced by discrete voltage weights over some set time step. In this case, the equation can be simplified to $\tau du(t)/dt = u_0 - u(t)$.

$$I(t) - \frac{u(t) - u_0}{R} = C\frac{du(t)}{dt} \implies \tau\frac{du(t)}{dt} = u_0 - u(t) + RI(t) \tag{2.12}$$

An example of a voltage curve is shown in figure 2.10. The refractory period is twofold: during the spike itself, the membrane potential is not changed meaningfully, and the membrane potential is inhibited below its resting state after the action potential, making it harder to fire again.

In addition to neurons being excited by spikes, neurons can be inhibited. Inhibition lowers the membrane potential such that it becomes more challenging for the neuron to generate a spike. Inhibitory behaviour can prevent neurons from spiking altogether over a prolonged period, as long as the inhibiting spikes override the excitatory spikes.

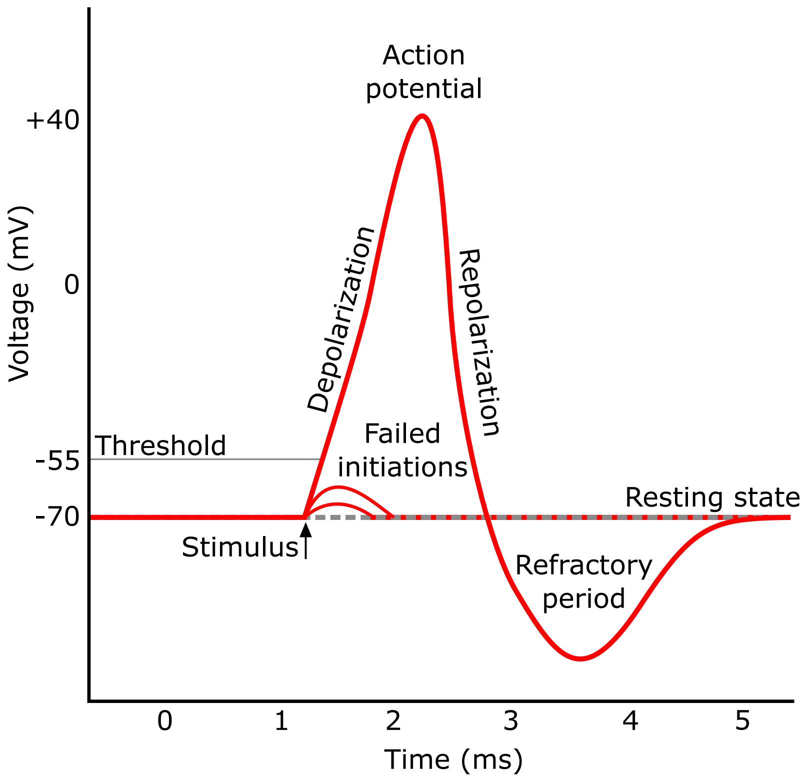


Figure 2.10: The action potential or firing of a neuron. As the membrane potential increases, the voltage crosses a threshold triggering the neuron to fire. Not all input results in a spike but does cause the membrane potential to increase.

After inhibition, the neuron's membrane potential returns to the resting state similar to excitation. In biology, synapses release inhibitory neurotransmitters that open channels to allow negatively charged ions (as opposed to positively charged ions when being excited) inside the neuron membrane, decreasing the voltage. In models or simulations, inhibitory behaviour can be seen as a negative current or negative weight, respectively.

2.3.3 Spike Encoding and Decoding

The input and output of an SNN are in the form of a series of spikes or spike trains. These spike trains can be described in multiple ways, generally divided into rate and temporal encoding [17] (figure 2.11).

Rate encoding modulates the spike rate over a specific period. The average rate over such a period corresponds to the actual input or output value. This can be done over a single neuron or population of neurons, where their combined average spike rate is representative of some input or output. Rate encoding can be seen as a dense encoding type since more spikes are required to represent data.

The temporal encoding uses the timing of individual spikes to encode the information. An example of temporal encoding is Threshold encoding, where a spike is generated when the intensity of an input signal crosses a certain threshold. The in-

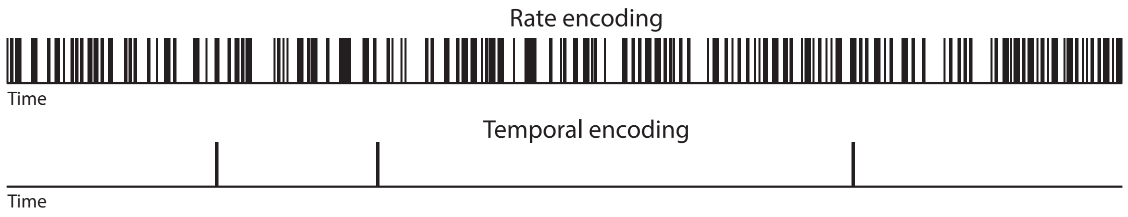


Figure 2.11: Two main ways to encode spikes are rate and temporal encoding. With rate encoding, the information is encoded in the frequency of the spikes alone, while temporal encoding also encodes information in the timing of individual spikes. Note that with rate encoding, spikes are not evenly spaced to fit a specific frequency.

formation density of the encoding scheme can be increased by using Step Forward encoding, where the difference between two values is used to compare.

All spikes must be independent when encoding signals into spikes to ensure the encoding itself does not influence the network. This can happen in simulation with rate encoding: when all spikes are dispatched simultaneously at simulation start, it can cause output spikes that would not have otherwise occurred. In that case, the simulation itself caused the output spike and not the neural network. Especially learning algorithms can reinforce this predictable behaviour. Stochasticity is introduced in the encoding in the form of Poisson encoding to prevent spike dependence. In this case, the spiking rate is not precisely given by individual spikes but on average.

Related Work 3

As early as 1993, scientists searched to improve the Constant False Alarm Rate (CFAR) algorithm due to its well-known limitations outside of Gaussian noise [2]. Due to the hardware back in the day, this study did not use the advanced Artificial Neural Networks (ANNs) used today, but rather a decision algorithm on what conventional CFAR algorithm should be used. The majority of neural networks being a conventional ANN, there is not much research into specifically enhancing CFAR with Spiking Neural Networks (SNNs). Most literature in CFAR changes the noise estimation algorithm or makes choices in what algorithm to use (adaptive CFAR).

3.1 Artificial CFAR

Several literature studies attempt to improve the conventional CFAR algorithm by adding an ANN somewhere in the pipeline. Although these studies are not related to this thesis as there is no learning component, the results and the way results are measured are. Carretero et al. [7] replace the whole algorithm with a shallow 2-layer neural network. The network outputs a new confidence estimation by training the network on prior information (whether a target is present or not). Because of the black-box nature of a trained ANN, any information about the P_{FA} is lost but is attempted to relate to the threshold output empirically for comparison purposes. The algorithm gets moderate improvements with homogeneous Gaussian noise but seems to classify clutter edges better, preventing them from appearing as false-alarm detections. Results are shown on a P_{D} vs SNR plot.

Similarly, Lin et al. [18] replaced the algorithm with an ANN but instead used a deep network of convolutional layers. The network aims to give a better estimation of the noise map such that it removes the masking effect caused by other targets. It does perform better but at a high computational cost. The results are presented in Receiver Operating Characteristic (ROC) plots.

Zhao et al. [25] look specifically at avoiding clutter caused by reflections of waves at sea. They combine a clustering algorithm (DBSCAN) with an ANN, where the clustering attempts to filter the outliers caused by clutter, and the ANN is used to estimate the remaining background clutter. The estimated background interference is compared with the Cell Under Test (CUT) as usual. Results are evaluated using plots with the detection probability and SNR.

3.2 Spiking CFAR

Before starting the work on this thesis, there was no known literature about using SNNs for radar target detection or CFAR in particular. In June 2021, Lopez et al.[19] published a relevant paper about implementing a CFAR algorithm (OS-CFAR) in SNNs. Similar to the work in this thesis, the authors used mathematically derived weights to implement the conventional algorithm in a spiking architecture. Instead of a well known SNN simulator, an own implementation was used. As the neuron model was an integrate-and-fire, without any leakage or refractory period, the algorithm was converted to spiking behaviour, potentially missing out on some of the temporal enhancements that spiking architectures inherently bring. Essentially, the authors converted conventional ANN operations into the temporal domain using rate encoding.

Instead of selecting the kth-smallest value, the authors simplify this by combining the comparator and summing in the output neuron. The CUT weights constitute both the number of spikes required to trigger the output neuron and the comparison by using inhibitory and excitatory spikes. Most notably, the architecture requires all inhibitory spikes by the neighbour cells to arrive before the spikes by the CUT to prevent incorrect output spikes.

The authors use the root mean square error for measuring the difference between conventional DFT and spiking DFT. There is no analytical comparison between the two CFAR algorithms.

4 Design and Methodology

The goal of this work is to approximate the Constant False Alarm Rate (CFAR) algorithm in an Spiking Neural Network (SNN). This chapter describes the theory and design of the Spiking CFAR. The first section introduces the essential concepts and principles of Spiking CFAR. The following sections describe the encoding, simulation, network model and metrics, respectively.

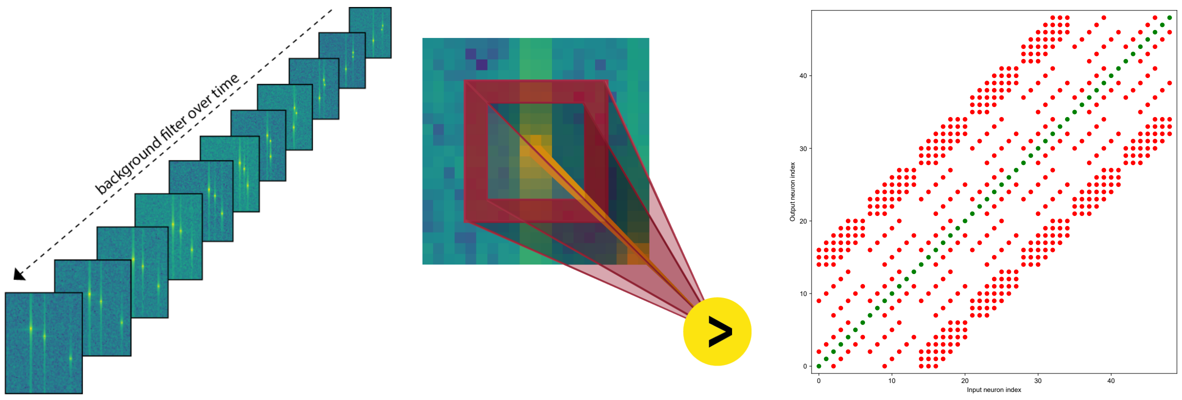
4.1 Spiking CFAR

The basic principle of Spiking CFAR is the so-called exposure of range or range-doppler images to an SNN. This network filters out noise based on the principle of the Cell Averaging CFAR algorithm. The resulting output is similar to the output of the conventional CFAR algorithm, and a subsequent network layer can use this output directly for further processing or classification. The use of an SNN introduces several low-frequency filtering elements without using extra processing elements. Another argument for doing pre-processing in SNNs is power consumption, as SNNs are known for their power efficiency due to their use of spikes. It is, however, not possible to accurately compare the power consumption without having a working hardware simulation.

Everything in this thesis is based on the working of the Integrate-and-Fire (IF) and Leaky Integrate-and-Fire (LIF) neuron models. This last type of neuron accumulates spikes by increasing its membrane potential. The SNN can use trains of spikes to represent the values of pixels in an image. In the case of rate encoding, the weight of each synapse will effectively act as a multiplier in the post-synaptic neuron. Without any leakage or reset, the membrane potential can thus be seen as the value multiplied by some weight w_{ij} and as a result, any neuron threshold will effectively be a threshold based on the input.

With CFAR, a simple threshold is not sufficient: it uses an estimation of the noise power. It is evident using simple algebra that a threshold voltage u_t can be subtracted from the membrane potential $u_j(t)$ and results in the same threshold operation centred around 0 (equation (4.1)). Inhibitory spikes can use this property by subtracting the threshold value from the Cell Under Test (CUT) and avoiding the issue of needing a dynamic threshold in the neuron model. This mechanism places the threshold at the synapse level: each neuron connection determines the effective threshold. The threshold can further be divided into a static and dynamic threshold. The dynamic threshold is, as before, determined by neighbouring cells to estimate the noise power. The static threshold is a set value allowing to filter low-frequency noise.

$$u_j(t) > u_t \iff u_j(t) - u_t > 0 \implies \underbrace{u_j(t)}_{\text{excitatory}} - \underbrace{u_{\text{dynamic}}(t)}_{\text{inhibitory}} > u_{\text{static}} \tag{4.1}$$



(a) (b) (c)

Figure 4.1: The several mechanisms of Spiking CFAR are illustrated. a) Because the membrane potential is discharged over multiple frames, the output neurons stay inhibited, making it harder for them to fire in the next frame. b) All spikes generated by the red pixels inhibit the yellow output neuron, while the spikes generated by the yellow pixel (the CUT) excite the output neuron. The output neuron also has a static threshold at which it will spike. c) The connectivity dots in a 7x7 example network. At the cost of synapses, each convolution will be executed in parallel. The green dots are excitatory connections, the red dots inhibitory. In this case, the network uses a single guard neuron and a reference neuron on each side.

The combination of inhibitory spikes and the temporal aspect of SNNs does come with several caveats, like the order in which excitatory and inhibitory spikes arrive can influence whether the output layer generates a spike. In addition, the leaky component conveniently left out of the above description attenuates both the signal and threshold value. While this results in a loss of power, both inhibitory and excitatory spikes are affected. It could, however, unintentionally remove low power targets that the normal CFAR would have picked up.

By leveraging the network connectivity, the SNN can perform all convolutions simultaneously at the cost of synaptic connections. With CA-CFAR, the guard and reference cells are now called reference and guard neurons, respectively. The reference neurons will inhibit the output neuron, while the CUT neuron will excite the output neuron. The guard neurons are unconnected.

Three mechanisms are all inherent when using Spiking CFAR (figure 4.1):

(a) The inhibitory spikes from previous frames can temporarily cause inhibition, acting as a high-pass filter on a temporal level. This can prevent the neuron from spiking in the next frame.

(b) Besides the dynamic threshold, there is also a static threshold introduced dependant on the threshold of the neuron, acting as a high-pass filter on a spatial level.

(c) Instead of using a convolution, the network processes all cells in parallel. Both the sum and comparison are combined in the output neuron.

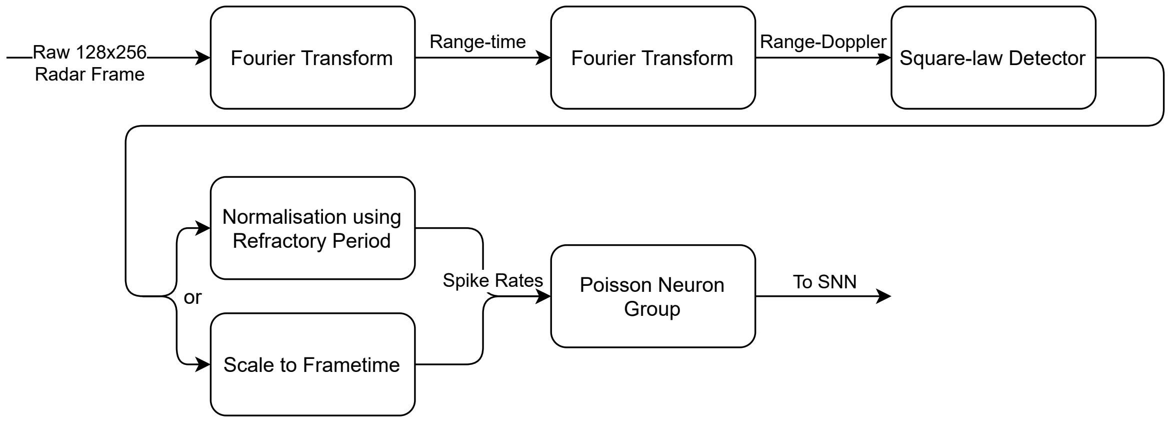


Figure 4.2: The encoding pipeline before it enters the SNN. Normalisation allows for lower spike rates when reflections have a uniform power distribution like targets at a constant distance.

4.2 Encoding Pipeline

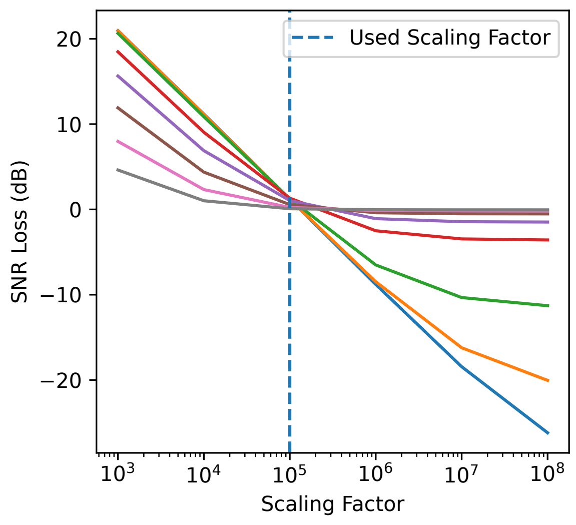
The initial experiments in this thesis are based on the RFBeam MR3003 radar. This 77GHz Frequency-Modulated Continuous Wave (FMCW) radar produces a frame consisting of 128 chirps every 50ms. In reality, each chirp takes about 140μs (from which 27.6μs is transmitting the chirp), and each frame is 17.9ms. The missing 32.1ms goes to processing and can not be disabled without reprogramming the firmware. Each chirp consists of 256 real numbered samples. The 50ms per frame deadline is taken as a design parameter, and all results are based on this. Later results are based on a simulation using Radarsimpy that use these same characteristics.

By applying the Discrete Fourier transforms as outlined in chapter 2, the output is a raw unprocessed range-doppler image. Alternatively, a Chirp Z-Transform can be used to zoom in at a specific range and velocity effectively, but the resulting image is similar to doing a larger Fourier transform and cropping the image. In this thesis, any (optimised) variation of the Discrete Fourier transform, like the Fast Fourier Transform, will be generalised as the Discrete Fourier transform. Since all the relevant information is in the frequency components, the next step takes the magnitude of the complex image and thus results in a 1D or 2D image consisting of range and range-doppler, respectively.

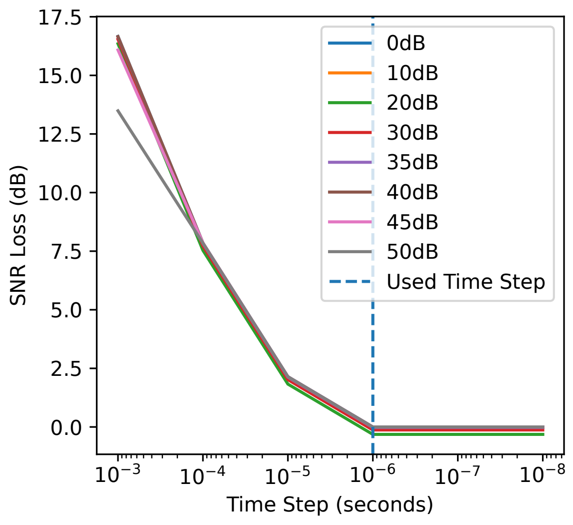
The assumed noise is Gaussian (Additive White Gaussian Noise specifically), and the exponential distribution is used to estimate the power of a range-doppler cell. The image is passed through a square-law detector to get the power of each cell.

4.2.1 Normalisation and Quantisation

There are two options to convert each pixel value to spike rates: (min-max) normalising them to a set scaling factor (which will become the maximum spike rate of each frame) or directly scaling the input values to the frame time of each frame. Min-max normalisation does not handle outliers well with a significant precision loss; low power



(a) normalised



(b) direct

Figure 4.3: The quantisation results in a loss dependant on the chosen scaling factor or simulation time step per noise figure. a) Min-max normalisation suffers from significant losses depending on the distribution of the target power. As the scaling factor increases, the loss becomes a gain as the distance between the signal and noise increases. b) Direct input effectively clips all the values above the maximum spike rate determined by the simulation time step, resulting in a higher loss. As soon as the dynamic range covers the whole input power range, this loss becomes zero or slightly negative due to the quantisation of integer values.

reflections are suppressed because of high power targets. A shorter simulation time step will generally result in higher spike rates and thus a lower quantisation error, resulting in a better approximation of the original algorithm. However, in the case of simulation, this costs more compute time, or in the case of hardware, a higher power consumption. This trade-off has to be made: normalisation reduces the probability of detection, higher spike rates increase the power consumption and simulation time. When no normalisation is used, the spike rates are converted to scale accordingly with the frame time such that for each frame, the input rate is the same as the value used in the standard algorithm. The rates are used for input in the Poisson Neuron Group, which introduces stochasticity in the spike timings to avoid spike dependence.

The signal is being quantised into atomic units (spikes), and as a result, the spike rate is an integer value. This causes a quantisation error related to the maximum spike rate determined by the refractory period and simulation time step. As the simulation time step is shortened, higher spike rates can be achieved, and the dynamic range of spike rates gets larger and the error smaller when using normalisation. However, a shorter time step results in longer simulation times (weeks) and is not practical.

In figure 4.3, the effects of both input methods are shown. At longer simulation time steps, the loss will be significant with both methods. Nevertheless, at high enough spike rates, both methods can provide acceptable results and simulation times. However, min-max normalisation will attenuate low power signals if there is also a high power

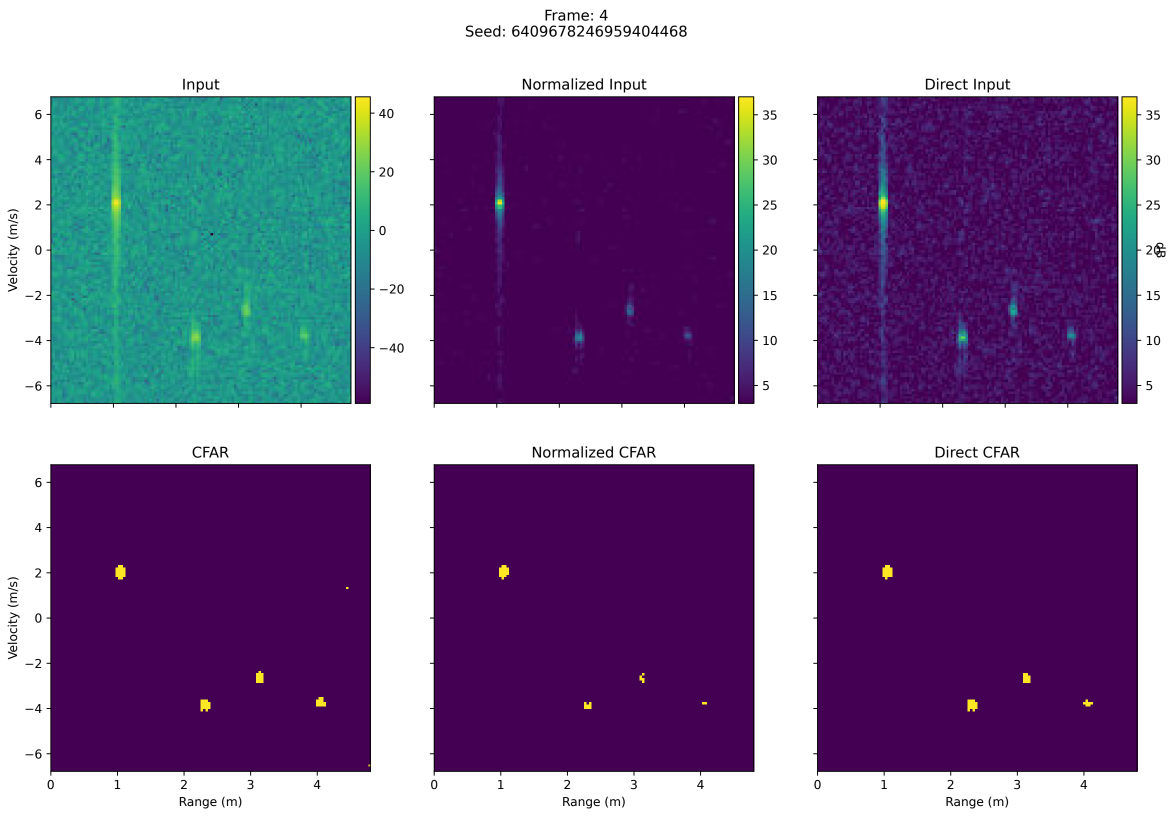


Figure 4.4: The (conventional) CFAR output of each spike encoding method when using a scaling factor of 10^5, time step of 10^{-6} and a P_{FA} of 10^{-8}. While the normalised input has a lower average spike rate, it attenuates both the signal and noise with a high power target. This can cause problems for quantisation, causing missed detections.

signal, lowering the detection rate. While both methods are tested, the direct method is preferred. The quantisation error at the used simulation time step is low enough, and it does not adjust the signal for each frame.

As can be seen in figure 4.4, low-power noise gets squished into either a 0 or 1. The direct input will handle this better on average as the chosen normalisation scaling factor never influences it. When there are only low power targets, the noise has a higher dynamic range and could provide better performance. The opposite is true when there is a high power target: the noise will be attenuated and make detection of other low-power targets impossible (also shown in figure 4.4). The floating-point input does not have this problem (apart from the floating-point precision).

4.3 Spiking Simulation

In order to simulate the SNN, the Python library Brian 2 [23] is used. This library facilitates the simulation of many neurons and synapse models given by differential equations. Various methods like Euler can then solve these equations, but generally, an exact method is used since the used equations are linear. Each layer consists of a

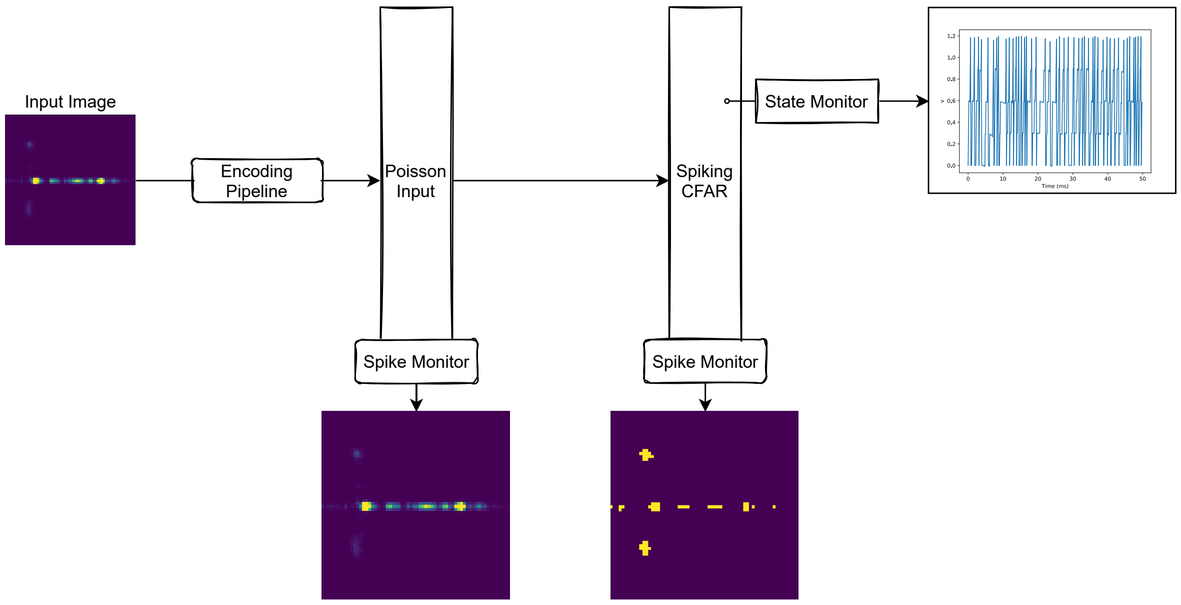


Figure 4.5: The structure of the simulation elements in Brian. Each spike monitor will register all the spikes that happen within that network. The state monitor can be used to inspect individual neurons or synapses. Larger simulations do not record input spikes to save memory.

Neuron group and is connected by synapses. Monitors are used to capturing spike rates and state variables like membrane voltage (figure 4.5). To save memory, usually, only the state of specific neurons is monitored. The spike times of all neurons are monitored in order to generate the in- and output images. Larger simulations do not monitor the input spikes. When simulating in Python, the complete network state can be stored and restored, which is used to reset the whole network back to its resting state.

4.3.1 Simulated Elements

For input, a Poisson group is used to avoid spike dependence. The Poisson group is a Neuron group that uses a dynamic threshold based on pseudo-random number generation and spike rates: $R < f_n * dt$ with R being a random variable sampled from a uniform distribution between range 0 and 1, f_n is the input rate for neuron n and dt the simulation time step. A neuron generates a spike when the threshold is met. Consequently, the Poisson group produces a spike rate that is normally distributed around the desired rate. From this equation can also be observed that when $f_n = 1/dt$, a spike will be generated at each time step, the so-called clipping effect.

The other neuron groups all use the same IF or LIF model. Because the simulation discretises over each step, the model can be simplified to calculate voltage using $u = q/C$. This results in neuron model equation (4.2), using time constant $\tau = RC$ where R is the resistance in Ohm, and C is the capacitance in Farad. As a result, the voltage is the only independent variable, making simulations faster. The threshold and spiking are not built into the model equation itself, neither are the refractory period and reset. The simulator handles both; after a spike, the simulator will prevent any neuron changes or spike and reset the membrane potential.

$$\frac{du(t)}{dt} = \frac{u_0 - u(t)}{RC} \tag{4.2}$$

The synapse model adds a set weight in voltage to the post-synaptic neuron on a pre-synaptic spike, i.e. when a neuron spikes, the corresponding neuron membrane potential increases. This is shown in equation (4.3a), where $u_j(t)$ is the new membrane potential for time t with a time step Δt. It takes the membrane potential from the previous time step and adds the sum of weights w_{ij} for each synapse connection between neuron i and j. This weight can be anything but is generally lower than the threshold.

$$u_j[t] = u_j[t - \Delta t] + \sum_i w_{ij} S_i[t - \Delta t] \tag{4.3a}$$

$$S_i[t] = \begin{cases} 1, & \text{if a spike occured at time t} \\ 0, & \text{otherwise.} \end{cases} \tag{4.3b}$$

When a neuron is in its refractory period, the membrane potential can not be modified by incoming spikes nor by the discharge of the capacitor. The refractory period effectively determines the maximum spike rate of neurons, together with the simulation time step. Figure 4.6 shows the effects of the simulation time step and refractory period on a 500Hz input. As the time step gets too high, the variance in output frequency goes down to zero. On the other hand, the refractory period lowers the chance of a spike occurring. E.g. with the input of 500Hz and a refractory period of 2ms, the neuron is in its refractory state for half the time, and during this time, it can not spike again; this limits the effective spike rates to around 250Hz. The refractory period is not used in the Poisson input group but is used in the output group.

4.3.2 Neuron Behaviour

The LIF model can be defined as a (linear) ODE. This can be seen in figure 4.7, the analytical solution is identical to the output of the Brian 2 simulation. This is as expected since the Brian 2 internals uses the same method to solve each time step.

The leakage of energy potential happens continuously and will thus lower the spikes required for detection. As can be seen in figure 4.8, even with the inhibitory and excitatory spike rates being equal (b), an output spike is still possible. As the difference between the spike rates grows, output spikes become more or less likely. A (non-zero) static threshold (u_{static}) can filter these random output spikes. When using a zero static threshold, compensation is necessary to avoid random false alarms. The excitatory synapses can be delayed, or the initial reset potential can be set on a negative offset. The effects of the leakage and threshold can also be seen in figures 4.9 and 4.10, which shows the resulting average output spike rate for both IF and LIF neurons, with and without a static threshold of 1V, which is five times the base weight (0.2V). As can be observed in figure 4.10, the output firing rate is reduced by about a fifth, and the output spikes at equal rates are attenuated. This behaviour is desirable for CFAR as it estimates a more stable thresholding function.

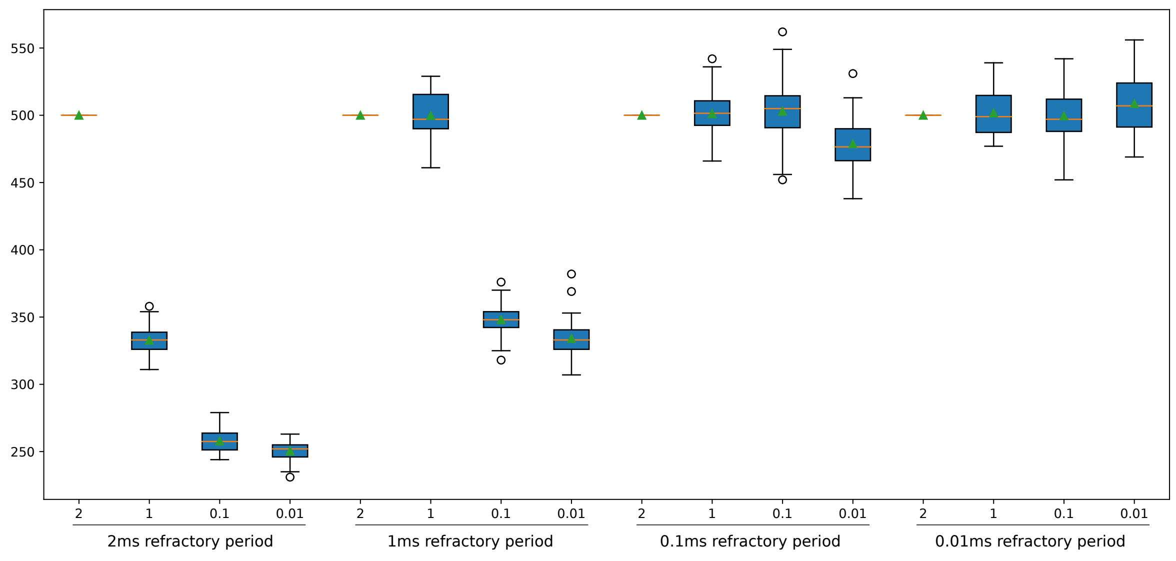
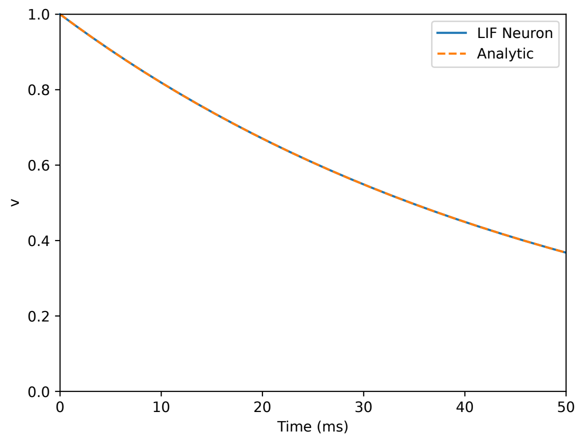
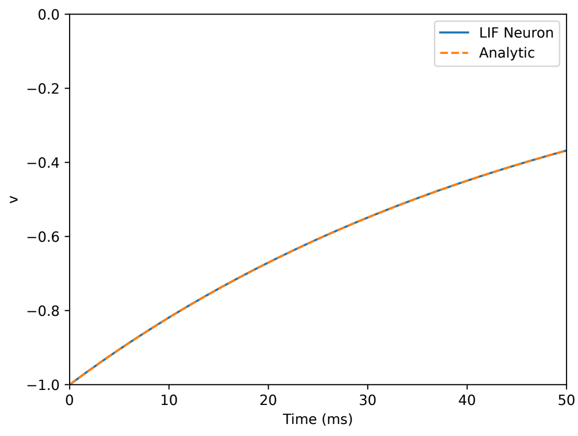


Figure 4.6: The output spike rates of a steady 500Hz Poisson input for multiple time steps and refractory periods. A time step (dt) that is too high causes the output rate to clip to $1/dt$. A high refractory period (T_{ref}) lowers the chance of a spike occurring, effectively lowering the mean output spike rate to $(f_i dt)/T_{\text{ref}}$, with input rate f_i. Input spike rates above $1/dt$, clip to $1/T_{\text{ref}}$ as expected.



(a) excitatory

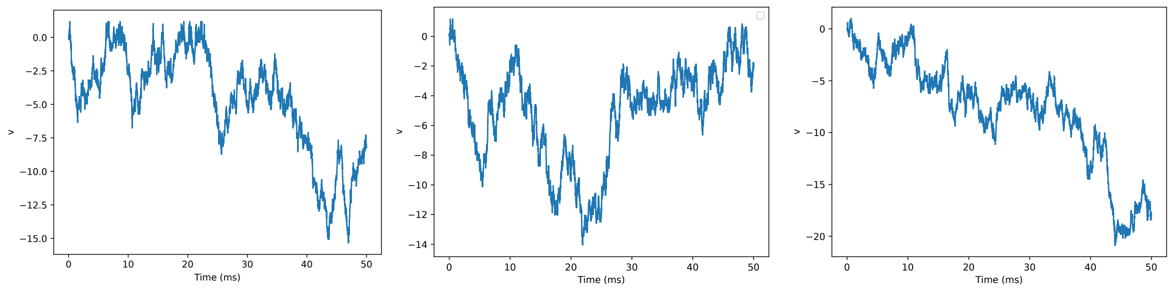


(b) inhibitory

Figure 4.7: The behaviour of a simulated LIF neuron with time-constant $\tau = 50$ms compared with its analytical function. Since Brian 2 uses an exact solution for linear Ordinary Differential Equations (ODEs), the results are identical. a) A neuron is charged to 1V and discharges over time. Anything over 1V would have immediately reset the neuron after spiking. b) A neuron has been inhibited to -1V and slowly discharges back to 0V.

4.3.3 Scheduling

Brian is a clock-driven simulator and, as such, uses discrete time steps to simulate the SNN. The clock-driven approach makes the simulation more computationally effi-



(a) -50Hz (b) equal (c) +50Hz

Figure 4.8: The membrane potential of three output neurons, with a constant excitatory input rate of 2500Hz. a) The inhibitory spike rate is lower than the excitatory spike rates and thus causes a reliable output spike train. b) The inhibitory and excitatory spike rates are equal. Spikes can still occur since energy potential is lost due to leakage. c) The inhibitory spike rate is 50Hz higher than the excitatory spike rate. Output spikes are rare but can still happen due to the Poisson input, especially at lower spike rates.

cient than event-driven simulators, especially with larger networks [5]. The time step introduces an error in spike timings since all threshold conditions are evaluated simultaneously, making Poisson spike rates even more important as it lowers the chance that neurons spike in the same time step.

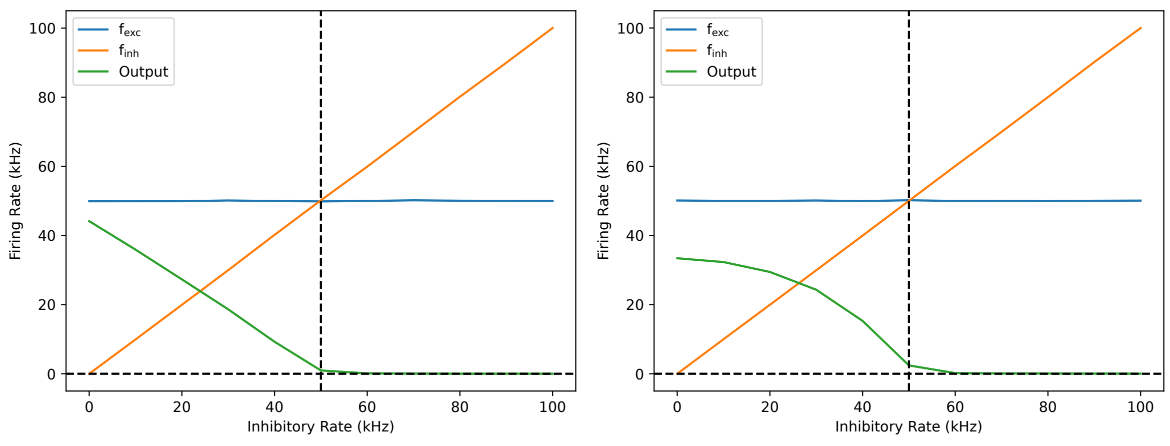
The clock-driven approach also means that everything is processed in a specific order. Mainly, Brian uses the following schedule, the simulation:

1. updates the neuron groups. In the case of LIF neurons, it applies the leakage.
2. compares thresholds and registers spikes.
3. checks synapses if a pre-synaptic spike occurred (as was just registered) and applies the model.
4. resets all neurons that spiked according to the reset conditions.

When the time between spikes approaches the simulation time step, the spikes will appear synchronised at the maximum possible spike rate and effectively clip the input values. This synchronisation will even occur when using Poisson inputs. While the effects of a small number of neurons clipping are negligible, it will affect performance when a large portion of the input image becomes clipped, as it would in the conventional algorithm. Normalisation will prevent this but with the caveats mentioned above. Unintended synchronisation can be prevented by choosing a shorter simulation time step and refractory period.

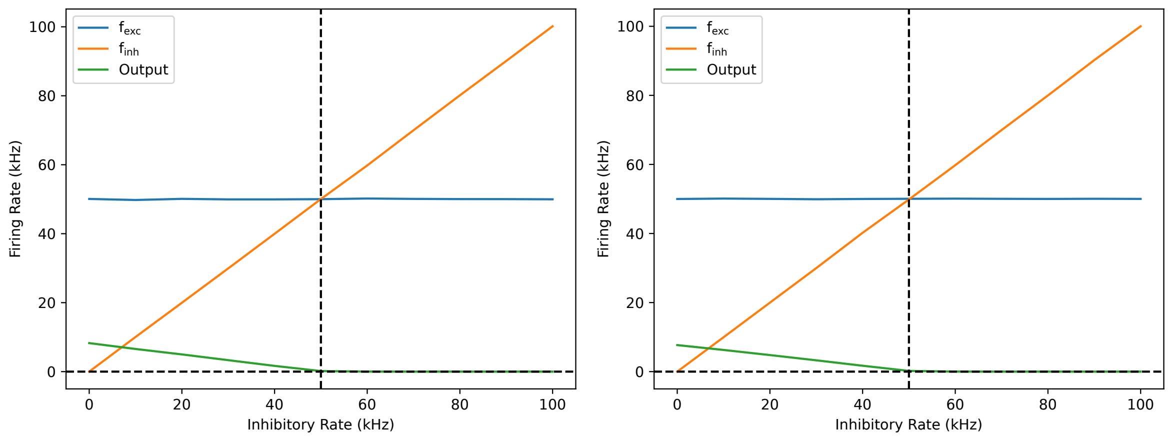
4.3.4 Optimisations

There are several optimisations in the Brian simulator that can be utilised. There are two types of code execution: runtime and standalone code generation. With runtime code generation, Brian either uses the Numpy library or Cython C++ compiled functions. These compiled functions are imported in Python and called like any other



(a) integrate-and-fire (b) leaky integrate-and-fire

Figure 4.9: The output spike rate with a fixed excitatory spike rate and increasing inhibitory spike rate. There is no static threshold.



(a) integrate-and-fire (b) leaky integrate-and-fire

Figure 4.10: The output spike rate with a fixed excitatory spike rate and increasing inhibitory spike rate, using a 1V static threshold.

function during runtime. This especially helps in larger networks, as is usually the case. All Brian functionality supports both the Numpy and Cython runtime code generation.

The standalone code generation generates a C++ program to run the simulation. By default, this hardcodes all the input spikes into the binaries, making them relatively large. The simulation is run outside of the Python environment, after which the results are imported back into Python. All the resulting spikes are stored on disk or memory, a bottleneck for larger outputs.

Since all inputs loaded at the start, there are a few changes compared to running in the Python runtime.

- A different clock is required to change the input rates every 50 milliseconds.
- Since the network state can no longer be stored/restored. The network state needs

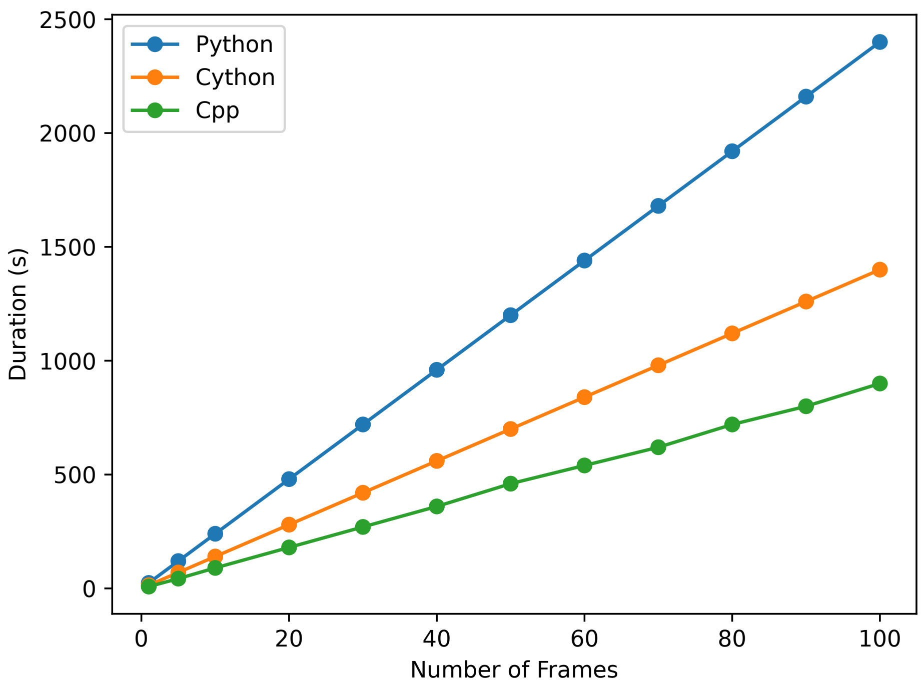


Figure 4.11: The Brian simulation times of a relatively large (128x128) network. The C++ simulation has a slight advantage over Python with Cython optimised code, but the compile-time makes it unideal for test runs. Using pure Python (using Numpy) is the slowest.

to be reset periodically.

- The output spikes need to be transformed back into 50ms frames.

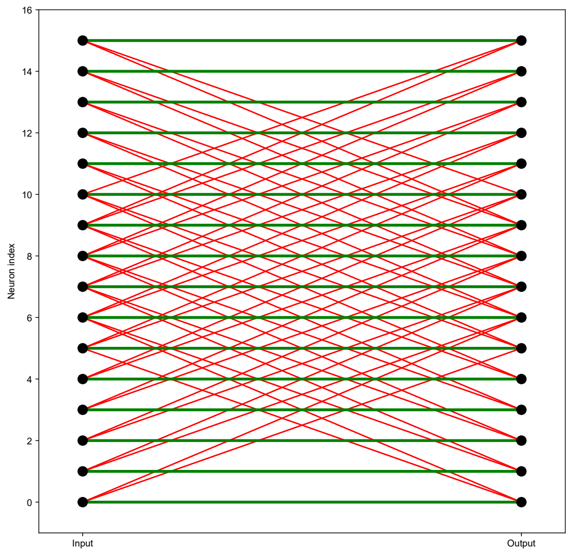
The standalone version results in about a 1.5-2 times speedup over the Cython version and 2-3 times over pure Python (figure 4.11), depending on network size. Larger networks benefit more from optimisation as more operations can be run in parallel[11]. The standalone code generation does require additional compile-time, making it inefficient for quick tries and debugging. In addition to C++, Brian also supports compilation to GPU-enhanced Neuronal Networks (GeNN). While the networks in this thesis have been implemented with compatibility for GeNN, it was not used. With large networks, this could further improve performance. It should be noted that since each time step is dependant on the previous, the simulation performance improvement is bound on the operations performed in each time step.

4.4 Network Architecture

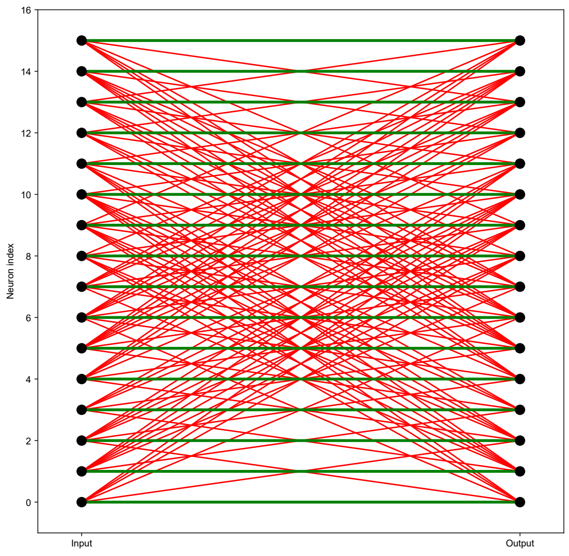
As discussed in chapter 2, the CFAR algorithm is used to estimate the background noise and threshold whether there is a target or not accordingly. Because it is an estimation and not an exact value, a factor α is used to correct for a given distribution and P_{FA}. This α is used as a weight modifier throughout the network and is dependant on the number of connected synapses. The network consists of two neuron groups: a

Table 4.1: The properties used in the SNN.

T_{frame}	T_{ref}	dt	R	C	τ	w_0	u_0	u_{static}
50ms	10μs	1μs	5MΩ	10nF	50ms	0.2V	0V	0 - 1V



(a) 1D Network



(b) 2D Network

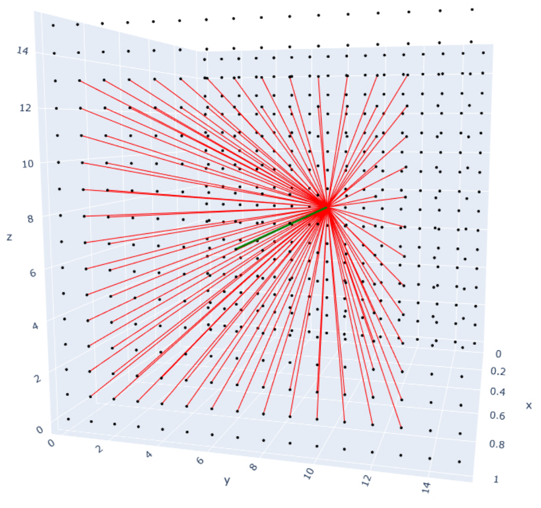
Figure 4.12: Examples of the network topologies used. The green edges are excitatory, the red edges inhibitory. In both examples, there is one guard and one training neuron on each side.

Poisson input group and an IF or LIF neuron group. As these neuron groups are 1D, a 2D input needs to be flattened. There are two groups of synapses: excitatory and inhibitory. The excitatory synapses are connected trivially with a $i = j$ condition, with i being the pre-synaptic neuron and j the post-synaptic neuron. The inhibitory synapses connect the j-th output neuron to its reference input neurons. All the properties used in the simulation can be found in table 4.1; the values will be explained in this section.

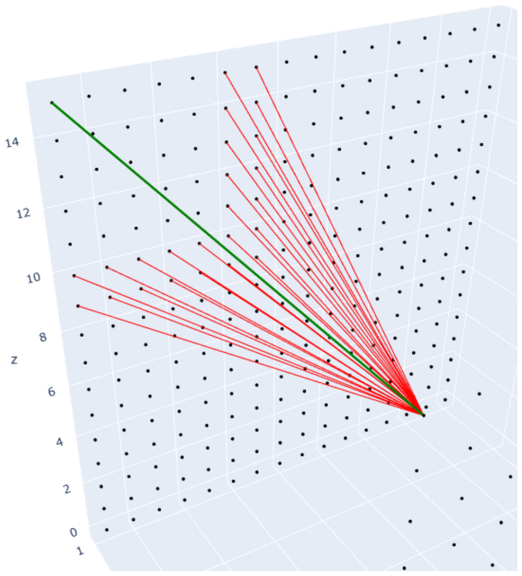
The T_{frame} is the time each radar frame is exposed to the network, T_{ref} is the refractory period of the output neurons, dt is the simulation time step, $RC = \tau$ are the resistance and capacitance of the neuron resistor and capacitor respectively and determine the time-constant τ. w_0 is the base-weight, and u_0 is the resting membrane potential. Finally, u_{static} is the static threshold that determines together with the base-weight how many input spikes are required for an output spike. Together with the leaky behaviour, the static threshold facilitates a high-pass filter.

4.4.1 Network Connectivity

The connectivity in the network is used to have a dynamic threshold using inhibitory spikes. To achieve the same behaviour as normal CFAR, the input neurons are connected to the output neuron based on the reference cells the standard CFAR algorithm



(a)



(b)

Figure 4.13: The 2D network connectivity is plotted in 3D, with two training neurons and four reference neurons on each side. a) shows the middle (green) excitatory synapse and (red) inhibitory synapses when all connections are possible. b) shows the connections on the edge; not all inhibitory synapses are available, and this output neuron will have a higher loss.

would use to compare with the CUT: the reference neurons are connected to the output neuron of the CUT (figure 4.12).

Because each neuron group is one-dimensional, a 2D image needs to be flattened to 1D. This is done with row-major to simplify computation, as the array is also stored in row-major in memory. The two 2D indices are then translated to their corresponding 1D index. All synapses connecting to out of bounds neurons on the edges are abandoned.

The number of synapses depends on the input size, number of guard neurons and number of reference neurons. For example, Let $I_{G,R}$ be the number of synapses for an $N \times N, N = 64$ range-doppler image and $G = 4$ guard neurons on each side, $R = 2$ reference neurons on each side. Each output neuron has one excitatory synapse connected (the CUT). This results in approximately $I_{4,2} < 4N^2R(2G+R+1) = 360448$ synapses, not counting unconnected unconnected neurons at edges.

Figure 4.13 shows a 3D representation of the connectivity. There will be fewer connections when the CUT is at an edge, and the correction factor (α) compensates for this. This does, however, make both the conventional and the spiking CFAR susceptible for missing detections (see CFAR loss figure 2.7) at the edges.

4.4.2 Network Weights and Threshold

Neural networks usually have an implied learning mechanism in the form of weights. As these weights change, the network will attempt to conform to the desired output. This network does not have such a learning mechanism. Instead, the weights are fixed

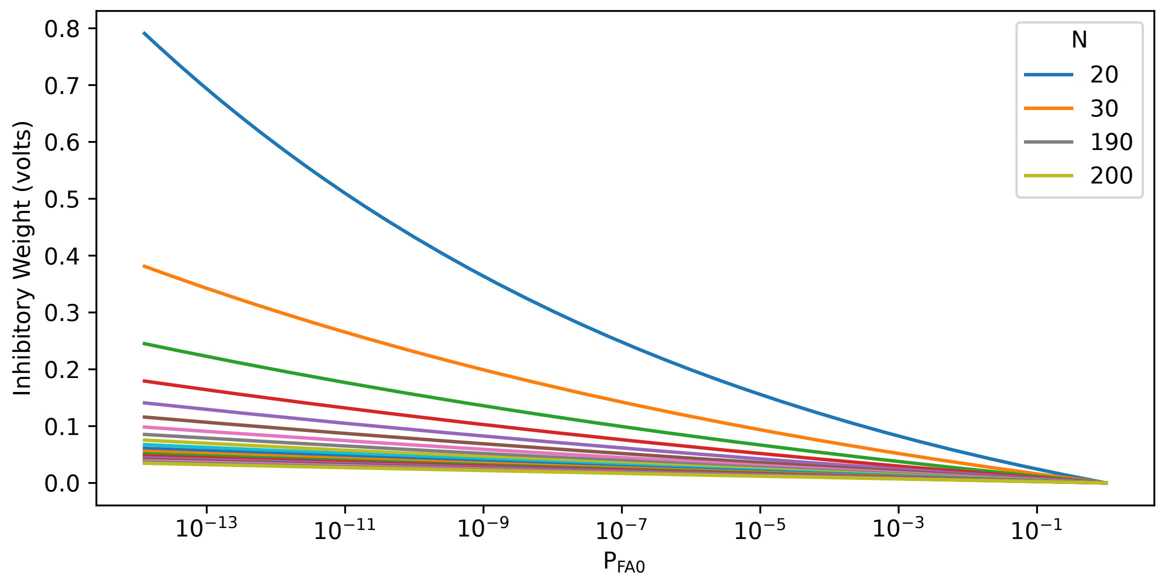


Figure 4.14: The voltage of the inhibitory weight, plotted per P_{FA0} and N. Each line represents a different number of reference neurons N. As P_{FA0} is lowered, each reference neuron will have a more considerable influence.

on some base weight w_0 and multiplied by α. The α is dependant in the number of connected reference neurons and the desired probability of false alarm P_{FA0}: $\alpha_j = N(P_{\mathrm{FA0}}^{-1/N} - 1)$. As discussed in chapter 2, α is a correction factor for the approximation of reference noise. The lower P_{FA0} gets, the more influence each reference neuron gets. If P_{FA0} is 1, all inhibitory weights become 0, and there is no dynamic thresholding. If P_{FA0} approaches 0, all inhibitory weights go to infinity, and all output neurons will be inhibited (equation (2.10)).

The base weight w_0 determines together with the threshold how many spikes are required for an output spike, i.e. detection. The base weight is used in both the inhibitory and excitatory synapses such that each spike is equivalent. w_0 is divided by the number of reference neurons and multiplied by an α for the inhibitory neurons. For example, for a Spiking CFAR network with 44 connected reference neurons (4 guard and 2 reference neurons on each side), a w_0 of 200 mV and P_{FA0} of 10^{-3}, each reference neuron will subtract approximately $(10^{3/44} - 1) * 200 \approx 34\mathrm{mV}$ per incoming inhibitory spike. In other words, for each excitatory spike, about six inhibitory spikes are required to prevent an output spike. Equation (4.4) is equation (4.3a) modified with α. $S_i[t]$ is 1 if a spike occurred at time t in input neuron i. Instead of having a different weight for each synapse, all inhibitory synapses connected to the same output neuron have the same weight. The weight for the excitatory $j = i$ synapses is fixed on w_0.

$$u_j[t] = u_j[t - \Delta t] + w_0 S_j[t - \Delta t] - \frac{1}{N_j} \sum_i \alpha_j w_0 S_i[t - \Delta t] \tag{4.4}$$

The threshold u_{static} in each neuron is the same and set on a fixed value. It can be set on anything, but 1 volt is a convenient unit. The weight w_0 is set on the value of 200mV, empirically found by looking at the similarity between the original and spiking CFAR algorithm, calibrating the spiking implementation to be as equal as possible. More importantly, the ratio between the threshold and weight is 0.2, i.e. each output neuron will require at least six spikes before an output spike is generated. Theoretically, a weight threshold ratio of 1 would produce the same values as the standard CFAR algorithm. However, the temporal aspect and introduced stochasticity require the network to have multiple input spikes before generating an output spike. This additional threshold acts as a high-pass filter, and while this will cause low-power targets to be missed, it also filters out low-power noise. This also makes the network robust to the stochasticity either introduced by hardware imperfections or the Poisson group.

4.4.3 Time Constant and Simulation Time Step

The time constant τ of the LIF is determined by the capacitance and resistance of the resistor and capacitor, respectively. The charge inside the capacitor decays exponentially in about 5τ. After 1τ, only 37% of the charge remains. With a frame time and time constant of 50ms, the network uses the noise estimation of the previous five frames when the network is not reset after every frame.

The chosen simulation time step depends on the range of the input values. For example, with a time step of 1ms will only be able to spike at a maximum of 1000Hz. With a frame time of 50ms, this would mean each value has to be encoded in $1000 * 0.05 = 50$ spikes. The quantisation from real-valued inputs to spike rates causes some error depending on the maximum spike rate. Since the time step constrains the maximum spike rate, the ideal network would have an infinitely low time step. Changing the time step is a trade-off between accuracy and simulation time. The simulation rate $1/dt$ is always at least twice the maximum input value to avoid clipping (also see figure 4.6).

4.4.4 Recurrent Output Layer

Normally during non-coherent integration, both the noise and target signals are integrated. Because of the neuron reset behaviour, this does fully not occur in SNNs: when there is a non-zero (static) threshold u_{static}, the membrane potential is reset to 0V, meaning there is a loss of the same amount. To counteract this, an additional network design with recurrent connections is introduced (figure 4.15).

These connections divide the charge required for a spike (the threshold voltage of a neuron) over each guard neuron and itself. While this does not precisely replicate non-coherent integration due to the leakage, it compensates for part of the loss and makes the guard neurons more sensitive for detections. Note that while for an individual neuron, this might seem insignificant, this effect occurs for all the output neurons. Because the neuron goes into a refractory period immediately after the spike, the connection has a random delay ranging from 1-2 times the refractory period.

For 0V thresholds (or any other configuration in which the reset voltage is the same as the threshold voltage), this concept makes no sense as there is no loss. All changes below the threshold are kept through the frames as usual.

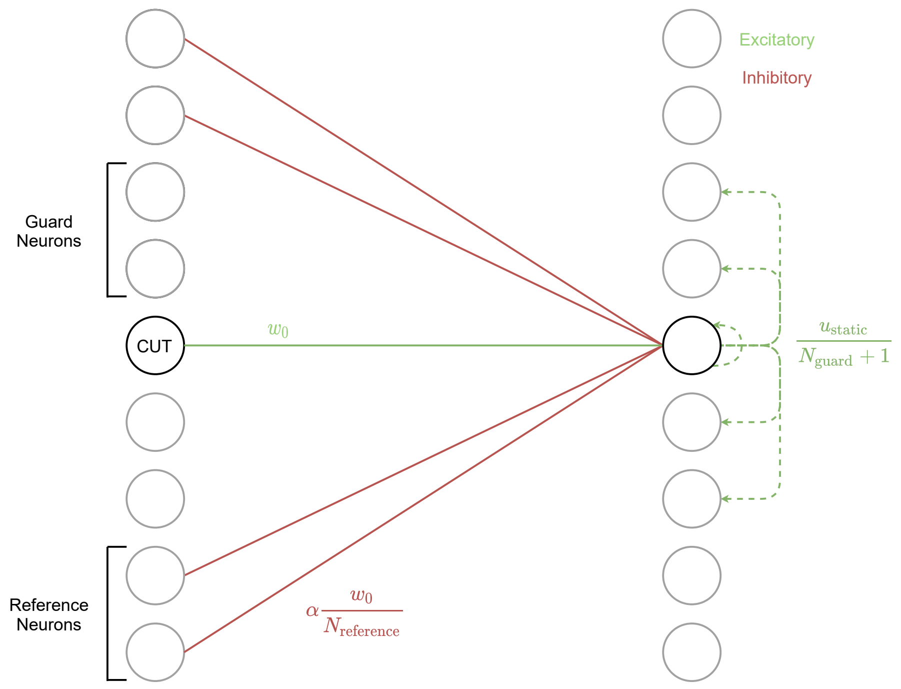

Excitatory
Inhibitory
Guard Neurons
CUT
w_0
$\frac{u_{\text{static}}}{N_{\text{guard}}+1}$
Reference Neurons
$\alpha\frac{w_0}{N_{\text{reference}}}$

Figure 4.15: A 1D example of a recurrent spiking CFAR network. The network design is the same as the regular spiking CFAR design, except for the recurrent excitatory synapse connections at the output layer on the right. These recurrent connections compensate for the energy loss by the neuron reset for the subsequent frames.

While previously introduced concepts are modelled after existing concepts, this is a novel change of which the performance will be evaluated separately.

4.5 Metrics and Dataset Generation

Most results in this thesis are based on the following metrics: the Receiver Operating Characteristic (ROC), an accuracy and similarity score. Because most freely available radar datasets do not have labelled data, the first results are based on the similarity of the binary output images processed by conventional and spiking algorithms.

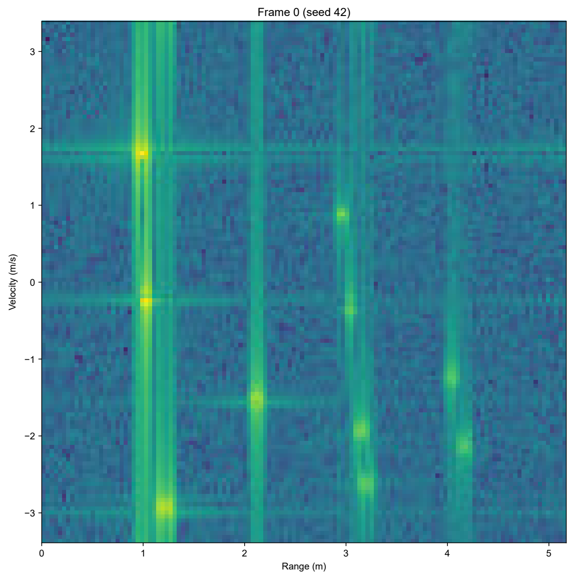
4.5.1 Radar Simulation

In order to have a measurable performance metric, the radar simulator Radarsimpy[1] is used. This Python simulator can simulate FMCW frames with the specifications of the RFBeam MR3003 radar (table 4.2) and any number of targets. While raytracing with 3D models is available, only the ideal point targets are used. The same data processing pipeline is used, except that the returned data is complex-valued I/Q instead of real-valued.

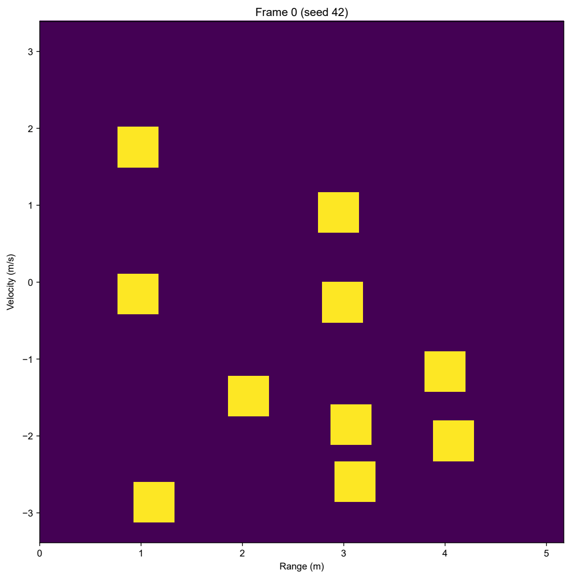
[1]https://github.com/rookiepeng/radarsimpy

Table 4.2: The Radar simulation settings, modelled after the RFBeam MR3003.

Transmitter					Receiver			
f_{min}	f_{max}	t_{ramp}	t_{chirp}	P_s	f_s	G_{rx}	G_{baseband}	R_{load}
GHz	GHz	μs	μs	dBm	MSps	dB	dB	Ω
77	81	24.6	139.975	16	5	26	15	50



(a) original



(b) reference

Figure 4.16: The reference image is generated based on the location and velocity given to the simulator and is used to calculate ROC performance. The results will use radar simulations with a maximum of four simultaneous targets.

Because the MR3003 radar outputs a frame every 50ms, an additional layer is added to recalculate each target's location per frame. This results in identical behaviour compared with the MR3003. Target generation has random values for each target's property (distance and velocity), and the used random seeds are documented to ensure replicability. The initial frame can use any position and velocity to spawn targets, new replaced targets in later frames are constrained to one edge with a random velocity towards the other edge, replicating realistic radar frames. To promote a larger variance in target characteristics, the minimum speed is set at 1.5m/s, which is humans' average preferred walking speed.

A set of target masks are generated for each frame and target. The size of each target mask is based on the number of guard neurons used, as this is also related to the expected target size (figure 4.16). This mask can be directly used to measure target detections, or the mask can be used to find the highest power pixels inside each mask, and these pixels can then be used to measure detections accurately.

4.5.2 Receiver Operating Characteristic

One of the primary ways to measure performance is by calculating the actual detection and false alarm rate and plotting this for each input value: the desired P_{FA0}. The resulting points can be connected and result in a curve indicating the performance, also called the ROC. The area under the curve is frequently used to quantify the performance of binary classifiers.

The detection rate is the number of detections divided by the total number of targets. The False Alarm Rate is the number of invalid detections divided by the total amount of negative values (equations (4.5) and (4.6)).

$$\text{Detection Rate} = \frac{\text{Detections}}{\text{Detections} + \text{False Negatives}} \tag{4.5}$$

$$\text{False Alarm Rate} = \frac{\text{False Alarms}}{\text{False Alarms} + \text{True Negatives}} \tag{4.6}$$

The detections are measured by using the highest power pixel located in the generated target mask. Since the number of targets is known, the detection rate can be calculated trivially. The false alarm rate uses each pixel value, except those behind the target mask; these are all the pixels supposed to be zero. These two rates can then be plotted for both the standard and spiking CFAR.

In addition, the average noise is calculated by subtracting the noiseless reference frame (separately generated by the radar simulator) from the output frame in the complex domain and taking the average. With this exact noise information, the SNR can be calculated per target. By also considering whether a target was detected or not, the detection rate can be calculated per target and its measured SNR, resulting in the detection rate ROC.

The false alarm rate ROC is calculated over the average Signal-to-Noise Ratio (SNR) of the whole frame. This means that while the detection rate ROC plot is per target and has 400 samples per dataset, the ROC P_{FA} plot only has one data point per frame for a total of 100 per typical dataset.

4.5.3 Similarity and Accuracy

The Jaccard Similarity Index or Jaccard score is used on a pixel level and acts as a binary classification to determine how similar the conventional and spiking results are. To calculate similarity, the standard CFAR algorithm is used as ground truth, the Jaccard score is the number of correct pixels (M_{11}: both are 1) divided by the number of incorrect and correct pixels ($M_{01} + M_{10} + M_{11}$): equation (4.7).

$$\text{Jaccard score} = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} \tag{4.7}$$

The similarity is not a metric for the performance of the CFAR algorithm but rather a metric to measure how matching the standard and spiking algorithms are. This can be interesting when the CFAR algorithm performs well on specific unlabeled datasets. In addition, the Jaccard score is a quantifiable way to show the similarity while also

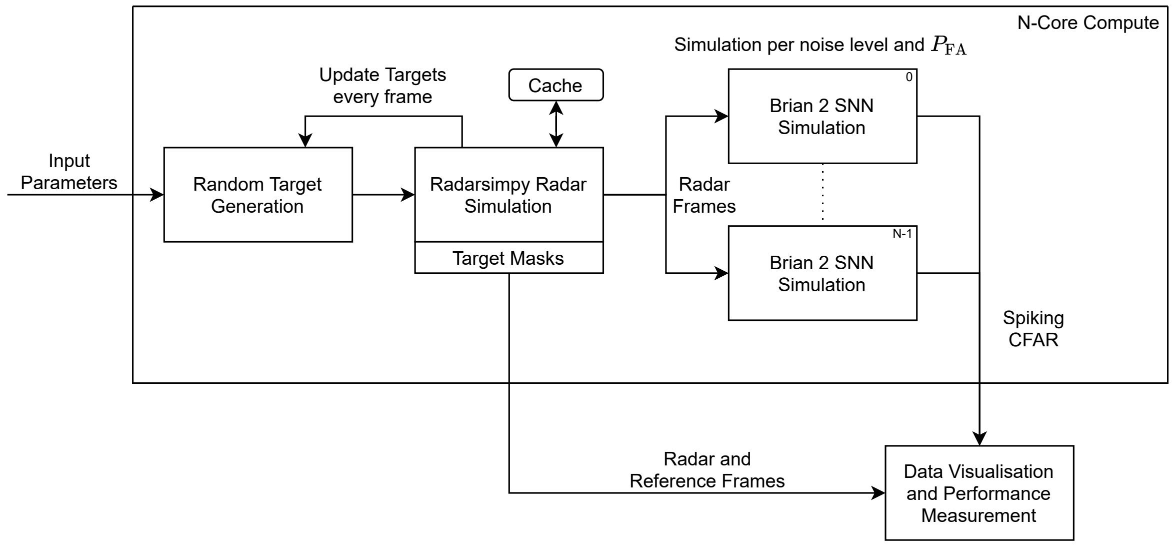

N-Core Compute
Simulation per noise level and P_{FA}
Update Targets every frame
Cache
Input Parameters
Random Target Generation
Radarsimpy Radar Simulation
Target Masks
Radar Frames
Brian 2 SNN Simulation
0
N-1
Brian 2 SNN Simulation
Spiking CFAR
Radar and Reference Frames
Data Visualisation and Performance Measurement

Figure 4.17: The simulation infrastructure is designed for running as many simulations in parallel as possible. The data processing can either be done on the compute instance or locally on a workstation.

looking at the other metrics. Because of the stochasticity and spike rate quantisation, the two algorithms are never expected to be 100% equal.

In addition, the Jaccard score can be used to calculate the accuracy. In this case, the highest power pixels are used as ground truth. The accuracy will be the sensitivity of the CFAR detector, and this metric can be used to say something about the algorithm's performance.

4.5.4 Simulation Infrastructure

In order to efficiently generate and simulate datasets, the use of Radarsimpy (Radar simulator) and Brian 2 (SNN simulator) has been combined into a parallelised structure such that many simulations can run simultaneously on each computing core (figure 4.17). For this thesis, over 1700 simulations have been run spread over 81 compute instances. The time it takes to run a single simulation ranges from 30 minutes to 2 hours, depending on the number of radar frames. Each instance has 32 CPU cores and up to 256GB of memory, depending on the simulation run.

Before starting a batch of SNN simulations, the radar frames are generated and saved to a cache on the SSD. The radar frames are then passed to each Brain 2 simulation instance, and each instance runs separately and writes the unprocessed output to its data files. After each run, all the data is downloaded for processing. The radar and reference frames are also saved into the data files.

Several other images are generated to evaluate performance during processing, including an SNR map to evaluate the SNR performance per target.

Results 5

This chapter documents all the significant results. The results consist of four meaningful metrics: the similarity between the original and spiking CFAR, the accuracy of both, the measured probability of detection (P_{D}) and the probability of false alarm (P_{FA}), both part of the ROC. Two datasets will be considered: a real-world dataset of spinning or rotating objects (unlabelled) and radar simulations with random targets (labelled). The real-world dataset is limited to visual inspection and the similarity metric, and the radar simulation allows for an analysis of the metrics.

The first section shows both the real-world and radar simulation results for the spiking CFAR implementation. The second section looks at the results of the novel recurrent spiking CFAR implementation. While this chapter might already discuss some results here, the main discussion will take place in chapter 6.

5.1 Spiking CFAR

This section shows all the results from the Spiking CFAR implementation. The results are collected for two datasets, one unlabelled real-world dataset and a labelled simulated radar dataset.

5.1.1 Real-World Datasets

The real-world datasets contain three scenarios: two from a rotating fidget spinner (horizontal and vertical) and aluminium plates rotating on an arm. These datasets are collected with the aforementioned RFBeam MR3003 radar, on the 5m maximum range and 25km/h maximum speed settings. Both the conventional and spiking CFAR use four guard cells/neurons and two training cells/neurons per side, for a total of 88 reference cells. As this data is unlabelled, the performance of Spiking CFAR can only be evaluated by looking at the similarity between the conventional and spiking algorithms. The similarity is all the correctly classified pixels divided by the correctly and incorrectly detected target pixels. The average and best results of the P_{FA0} can be found in table 5.1, all results can be found in the appendix (table A.1).

In figure 5.1 all the results are plotted. Notable is that the highest P_{FA0} has a wide variety of similarities. By definition, a probability of a false alarm of 1 means that a false alarm should always occur. This does happen when there is no network reset after each frame and no static threshold (0V static threshold), but not when the static threshold or the integration over time lower the false alarm rate. The runs are most similar between 10^{-4} and 10^{-6}, and get lower as the influence of the noise estimation increases (with lower P_{FA0}). The resulting images are not identical; this may be caused by the stochasticity and the lost energy potential after a spike caused by the neuron

Table 5.1: A list of the best performing P_{FA0} and averages of the RFBeam real-world datasets.

				Similarity	
Dataset	Model	Reset	u_{static}	Avg	Best P_{FA0}
Fidget 1	LIF	✗	0V	66%	76%
		✗	1V	59%	81%
		✓	0V	70%	86%
		✓	1V	67%	89%
	IF	✗	0V	38%	75%
		✗	1V	34%	48%
		✓	0V	68%	89%
		✓	1V	68%	89%
Fidget 2	LIF	✗	0V	64%	77%
		✗	1V	57%	76%
		✓	0V	69%	83%
		✓	1V	67%	87%
	IF	✗	0V	36%	76%
		✗	1V	31%	45%
		✓	0V	68%	86%
		✓	1V	68%	87%
Meccano	LIF	✗	0V	65%	81%
		✗	1V	55%	71%
		✓	0V	66%	76%
		✓	1V	62%	81%
	IF	✗	0V	50%	81%
		✗	1V	45%	55%
		✓	0V	64%	80%
		✓	1V	64%	80%

reset.

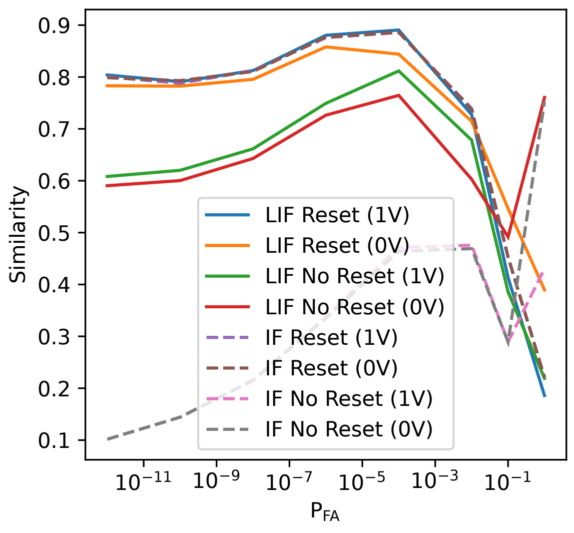
The drop-off at lower desired false alarm probabilities can be explained because the frames will generally have fewer detections, resulting in a bigger influence of (random) false alarms. E.g. with a $P_{\mathrm{FA0}} = 10^{-6}$, about 34 pixels are detected as a target, with a $P_{\mathrm{FA0}} = 10^{-12}$ this dropped to 23, making each false alarm more significant. The Meccano dataset seems to suffer most from this.

The LIF neuron with reset seems to produce the most similar results, with the IF variant nearly identical. This is further analysed by looking at the frames manually in chapter 6.

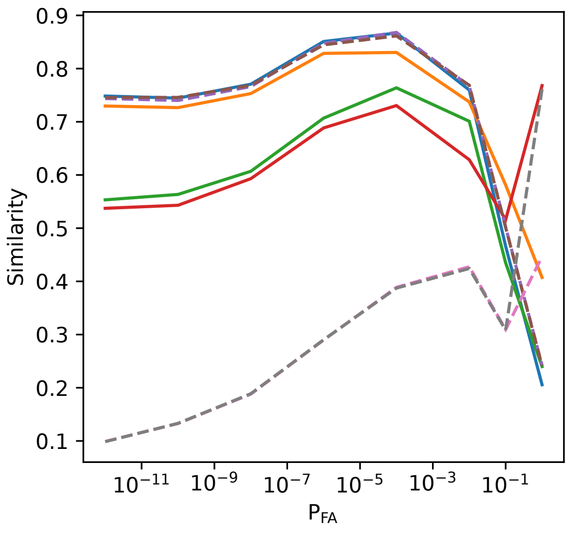
5.1.2 Radar Simulation Datasets

The simulation consists of four random targets generated from a random seed[1]. When a target leaves the frame, a new one will be spawned from one of the edges. The

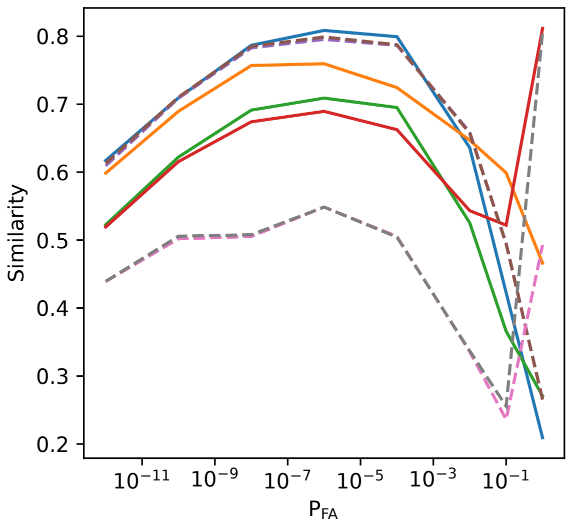
[1]All data collected in this thesis used seed *6470602006422716455*, some manual inspections also used *42*.



(a) fidget1



(b) fidget2



(c) Meccano

Figure 5.1: The similarity of each real-world dataset plotted per $P_{\mathrm{FA}0}$ input.

generated radar noise is random per run and will cause a slight variance in results. The frame time is 50ms, and for each run, 100 frames are generated for a total of 5 seconds. The most promising configuration will be run with a high-resolution configuration to improve the false alarm rate resolution, simulating 1000 radar frames for a total of 50 seconds. The minimum speed of targets is set at the preferred walking speed of humans (1.5m/s) to collect an as diverse amount of target samples. The conventional and spiking CFAR use four guard cells/neurons and four training cells/neurons on each side, for a total of 208 reference cells.

Over 500 simulations have been run to get the results with different noise figures and false alarm probabilities. To get an overview of the relative performance per

Table 5.2: A list of all the average similarities and accuracies from direct input simulation runs.

			Raw		Quantised		
Model	Reset	u_{static}	Similarity	CFAR	Similarity	CFAR	SCFAR
LIF	✗	0V	43%	40%	47%	32%	23%
	✗	1V	38%	40%	40%	32%	51%
	✓	0V	63%	40%	66%	32%	39%
	✓	1V	60%	40%	60%	32%	50%
IF	✗	0V	13%	40%	16%	32%	15%
	✗	1V	11%	40%	13%	32%	18%
	✓	0V	61%	40%	62%	32%	47%
	✓	1V	61%	40%	62%	32%	47%

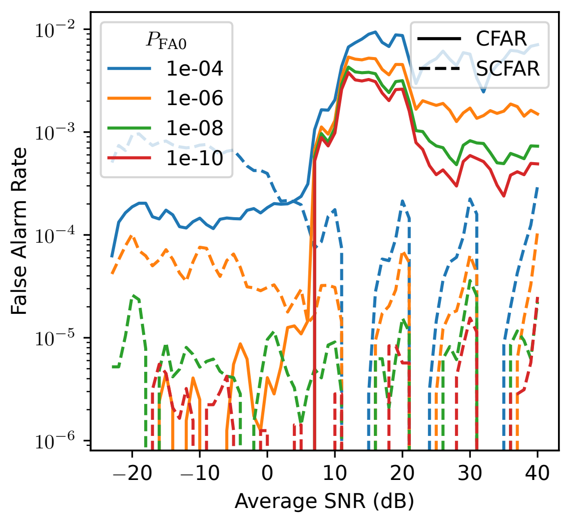
Table 5.3: A list of all the average similarities and accuracies from min-max normalised input simulation runs.

			Raw		Quantised		
Model	Reset	u_{static}	Similarity	CFAR	Similarity	CFAR	SCFAR
LIF	✗	0V	44%	40%	45%	32%	22%
	✗	1V	32%	40%	37%	33%	52%
	✓	0V	57%	40%	64%	32%	41%
	✓	1V	52%	40%	57%	32%	53%
IF	✗	0V	16%	40%	15%	33%	16%
	✗	1V	11%	40%	13%	32%	18%
	✓	0V	53%	40%	60%	33%	50%
	✓	1V	53%	40%	60%	33%	50%

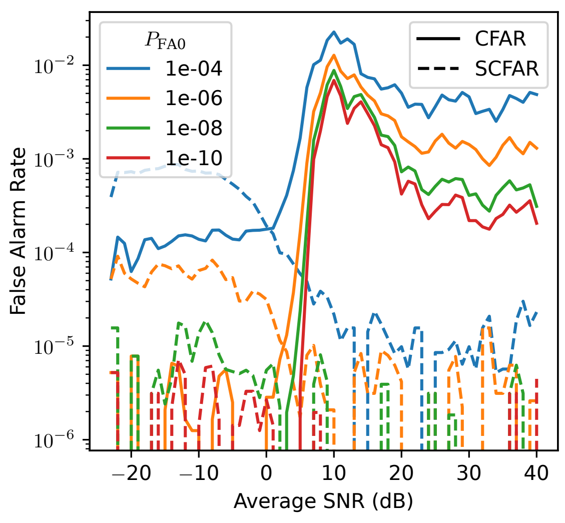
simulation, each simulation run is summarised by average similarity and accuracy. This is done for both direct and normalised input and can be found in tables 5.2 and 5.3 respectively. These results are meant as a relative indication of how well each version performs relative to the conventional CFAR algorithm. While, in essence, the same CFAR algorithm is run, a small percentage point difference can come from the radar noise, which is randomly seeded for each run. The quantisation influences the accuracy scores negatively (as shown in figure 4.3).

As low noise values are quantised to 0, they cannot be used to estimate noise reliably and, as such, increase the false alarm rate and thus lower the accuracy score (figure 5.2). The conventional CFAR algorithm is run both with and without quantisation to account for any performance gain. This way, the results are compared with the best scoring version to provide a reasonable frame of reference.

The average SNR used in the false alarm rate ROC is an average of the SNR of the whole image. Runs are done using varying noise figures to get a wide range of SNRs between runs. When the noise figure is too low (below 30dB), the quantisation error starts introducing a significant error and radar sidelobes start incorrectly being



(a) direct input



(b) normalised input

Figure 5.2: $u_{\mathrm{static}} = 1\mathrm{V}$ (LIF, reset). The quantised input is used to calculate the CFAR. Once the noise figure is low enough and the noise is squished to zero, the false alarm rate goes up significantly.

detected as targets. The highest SNR values are also plotted, but these effects should be considered while analysing the results.

A reset means that after each frame, the network is reset to its resting state. This replicates the behaviour of the conventional algorithm without any integration.

The accuracy score takes into account both detections and false alarms (see chapter 4). While it does not show the trade-offs, it does make overall performance measurable with a single metric. Figure 5.3 show the best performing configuration according to the accuracy score. However, as can be observed by looking at the P_{D} and P_{FA} ROC, it loses accuracy on detection rate. The false alarm rate, however, makes up for the lost accuracy on the detection rate. This trade-off is hard to quantify without looking at the ROC plots.

The ROCs of the lowest accuracy score is plotted in figure 5.4. It shows the performance of the IF model without a frame reset and static threshold. This does not work out as the IF neurons end up inhibiting all detections over time. This makes sense: the IF does not have a leaky component and will integrate over all the frames. Furthermore, since the neuron membrane potential resets to 0V after spiking, a net positive potential can not build-up to compensate.

In contrast, figure 5.6 shows the best performing detection rate configuration: the LIF model with frame reset and 1V static threshold. The detection rate is marginally better than the conventional algorithm. However, the false alarm rate is not constant: it starts higher and drops off, as also can be seen in the high-resolution[2] run (figure 5.5). Figure 5.7 shows the same configuration but with the IF neuron model. It shows almost identical detection accuracy and false alarm rates, and any difference falls within the

[2]High-resolution runs are run for ten times longer and take more than 24 hours in real-time.

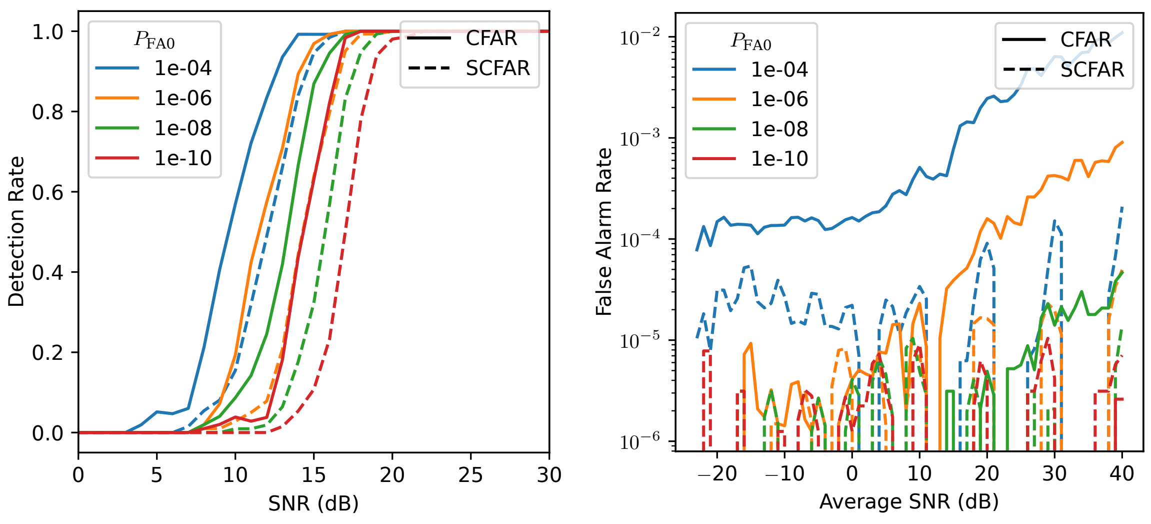
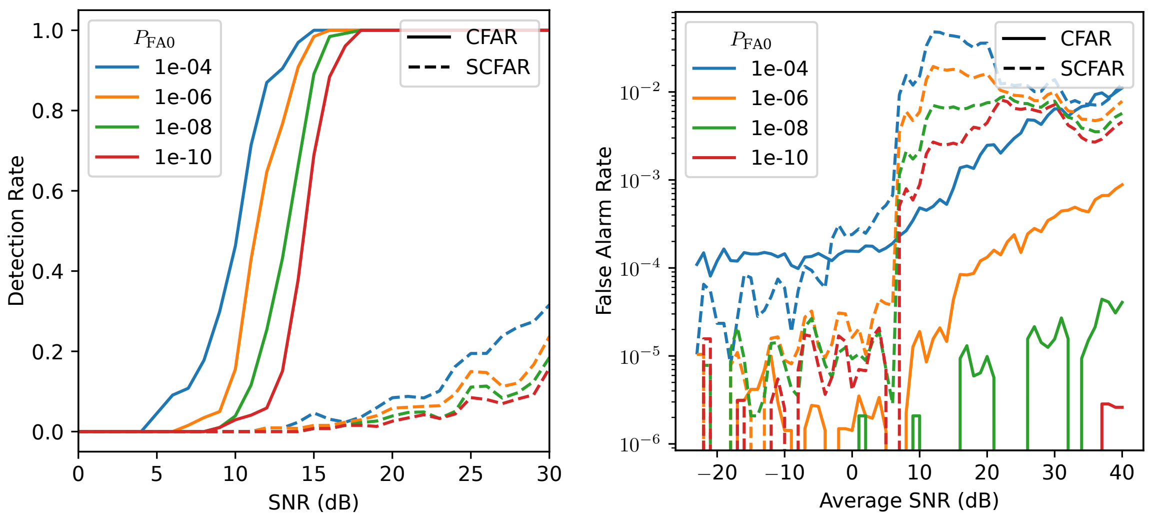


Figure 5.3: $u_{\text{static}} = 1\text{V}$ (LIF, no reset). According to the accuracy score, this is the best direct input configuration. While a higher SNR is required for detection, the false alarm rate is low enough to compensate.

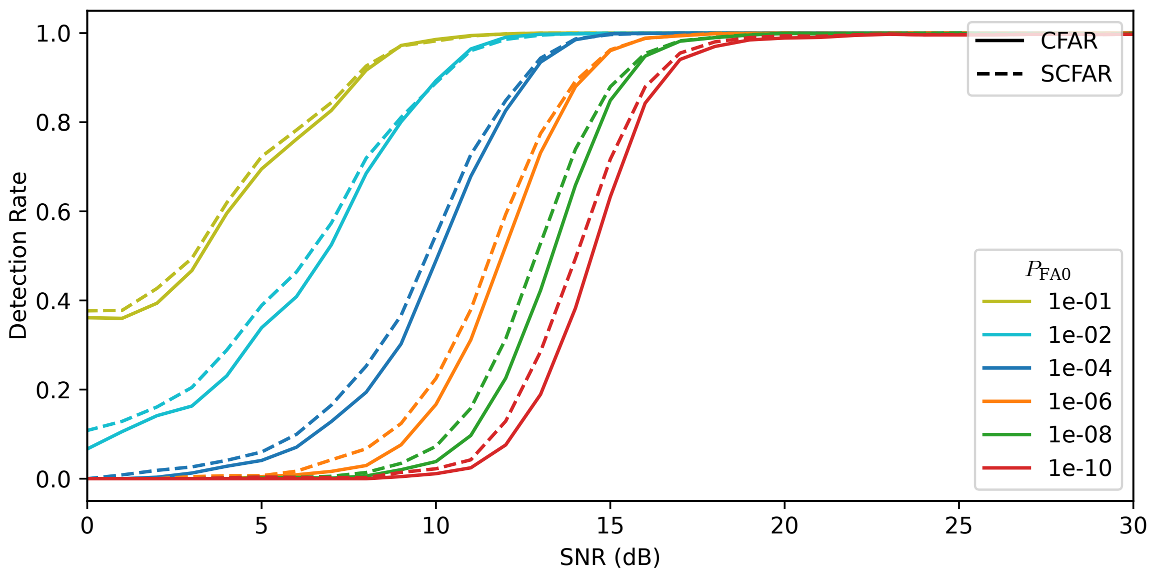


(a) PD ROC (b) P_{FA} ROC

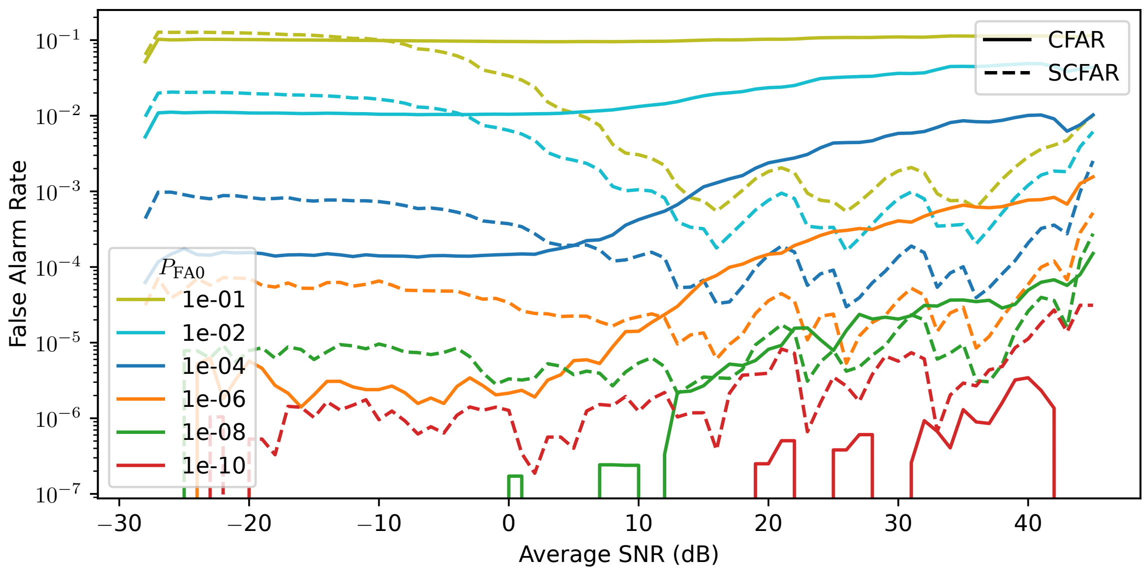
Figure 5.4: $u_{\text{static}} = 0\text{V}$ (IF, no reset). The detection rates and false alarm rates for the lowest accuracy direct input simulation run.

error margin.

In figure 5.8 the similarity and accuracy score per noise figure is shown. The similarity is lowest at the high desired false alarm rates. The false alarms are random and are the cause for a low similarity. Notable is the performance with the low and higher noise figures. The static threshold filters out false alarms that normally occur at high desired false alarm probabilities. Unfortunately, high desired false alarm rates are not as interesting such that this gain is mostly cosmetic. However, some of these gains



(a) P_{D} ROC

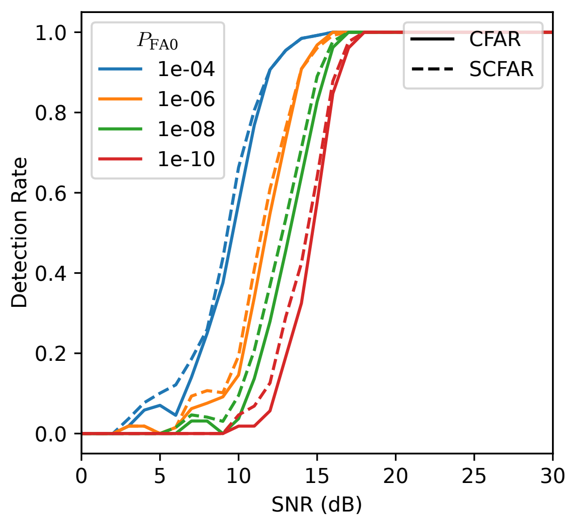


(b) P_{FA} ROC

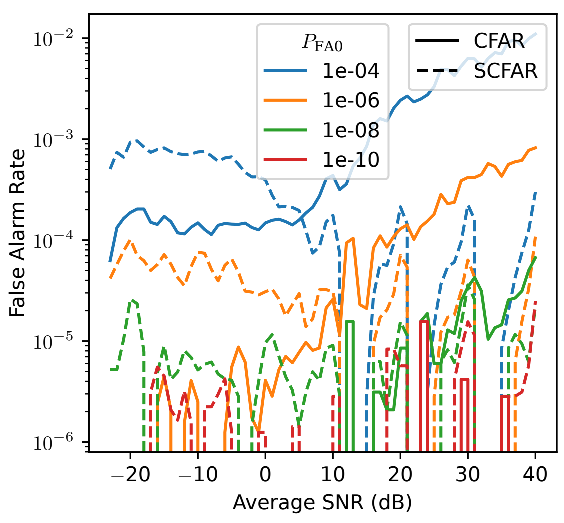
Figure 5.5: $u_{\text{static}} = 1\text{V}$ (LIF, reset). The ROCs of a high resolution (10x) run of the best performing direct input configuration, giving a more accurate representation of the performance.

carry over to the more realistic desired false alarm probabilities. At the low desired false alarm probabilities, the spiking algorithm performance is superior or at least similar. However, at the middle (10^{-4} to 10^{-8}), the high noise figures most likely cause false alarms.

All results outlined so far relate to the direct input, but the normalised spike rate appears to perform better based on accuracy. Because the spike rates are scaled from

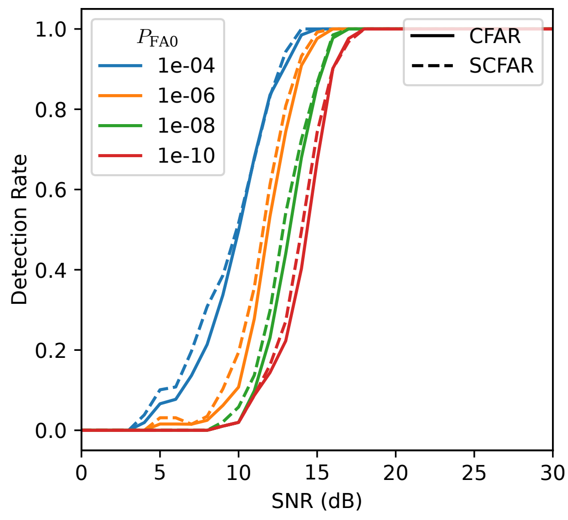


(a) P_{D} ROC

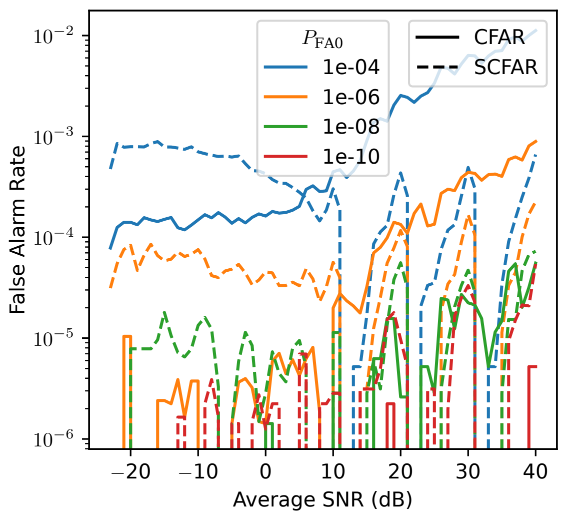


(b) P_{FA} ROC

Figure 5.6: $u_{\mathrm{static}} = 1\mathrm{V}$ (LIF, reset). The detection rates and false alarm rates for the highest detection rate direct input simulation runs.



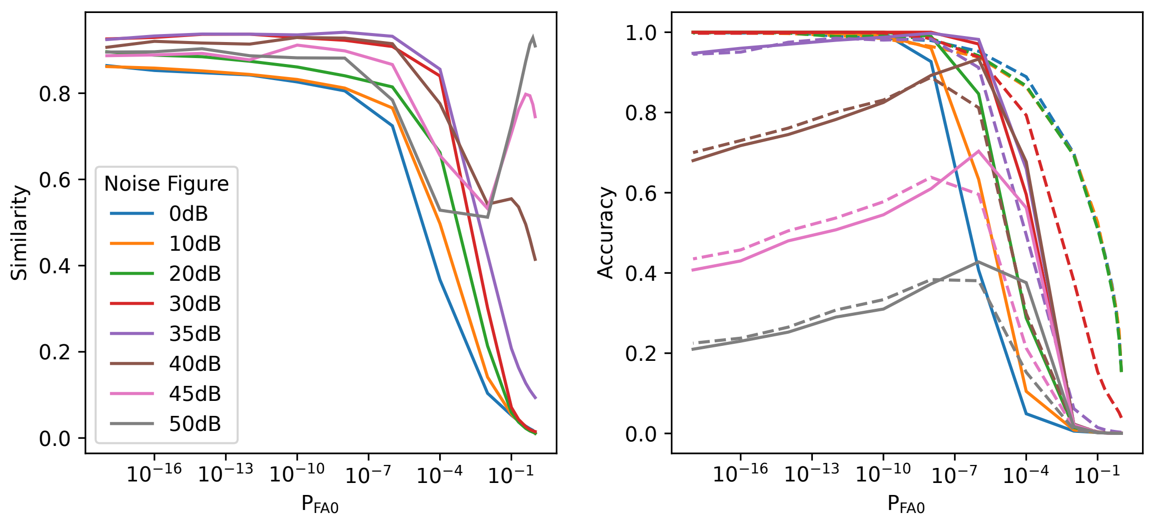
(a) P_{D} ROC



(b) P_{FA} ROC

Figure 5.7: $u_{\mathrm{static}} = 1\mathrm{V}$ (IF, reset). The detection rates and false alarm rates for the highest detection rate direct input simulation runs.

0 to 5000, this significantly influences the result. Figure 5.9 shows the comparison between direct and normalised, and as the average SNR increases (i.e. the noise floor is decreased), the false alarms get suppressed. While this is a potential research topic to find out the exact effects of using min-max normalisation to improve the results, it is considered out of scope for the remainder of this thesis.



(a) similarity (b) accuracy

Figure 5.8: $u_{\text{static}} = 1\text{V}$ (LIF, reset). The similarity and accuracies scores per added noise figure for direct input. Solid lines are conventional, dashed is spiking.

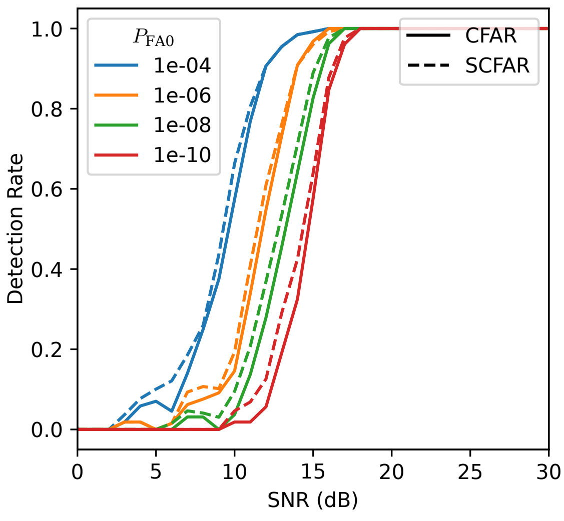
Table 5.4: A list of all the average similarities and accuracies from direct input recurrent spiking CFAR simulation runs.

			Raw		Quantised		
Model	Reset	u_{static}	Similarity	CFAR	Similarity	CFAR	SCFAR
LIF	✗	0V	43%	40%	47%	32%	23%
	✗	1V	39%	40%	39%	32%	36%
	✓	1V	47%	40%	47%	32%	37%

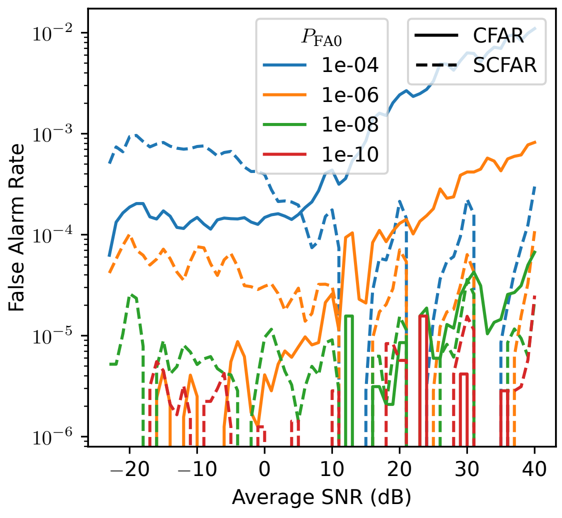
5.2 Recurrent Spiking CFAR

Only a few candidates from the direct input are simulated to evaluate the recurrent version of Spiking CFAR: the 1V static threshold LIF with and without a frame reset. The reasoning behind choosing these two candidates is that the reset behaviour has good performance overall. In contrast, the runs without resets have worse detection rates but a low false alarm rate, which may be improved. The 0V runs are not considered since they cause too many false alarms in their current form (also see chapter 6), adding the recurrent connections would only amplify this. The integrate-and-fire model suffers too much from the loss caused by resets to be considered for further optimisation.

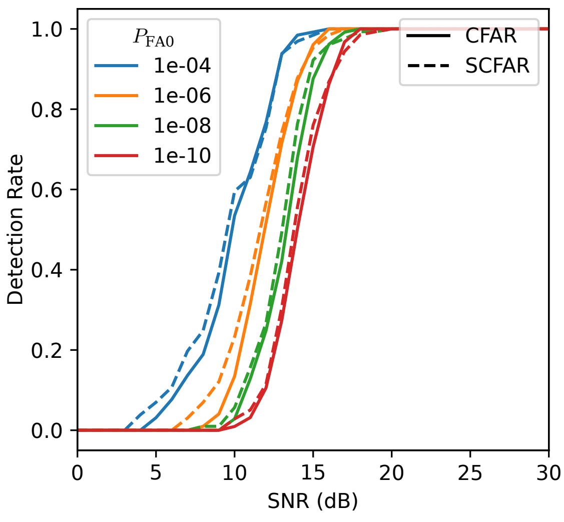
The results of the recurrent network can be found in table 5.4. The accuracy score is worse than its non-recurrent counterparts. When looking at the ROCs in figure 5.11, it becomes clear why. The false alarm rates are an order of magnitude higher compared to the non-recurrent networks. Possible reasons that this is the case is the recurrent connections amplifying noise, and the target becoming bigger than its mask and as such is counted as a false alarm.



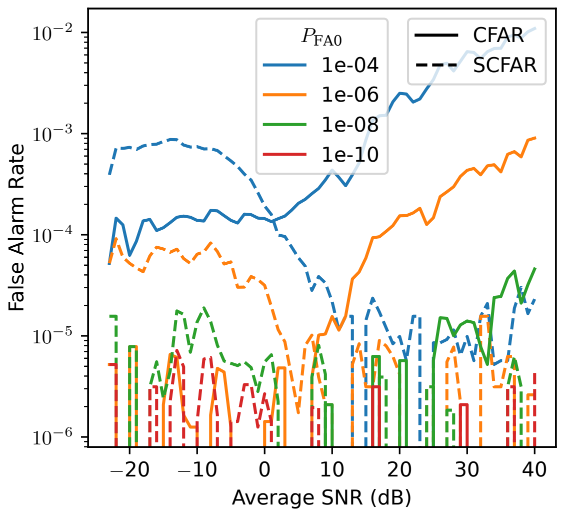
(a) P_{D} ROC direct



(b) P_{FA} ROC direct

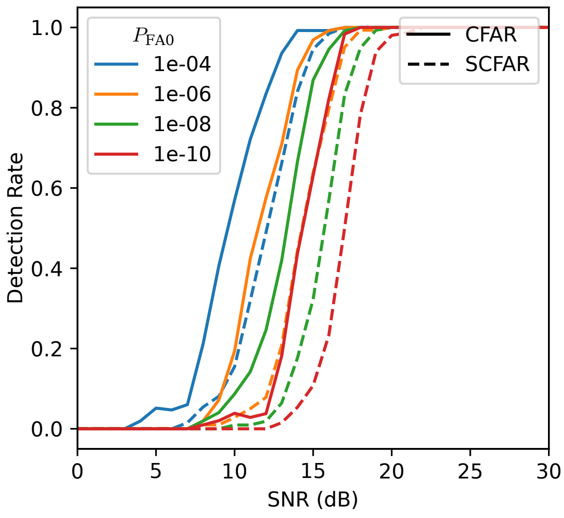


(c) P_{D} ROC normalised

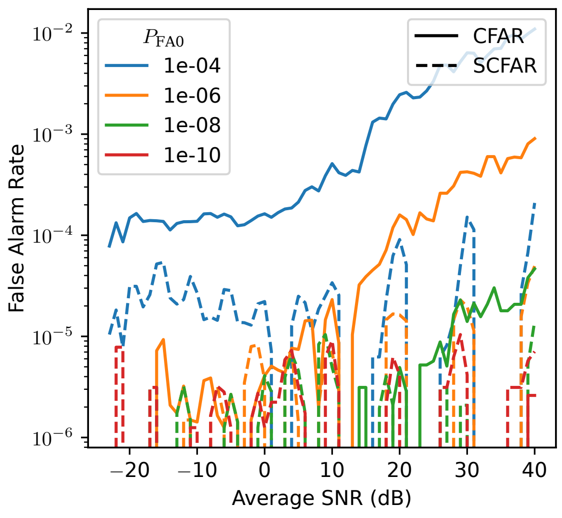


(d) P_{FA} ROC normalised

Figure 5.9: $u_{\text{static}} = 1\text{V}$ (LIF, reset). The detection and false alarm rates direct vs. normalisation.

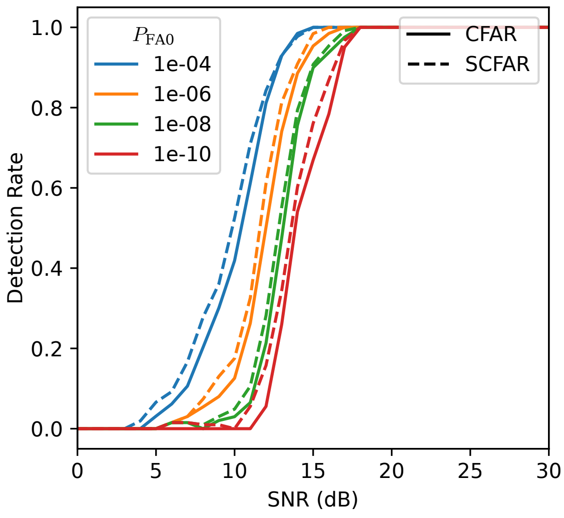


(a) P_D ROC

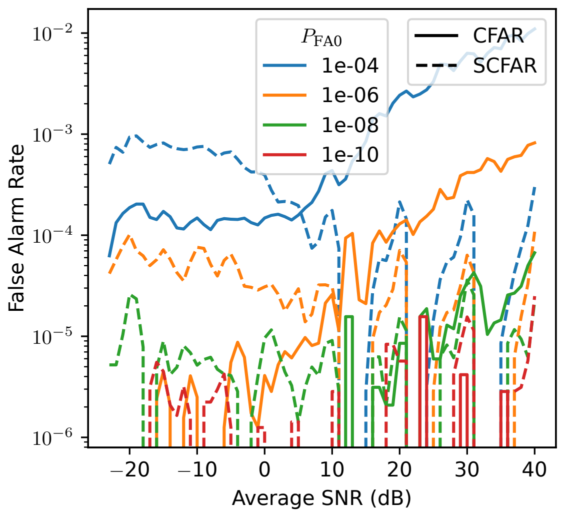


(b) P_{FA} ROC

Figure 5.10: $u_{\text{static}} = 1\text{V}$ (LIF, no reset). The results for the most accurate recurrent run without a frame reset. The false alarm rate starts lower but gets worse in the higher SNR ranges.



(a) P_D ROC



(b) P_{FA} ROC

Figure 5.11: $u_{\text{static}} = 1\text{V}$ (LIF, reset). The results for the most accurate recurrent run with a frame reset. By looking at the false alarm rates, it is clear why the accuracy is lower: the false alarm rate is an order of magnitude higher over the whole SNR range.

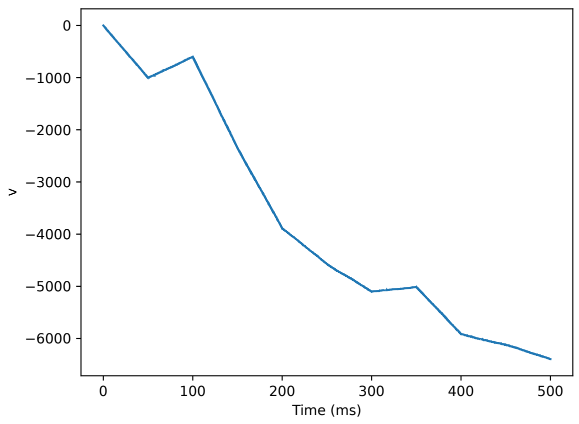
Discussion 6

This chapter analyses and discusses the results to determine the cause of specific outcomes. The first section discusses the results of the real-world dataset, and the second section discusses the results of the simulated dataset.

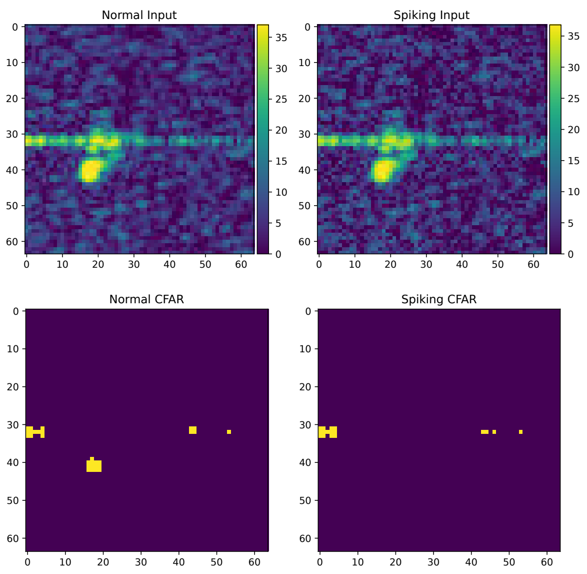
6.1 Real-world Dataset

This section discusses the results from the real-world dataset using the Spiking CA-Constant False Alarm Rate (CFAR). The unlabelled data limits the amount of objective analysis that can be performed. Apart from visual inspection, the similarity score merely tells how identical the produced results are, even if one version improves the other.

From visual inspections, it can be seen that the runs without a frame reset differ from the conventional CFAR algorithm: many detections are missed. This makes sense because the frames are integrated over time, and thus the noise can suppress output neurons over these frames. The main difference is that after reaching the threshold, the membrane potential gets reset to 0V, favouring the noise over the signal. The IF model

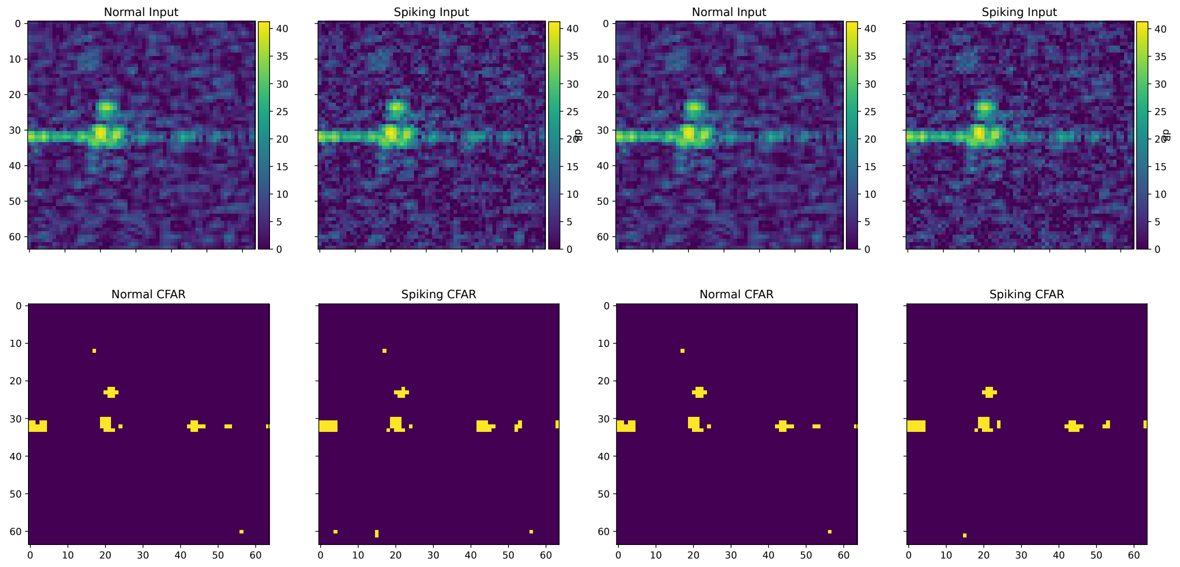


(a) membrane potential



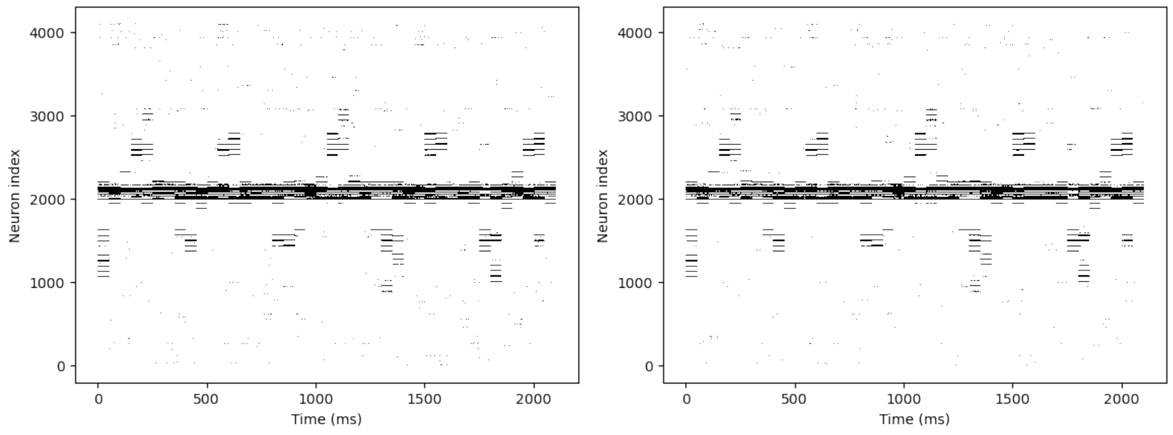
(b) frames

Figure 6.1: The membrane potential of an Integrate-and-Fire (IF) output neuron containing a low-power alternating target (manually picked). As the frames get integrated over time, the neuron gets significantly inhibited.



(a) $u_{\text{static}} = 1\text{V}$ (IF, reset) (b) $u_{\text{static}} = 1\text{V}$ (LIF, reset)

Figure 6.2: Frames of the real-world Meccano dataset using the IF and LIF model, respectively. $P_{\text{FA0}} = 10^{-4}$

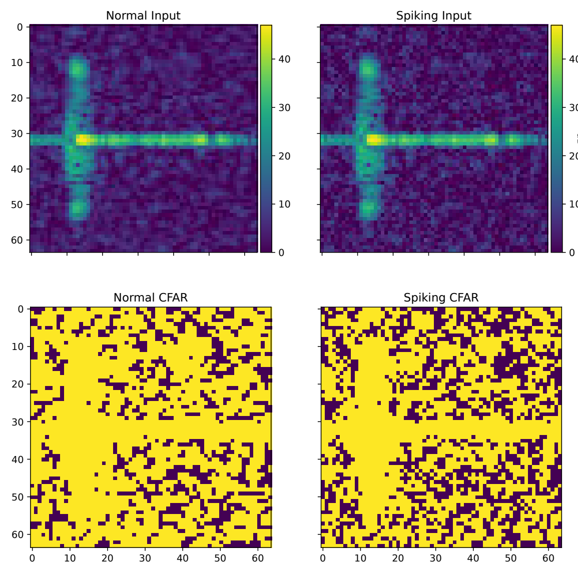


(a) $u_{\text{static}} = 1\text{V}$ (IF, reset) (b) $u_{\text{static}} = 1\text{V}$ (LIF, reset)

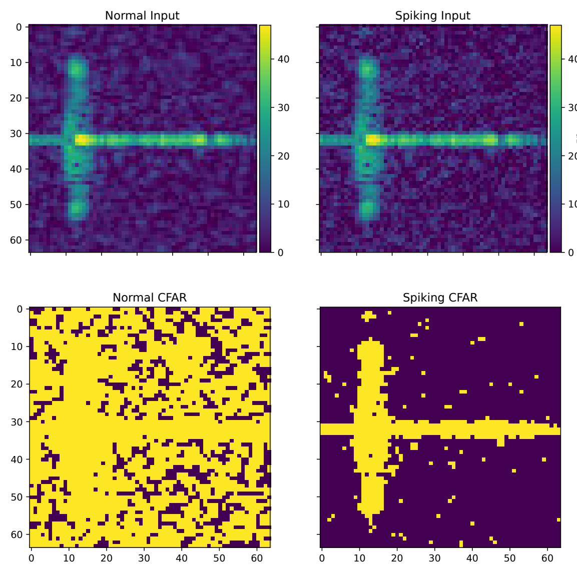
Figure 6.3: The spikes of the first 50 Meccano frames, including those shown in figure 6.2. Potential targets are somewhat easy to distinguish. The IF model has more false alarms compared to the LIF model. $P_{\text{FA0}} = 10^{-4}$

seems to suffer most from this: it is not able to build up a positive signal, and over time this causes the neuron membrane potential to be significantly inhibited (figure 6.1), especially at lower false alarm rates where the influence of each reference cell gets a more significant influence. This is because, contrary to the Leaky Integrate-and-Fire (LIF) model, the frames get integrated indefinitely.

Figures 6.2 and 6.3 show the difference between IF and LIF neurons that are reset after each frame (50ms). These are the best performing configurations according to



(a) $u_{\text{static}} = 0\text{V}$ (LIF, reset)



(b) $u_{\text{static}} = 1\text{V}$ (LIF, reset)

Figure 6.4: One of the first frames of the Fidget Spinner 1 real-world dataset. The input to the normal CFAR is quantised; all zero pixels are thus caused by the quantisation error, i.e. input values that are clipped to 0. $P_{\text{FA0}} = 1$

the similarity. The LIF neuron seems to be most similar round 10^{-6}. As seen in the frames and spike plots, the IF neurons seem to spike incorrectly more often. The only difference is the lack of leakage. A plausible explanation would be that the leakage lowers the chance of random spikes by filtering the low-frequency spike rates (i.e. spikes generated by low power cells). It must be noted that the conventional CFAR, even with the quantisation error, does not have these false alarms at all. This suggests that the LIF neuron only lowered the chance that more excitatory spikes arrived before the inhibitory spikes, even though the inhibitory rate might be higher.

Another interesting case is when the probability of a false alarm equals one (figure 6.4. By definition, this should always result in a false alarm. This is, however, not the case when there is a static threshold or the frames are integrated over time since the low-power noise gets filtered out before the thresholding (and spike) occurs. One could argue that this means that the spiking CFAR is, in fact, no longer a constant false alarm rate. Or, at the least, it no longer lines up with the given input probability of false alarm.

6.2 Simulated Dataset

In this section, the performance of the simulated dataset will be analysed. There are two questions to discuss: (1) Does the detection rate increase without increasing the false alarm rate? (2) Is the false alarm rate constant, or in other words, is the implemented algorithm a CFAR detector?

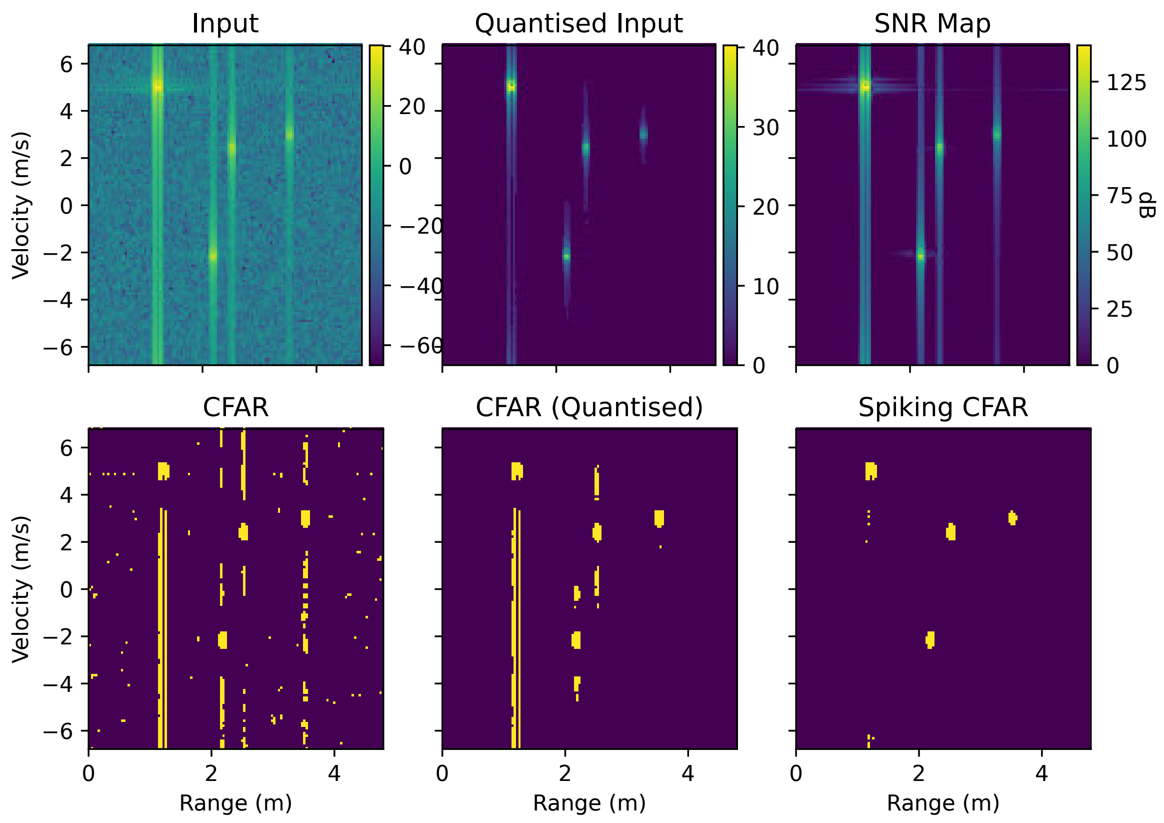
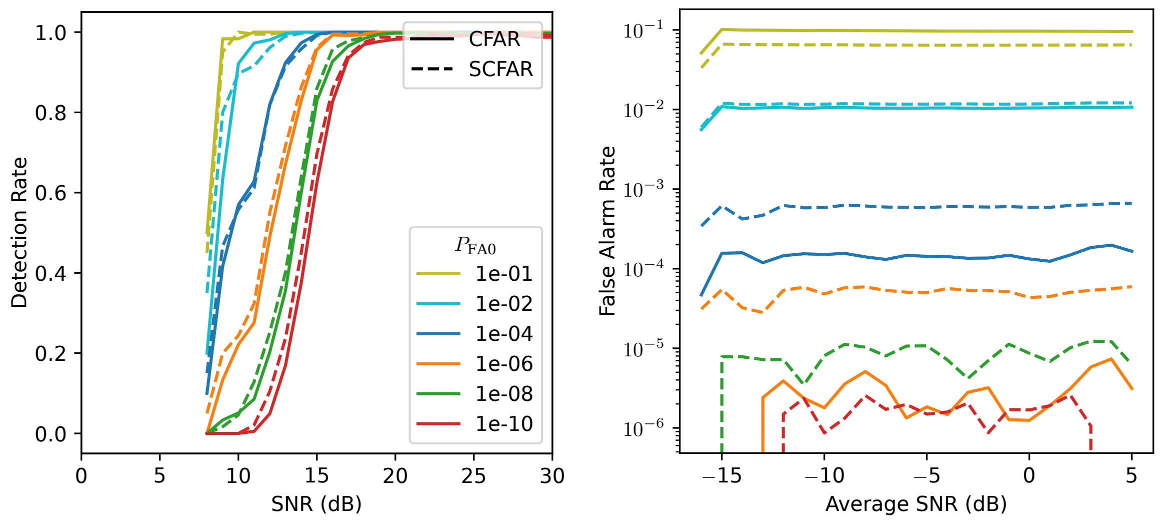


Figure 6.5: When the noise figure is below 20dB, the target sidelobes will be detected as targets. A noise figure this low is not usual when using real-world data. This is as an example with $P_{\mathrm{FA0}} = 10^{-2}$, but also occurs when using lower values.

6.2.1 Regular Network

The first question can be answered by looking at the high-resolution Receiver Operating Characteristics (ROCs) plots in figure 5.5. While the detection rate is higher than the conventional algorithm, the false alarm rate is also an order of magnitude higher. This result means that objectively, while the detection rate is higher, it comes at the cost of a higher false alarm rate. The integration over time (i.e. not resetting the network between frames) was expected to increase the Signal-to-Noise Ratio (SNR) and thus increase the detection rate without increasing the false alarm rate (or vice versa). This, however, is not the case. Because the membrane potential is reset after a spike, only the noise is integrated and weighs down the detection rate too much. This was also observed in the real-world datasets. In conclusion, does the Spiking CFAR not improve on the conventional CFAR algorithm in its current form.

The second question is a bit more difficult to answer using just the current results. The conventional and spiking CFAR does not seem to have a constant false alarm rate when looking at the ROC plots. Each simulation is run with a wide variety of noise figures to get a high range of SNR values. The highest SNR values are unrealistic and start being influenced by the quantisation error significantly. Also, as the reflection



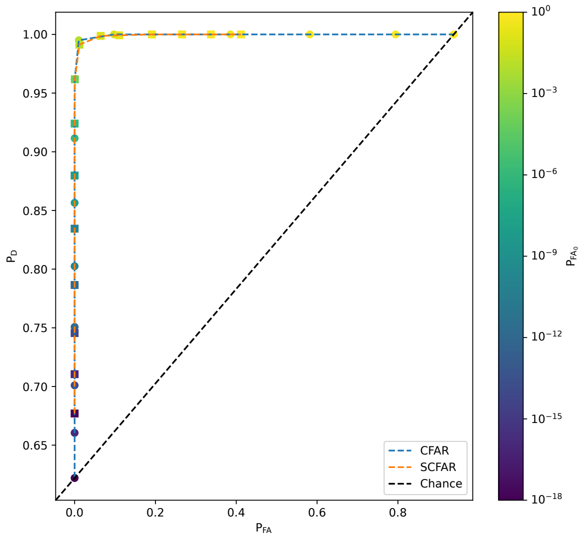
(a) P_{D} ROC (b) P_{FA} ROC

Figure 6.6: $u_{\text{static}} = 1\mathrm{V}$ (LIF, reset) The same run as before but with a constant noise figure of 40dB and a varying RCS. This reduces the SNR resolution but gives a more accurate false alarm rate plot.

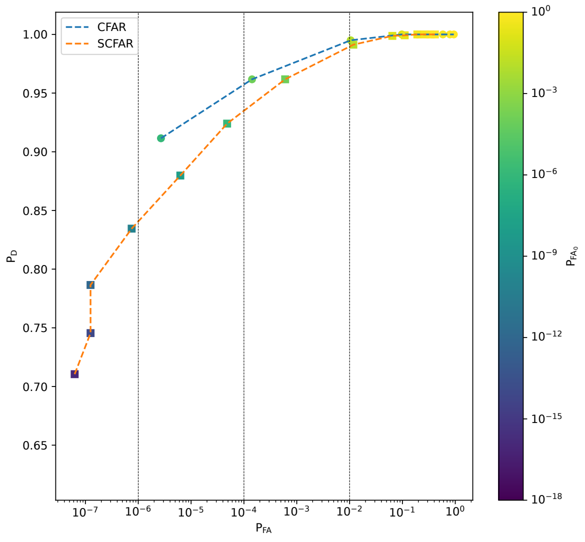
power caused by sidelobes raises significantly above the noise power, these will be seen as targets when having P_{FA0} that is too high (figure 6.5), and the masks do not account for this. As it turns out, the static threshold helps dampen these false alarms once the noise is low enough, lowering the false alarm rates in spiking CFAR while the conventional algorithm has more false alarms. A separate high-resolution run is done using a 40dB noise figure to verify the constant false alarm behaviour. To still account for various SNRs, the Radar Cross Section (RCS) now varies between 0 and 3dBsqm. Figure 6.6 shows that indeed both implementations do in fact have a constant false alarm rates. This again also shows that the increased detection rate comes at the cost of an increased false alarm rate.

The high-resolution run data can also be used to generate a ROC curve with a single noise power in terms of performance measurement. Figure 6.7 shows the ROC curves of a single noise floor of the high-resolution run. While the classification rates of both versions are high, the conventional algorithm does perform better. In theory (with infinite samples), each dot should align with the desired false alarm rate. Just as with the separate SNR plot, it is visible that the false alarm rate is higher for each point, indicating worse performance. False alarm rates below 10^{-6} are not reliable as not enough samples have been collected to estimate the probability.

The stochasticity from the Poisson group explains the difference in false alarms. Figure 6.8 shows some examples of false alarms. As the P_{FA0} is lowered, the error becomes bigger. The error between the exact algorithm and membrane potential is measured to analyse this anomaly. By setting the threshold at an unreachable value, the membrane potential of an IF neuron after 50ms should reflect the same output. Figure 6.10a displays the error that is also observed in the ROC plots. As P_{FA0} increases,



(a) normal scale



(b) log scale

Figure 6.7: $u_{\text{static}} = 1\text{V}$ (LIF, reset) The ROC curve of the 40dB noise figure run indicates the performance of both the spiking and conventional version. The colour of each node is the input desired probability of false alarm (P_{FA0}).

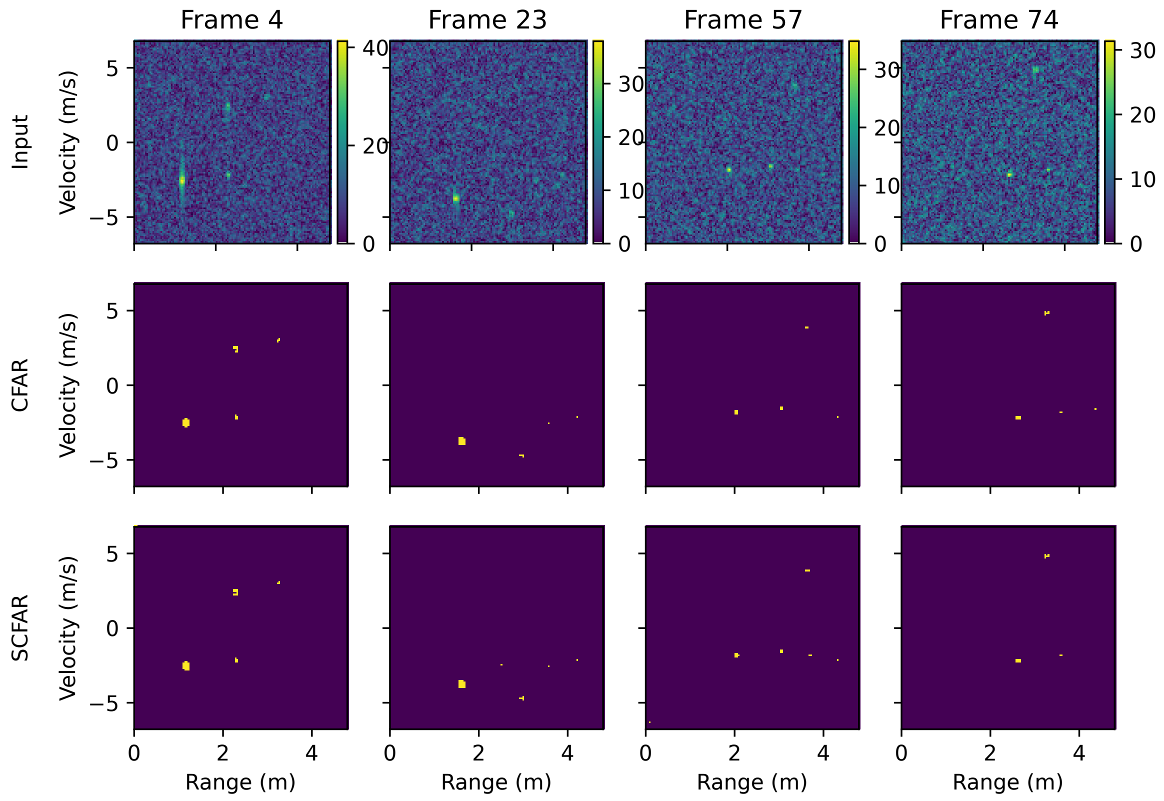
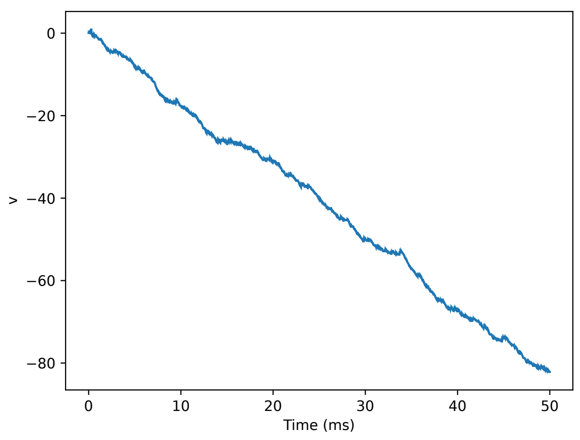
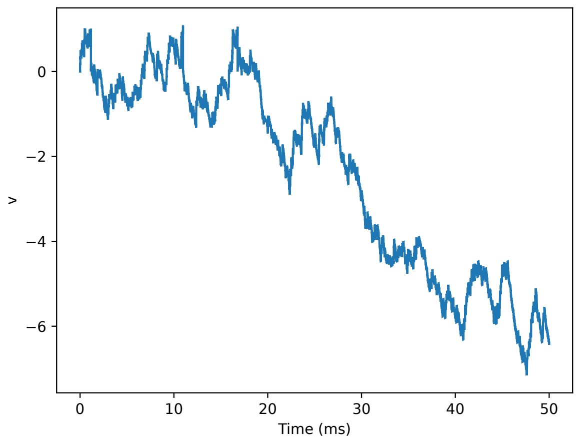


Figure 6.8: $u_{\text{static}} = 1\text{V}$ (LIF, reset). The worst frames from $P_{\text{FA0}} = 10^{-6}$. The spiking implementation generally has more false alarms, both miss detections.



(a) IF



(b) LIF

Figure 6.9: The membrane potential of two false alarms run with $P_{\mathrm{FA0}} = 10^{-8}$. In both cases, the dynamic threshold was double the Cell Under Test (CUT) value.

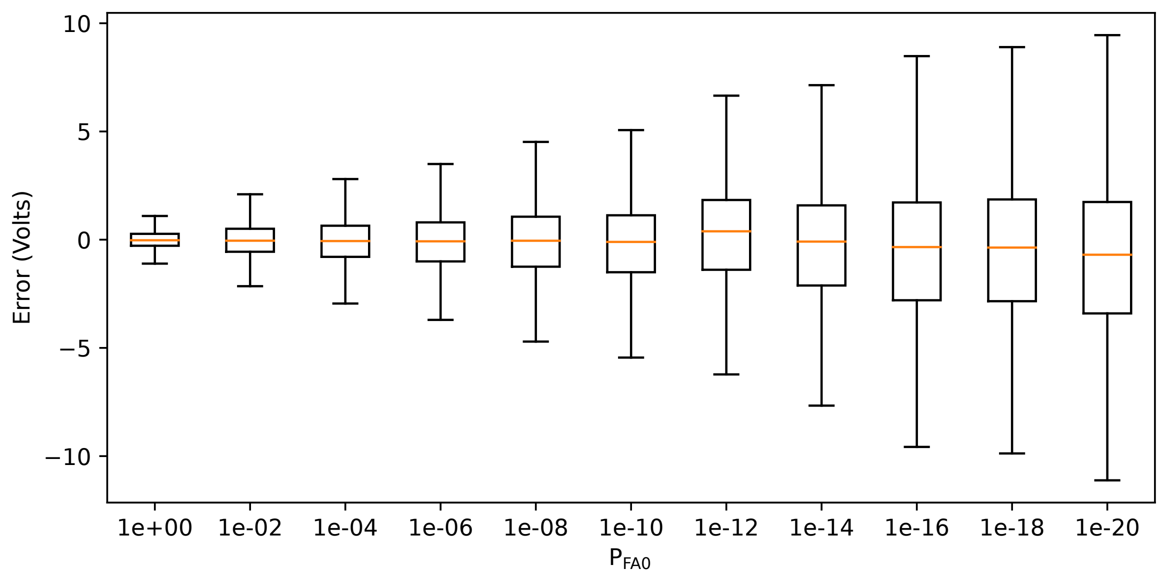
α becomes more significant, and the weight of inhibitory synapses grows larger. Usually, this would not be a problem, but with the Poisson input, a random spike at the wrong moment could result in a false alarm (figure 6.9). There are two possible solutions to this problem: (1) increase the spike rate to lower the effect of random individual spikes being missed (figure 6.10b). (2) adjust the static threshold per P_{FA0} to take into account the increased error rate.

Another high-resolution run with a static threshold dependant on the desired false alarm rate was run to investigate whether having a different static threshold per P_{FA0} will make the false alarm rate more in line with the conventional CFAR detector (figure 6.11). While it did lower the false alarm rate, it confirms the results in figure 6.7 that even with higher detection rates, simply choosing a lower input false alarm rate will perform better. In conclusion, the spiking CFAR will not perform better than the conventional algorithm because of the variance in spike when using one-to-one spike rate encoding,

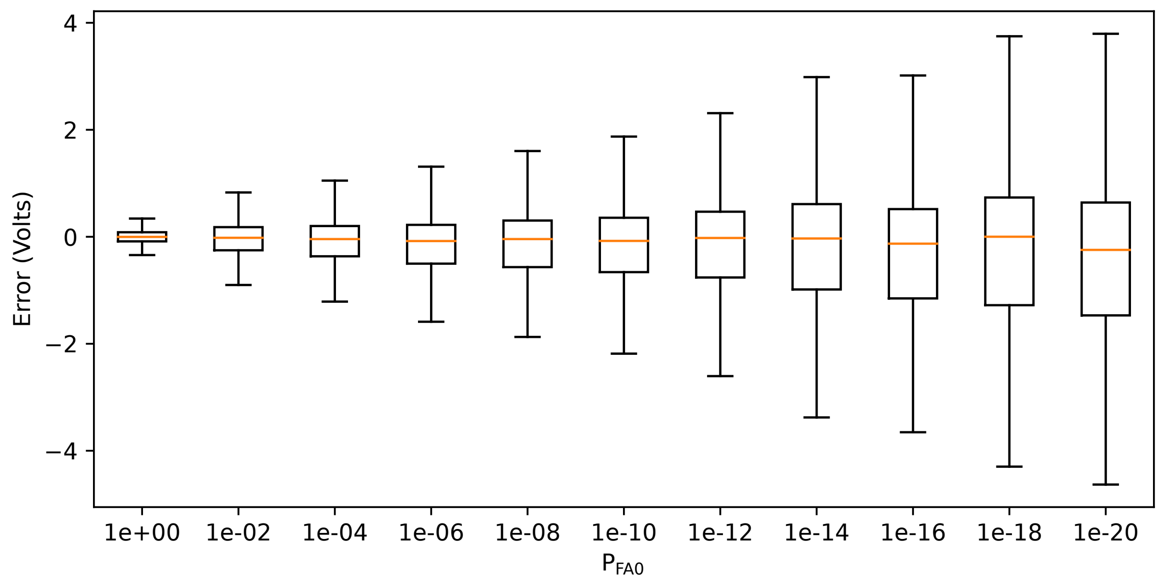
6.2.2 Recurrent Network

As shown in chapter 5, the recurrent network increases the false alarm rate too much for it to be worth it. Picking out some of the relatively worst frames compared with the original algorithm reveals more while this is the case. While the high power nearby targets are amplified, the low power distant target benefits from this insufficiently. In addition, the recurrent spikes generate false alarms, as can be seen in figure 6.12.

Unfortunately, the amplified targets are not beneficial enough to compensate for the false alarm rate loss: in theory, only a single pixel is needed for detection, and the target size is irrelevant. As such, the recurrent network in its current form is not feasible as an accurate replacement.

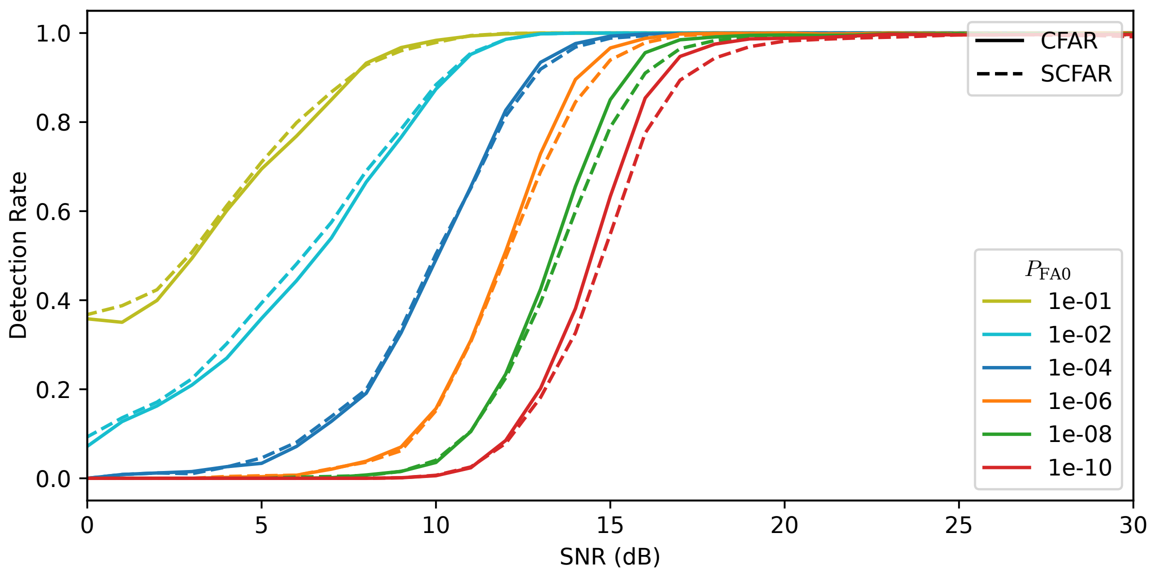


(a) 1x spike rate

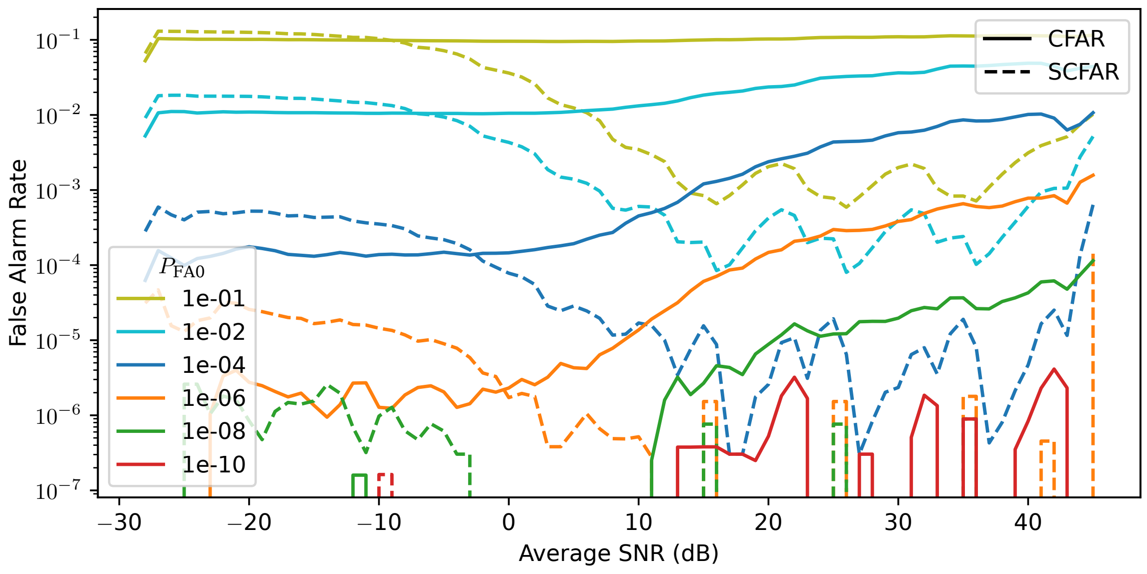


(b) 10x spike rate

Figure 6.10: The error was measured using the membrane potential between the exact and spiking algorithm using an IF neuron. Using ten times the spike rate about halves the error.



(a) P_{D} ROC



(b) P_{FA} ROC

Figure 6.11: (LIF, reset). A different high-resolution run with the threshold adjusted for each P_{FA0}. It effectively lowers both the probability and false alarm rate but is not enough to compensate for the loss introduced by the Poisson input.

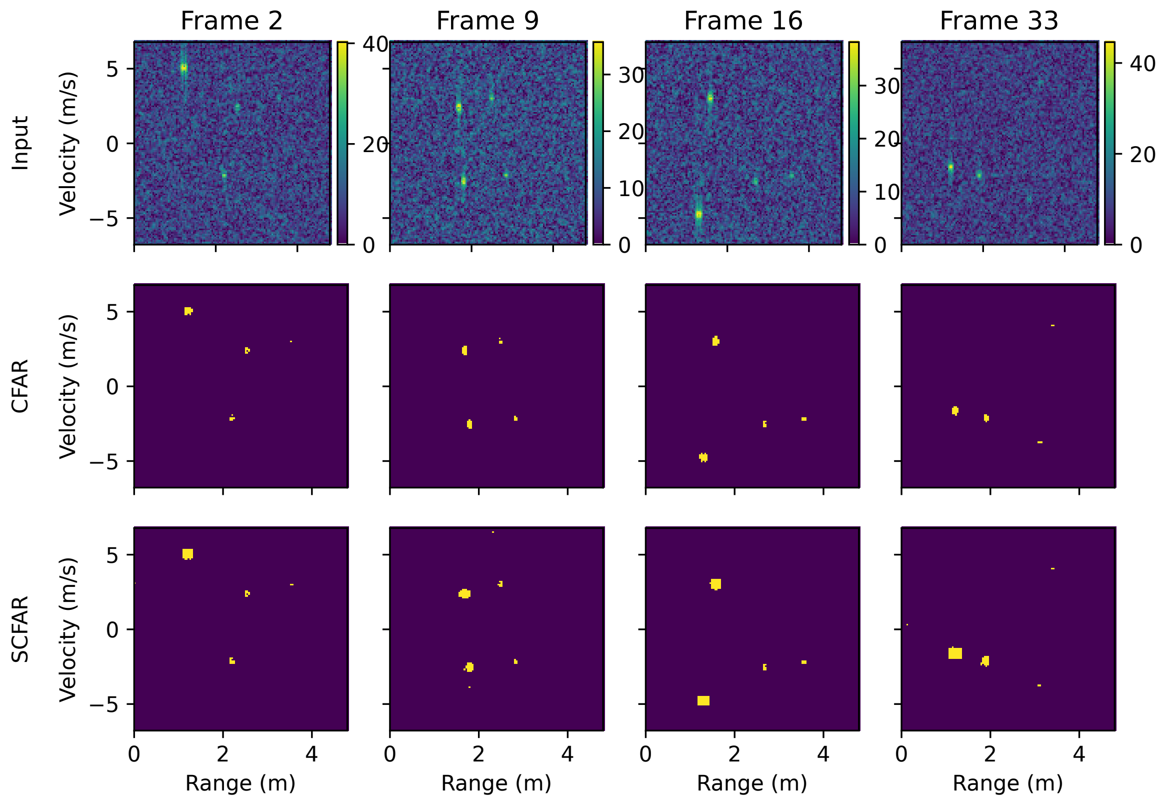


Figure 6.12: $u_{\text{static}} = 1\text{V}$ (LIF, reset). The worst frames from $P_{\text{FA0}} = 10^{-6}$ with the recurrent network. While high power targets are amplified outside of their guard neurons, low power targets are not enough. In the process, the recurrent synapses generate an additional false alarm.

7 Conclusions

This thesis presents several Spiking Cell-Averaging Constant False Alarm Rate (CFAR) implementations. While the implementation itself was the goal, several configurations have been simulated and analysed. The spiking CFAR is not identical to its conventional counterpart but behaves similarly. The configuration that resets each frame using the Leaky Integrate-and-Fire (LIF) neuron model has a higher detection rate at the cost of an increasingly higher false alarm rate per desired false alarm rate. A novel recurrent version of spiking CFAR attempted on improving this further but raised the false alarms rates to an unacceptable level.

Even though both the conventional and spiking CFAR have variations throughout the false alarm rates, the implementation without any frame reset, and a 1V static threshold has almost no measured false alarms, but at the cost of detection rate. This attenuation was caused by the Spiking Neural Network (SNN) primarily integrating over the noise and not the target power, significantly increasing the Signal-to-Noise Ratio (SNR) required for detection.

Spiking CFAR does have a constant false alarm rate. While frames with a high SNRs, the spiking CFAR performed better due to its static threshold. With realistic SNRs, the false alarm rate is constant in both the conventional and spiking CFAR detector and generally worse than the exact algorithm.

In conclusion, this thesis presents a working Spiking Cell-Averaging CFAR algorithm using an established SNN simulator. The false alarm rate of Spiking CFAR is higher than its conventional counterpart due to the Poisson input group. While this also comes with a higher detection rate, the trade-off is not worth it. Future improvements will have to increase the spike rate or change the network architecture to consider the spike rate variance.

Future Work

There are still several open subjects to research and improve. The topics relate to better understanding the influence of spike rates and accuracy, using the network connectivity to improve detections further and suppress false alarms.

- Have a population of neurons per output value to counter the stochasticity. The majority of the population would decide whether there is a target or not.
- Test the Spiking CFAR performance on clutter edges (different noise floors within a radar frame).
- Find out what is the lowest possible spike rates to get accurate detections still. e.g. neurons in humans have a refractory period of 1ms.

- Employ a learning scheme to modify the weights of recurrent synapses. I.e. if an output neuron receives a recurrent spike and spikes itself shortly after, increase the weight of the recurrent connection (Spike-Timing Dependent Plasticity).
- Implement Discrete Fourier Transforms in the network such that the raw radar input can directly be used as input into the SNN.
- Neuromorphic hardware implementation to find the power consumption of Spiking CFAR.

Result Tables A

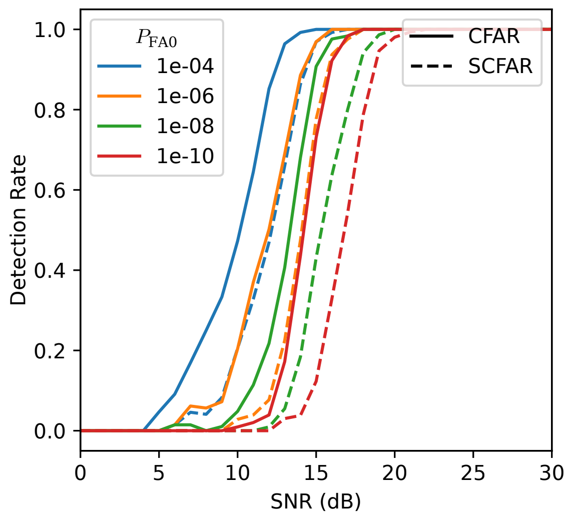
A.1 Real World Dataset Similarities

Table A.1: The similarity of simulations for unlabeled datasets.

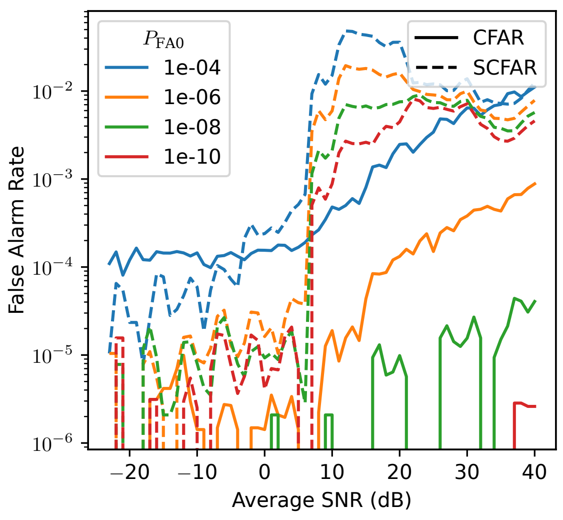
				Similarity						
Dataset	Model	Reset	u_{static}	1e-00	1e-01	1e-02	1e-04	1e-06	1e-08	1e-10
Spinner 1	LIF	✗	0V	76%	49%	60%	76%	72%	64%	60%
		✗	1V	21%	38%	67%	81%	74%	66%	62%
		✓	0V	38%	54%	71%	84%	85%	79%	78%
		✓	1V	18%	41%	72%	89%	88%	81%	79%
	IF	✗	0V	75%	28%	46%	46%	33%	21%	14%
		✗	1V	43%	28%	47%	47%	33%	21%	14%
		✓	0V	22%	45%	73%	88%	87%	81%	79%
		✓	1V	22%	45%	73%	88%	87%	81%	78%
Spinner 2	LIF	✗	0V	76%	51%	62%	73%	68%	59%	54%
		✗	1V	23%	43%	70%	76%	70%	60%	56%
		✓	0V	40%	58%	73%	83%	82%	75%	72%
		✓	1V	20%	46%	75%	86%	85%	77%	74%
	IF	✗	0V	76%	30%	42%	38%	29%	18%	13%
		✗	1V	44%	30%	42%	38%	28%	18%	13%
		✓	0V	24%	50%	76%	86%	84%	76%	74%
		✓	1V	24%	50%	76%	86%	84%	76%	74%
Meccano	LIF	✗	0V	81%	52%	54%	66%	68%	67%	61%
		✗	1V	27%	36%	52%	69%	70%	69%	62%
		✓	0V	46%	59%	64%	72%	75%	75%	68%
		✓	1V	20%	42%	63%	79%	80%	78%	70%
	IF	✗	0V	80%	25%	33%	50%	54%	50%	50%
		✗	1V	49%	23%	33%	50%	54%	50%	50%
		✓	0V	26%	49%	65%	78%	79%	78%	71%
		✓	1V	26%	49%	65%	78%	79%	78%	70%

B Receiver Operating Characteristics

This appendix contains the most important Receiver Operating Characteristic (ROC) plots, showing the detection and false alarm performance of the simulated radar dataset.

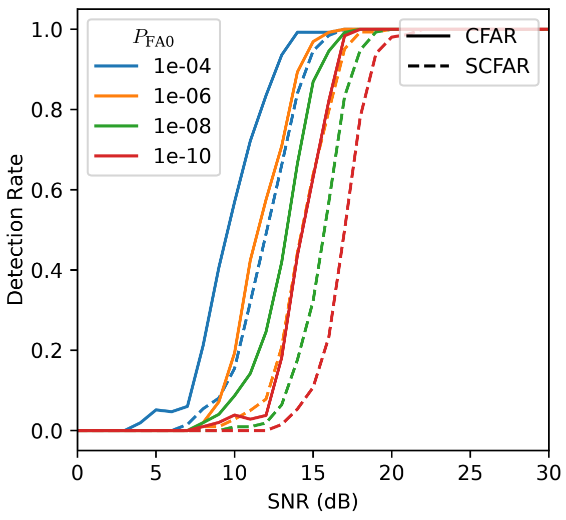


(a) P_{D} ROC

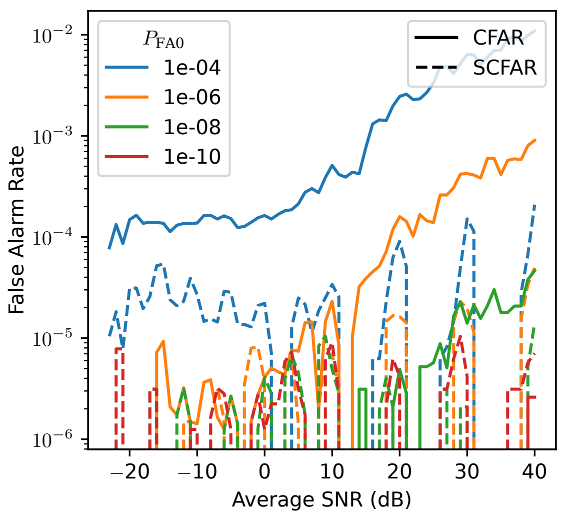


(b) P_{FA} ROC

Figure B.1: Direct Input $u_{\mathrm{static}} = 0\mathrm{V}$ (LIF, no reset)

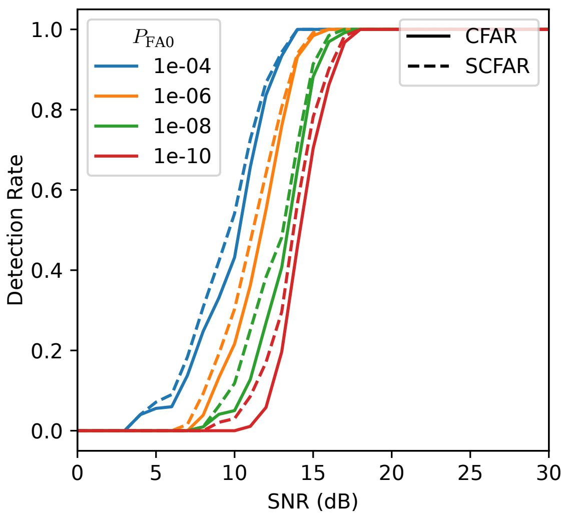


(a) P_{D} ROC

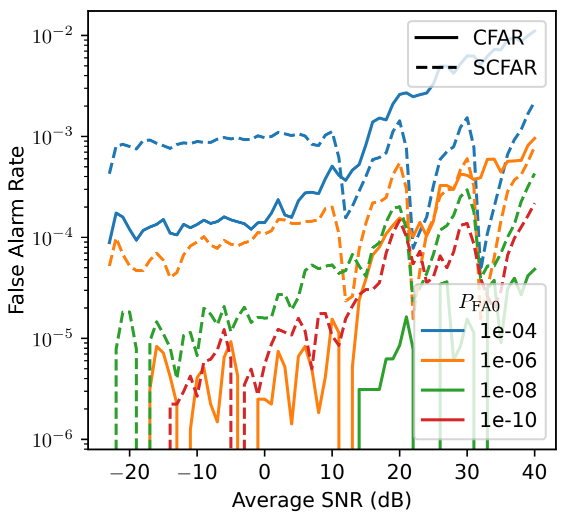


(b) P_{FA} ROC

Figure B.2: Direct Input $u_{\text{static}} = 1\mathrm{V}$ (LIF, no reset)

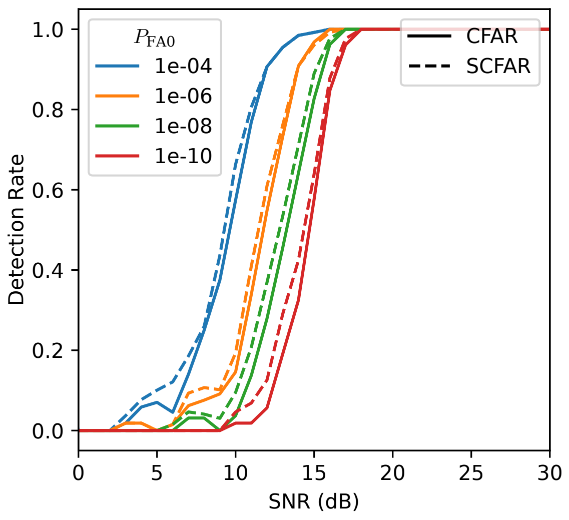


(a) P_{D} ROC

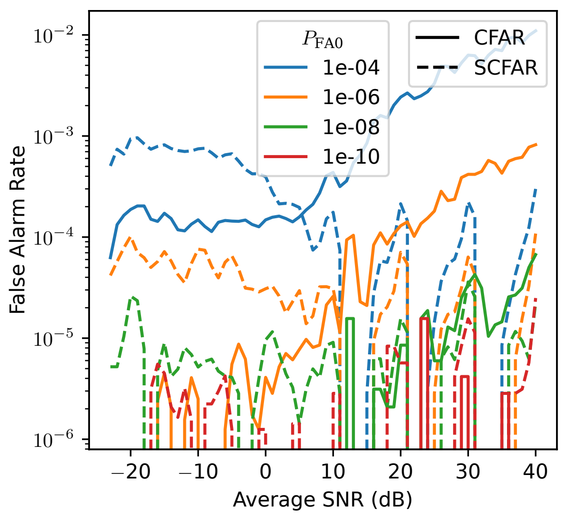


(b) P_{FA} ROC

Figure B.3: Direct Input $u_{\text{static}} = 0\mathrm{V}$ (LIF, reset)

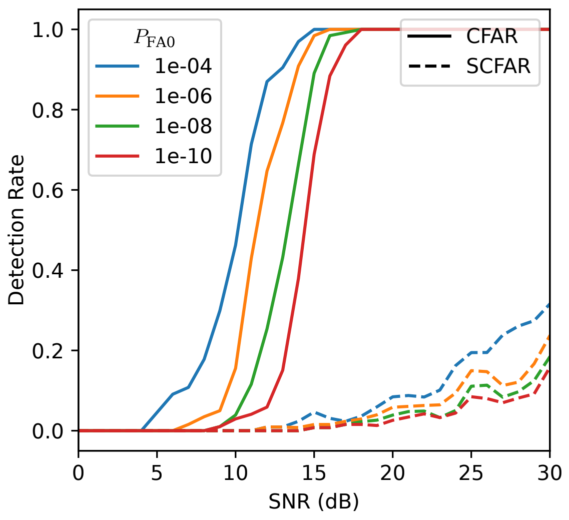


(a) P_{D} ROC

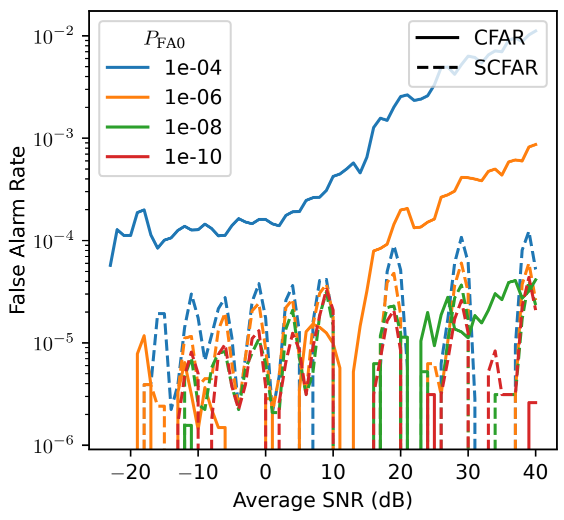


(b) P_{FA} ROC

Figure B.4: Direct Input $u_{\text{static}} = 1\text{V}$ (LIF, reset)

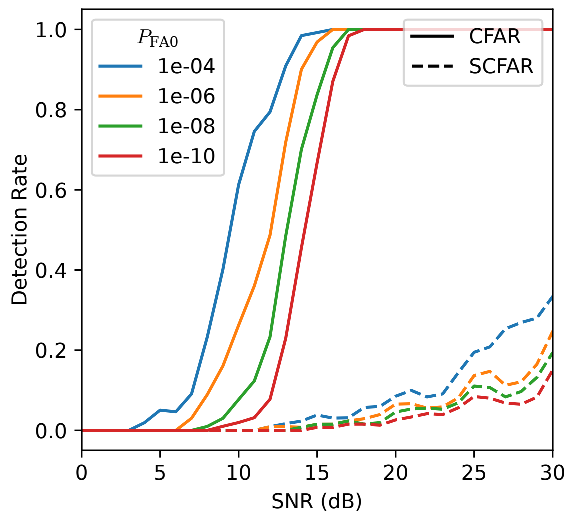


(a) P_{D} ROC

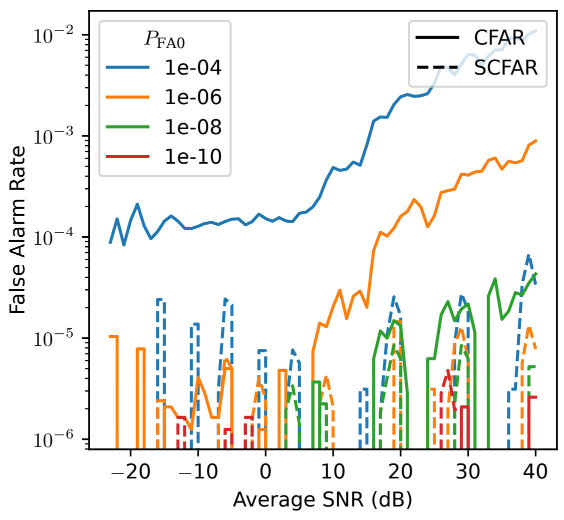


(b) P_{FA} ROC

Figure B.5: Direct Input $u_{\text{static}} = 0\text{V}$ (Integrate-and-Fire (IF), no reset)

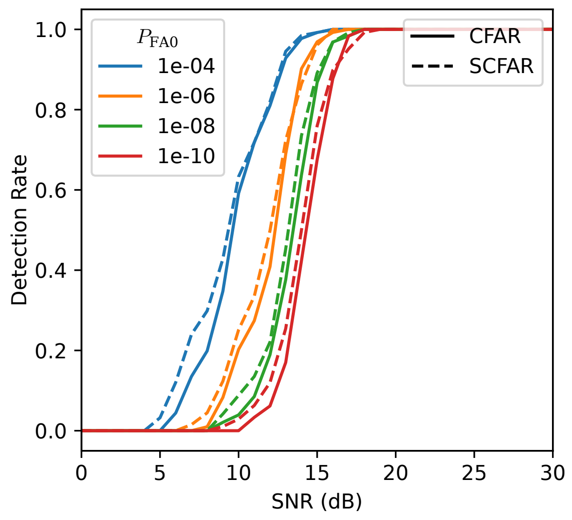


(a) P_{D} ROC

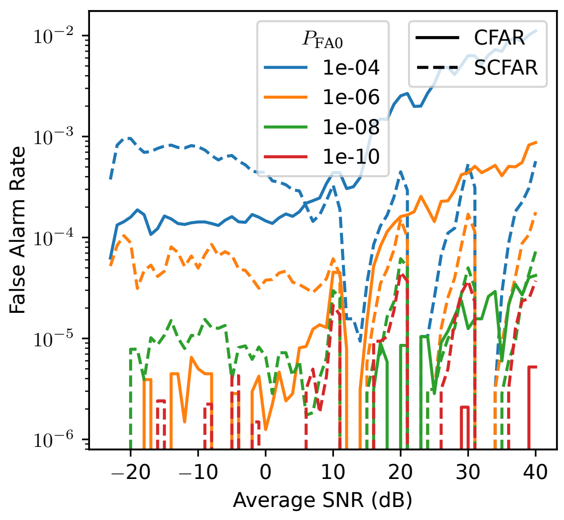


(b) P_{FA} ROC

Figure B.6: Direct Input $u_{\text{static}} = 1\text{V}$ (IF, no reset)

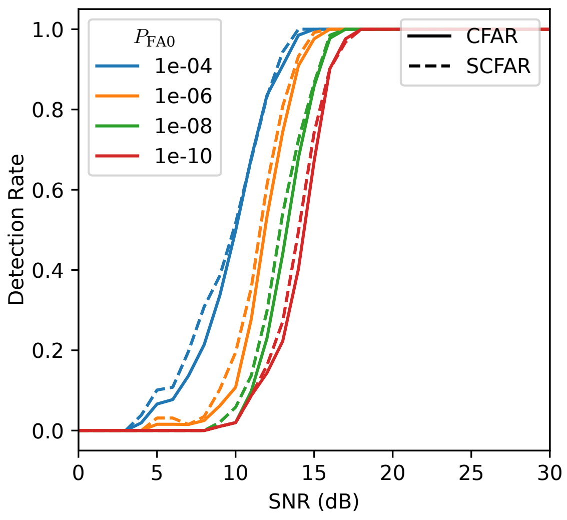


(a) P_{D} ROC

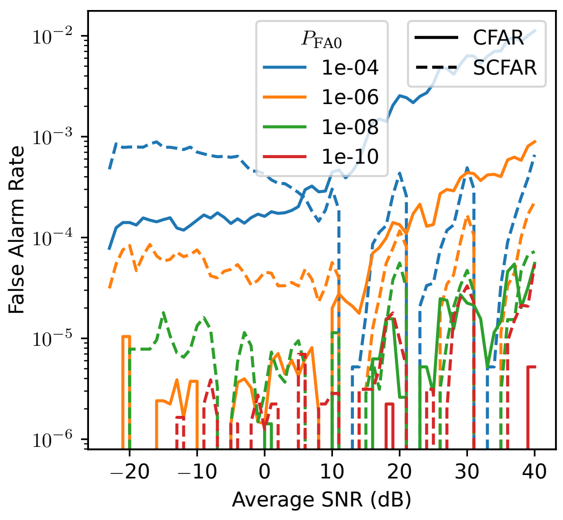


(b) P_{FA} ROC

Figure B.7: Direct Input $u_{\text{static}} = 0\text{V}$ (IF, reset)

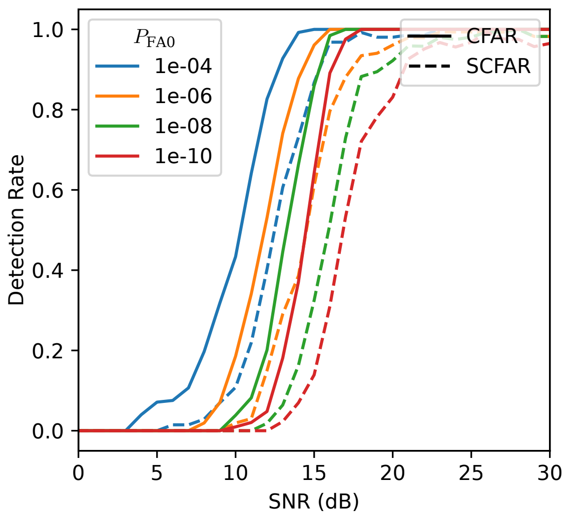


(a) P_{D} ROC

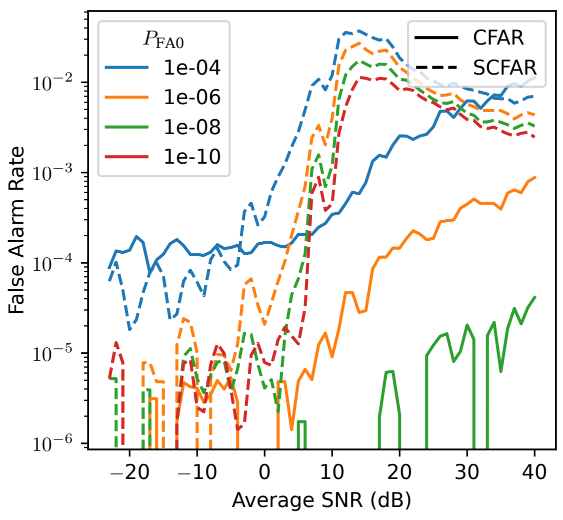


(b) P_{FA} ROC

Figure B.8: Direct Input $u_{\mathrm{static}} = 1\mathrm{V}$ (IF, reset)

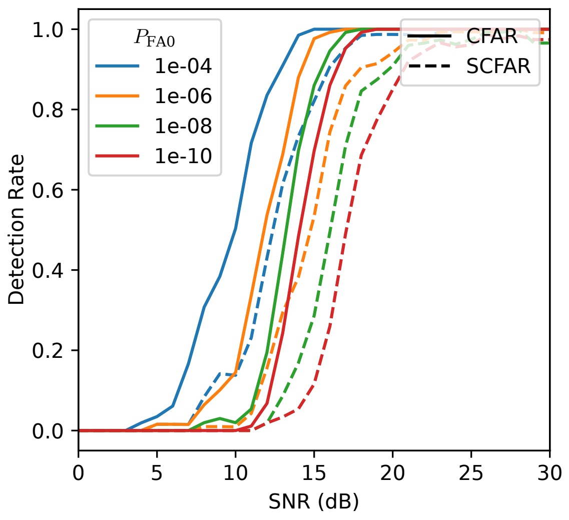


(a) P_{D} ROC

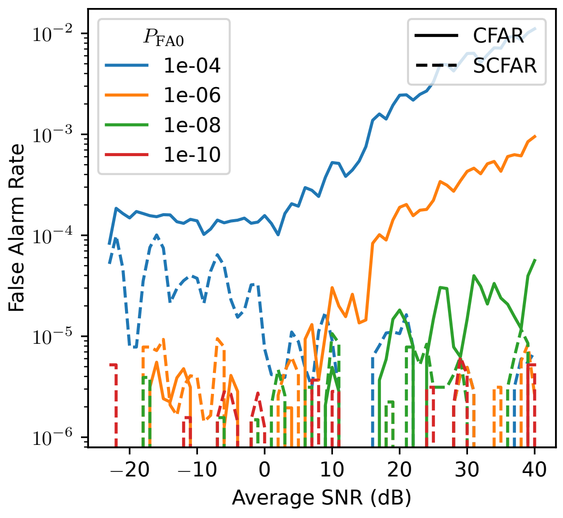


(b) P_{FA} ROC

Figure B.9: Normalised Input $u_{\mathrm{static}} = 0\mathrm{V}$ (LIF, no reset)

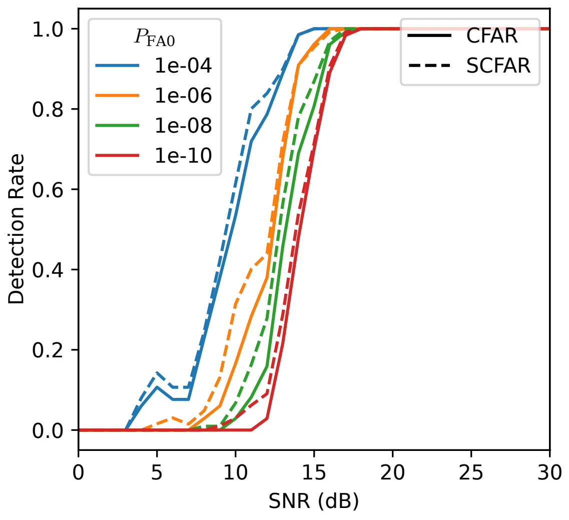


(a) P_{D} ROC

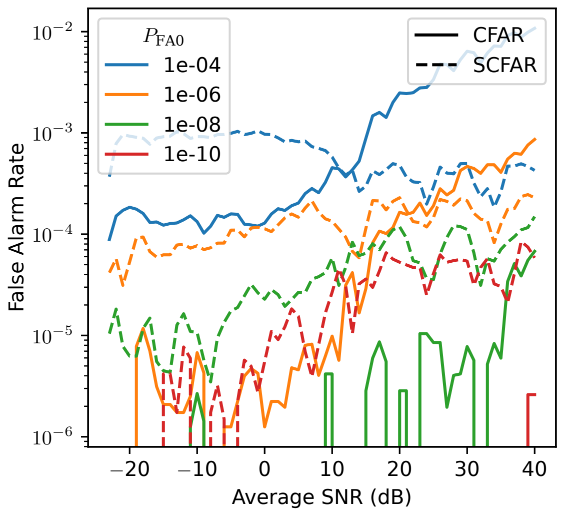


(b) P_{FA} ROC

Figure B.10: Normalised Input $u_{\mathrm{static}} = 1\mathrm{V}$ (LIF, no reset)

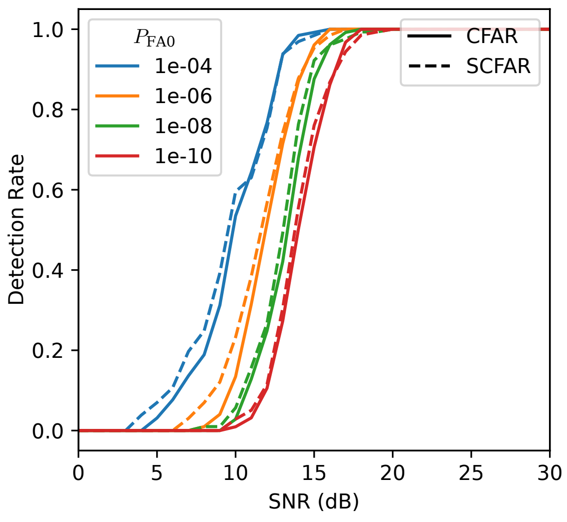


(a) P_{D} ROC

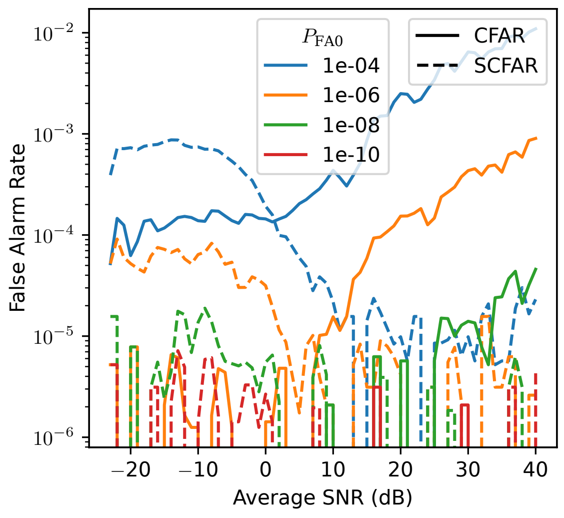


(b) P_{FA} ROC

Figure B.11: Normalised Input $u_{\mathrm{static}} = 0\mathrm{V}$ (LIF, reset)

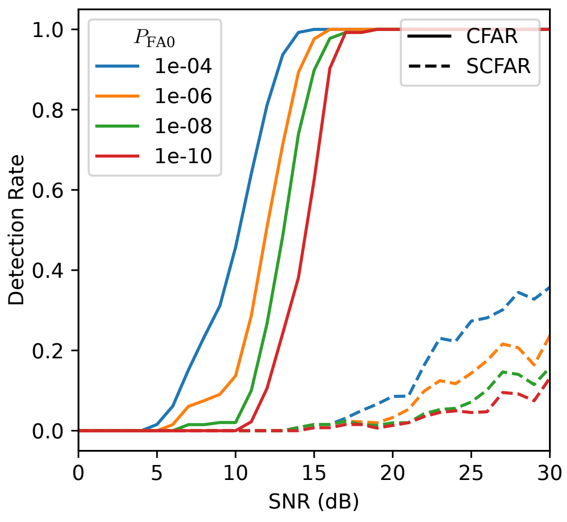


(a) P_{D} ROC

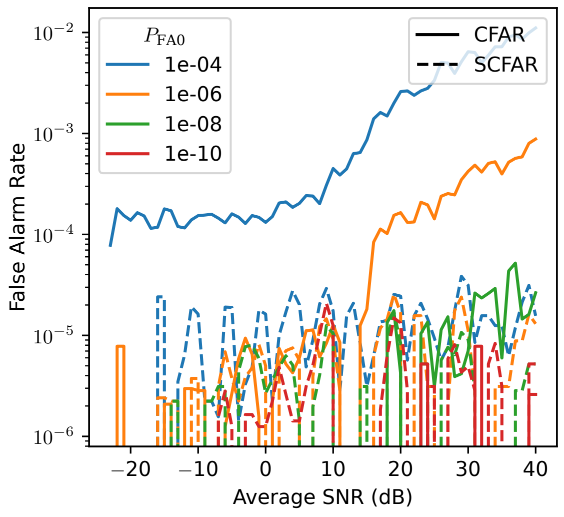


(b) P_{FA} ROC

Figure B.12: Normalised Input $u_{\mathrm{static}} = 1\mathrm{V}$ (LIF, reset)

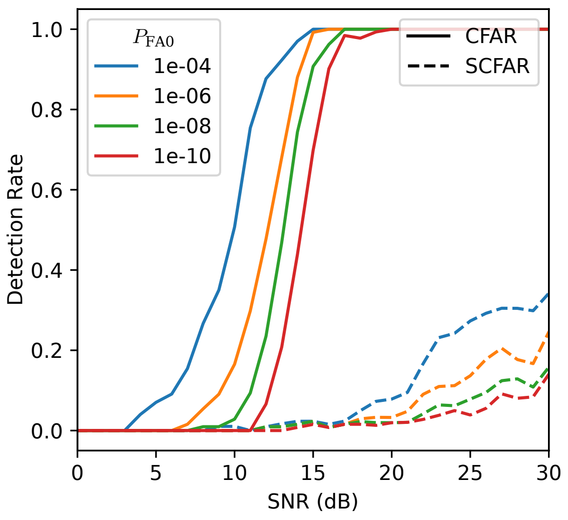


(a) P_{D} ROC

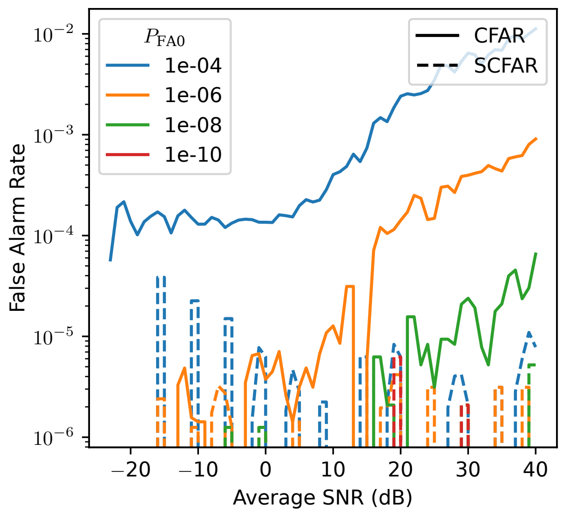


(b) P_{FA} ROC

Figure B.13: Normalised Input $u_{\mathrm{static}} = 0\mathrm{V}$ (IF, no reset)

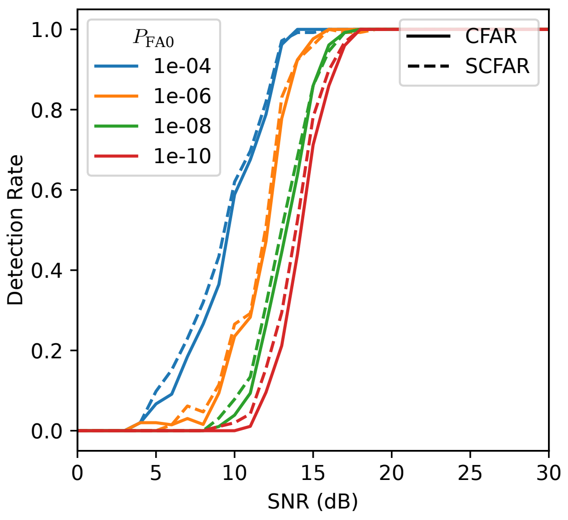


(a) P_D ROC

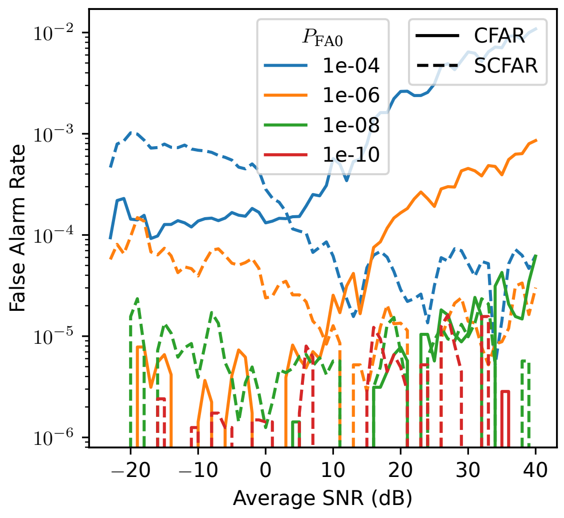


(b) P_{FA} ROC

Figure B.14: Normalised Input $u_{\text{static}} = 1\text{V}$ (IF, no reset)

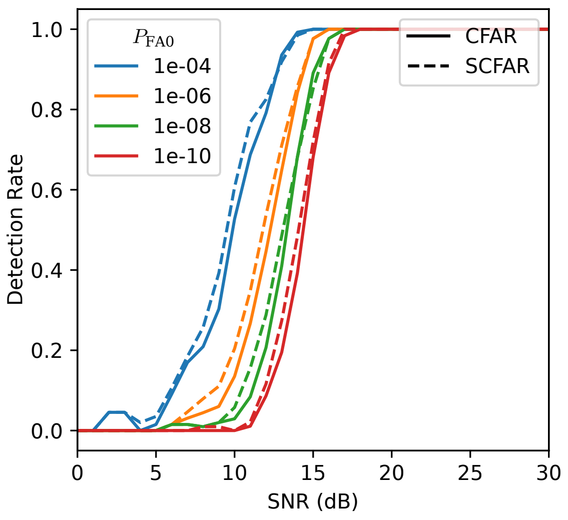


(a) P_D ROC

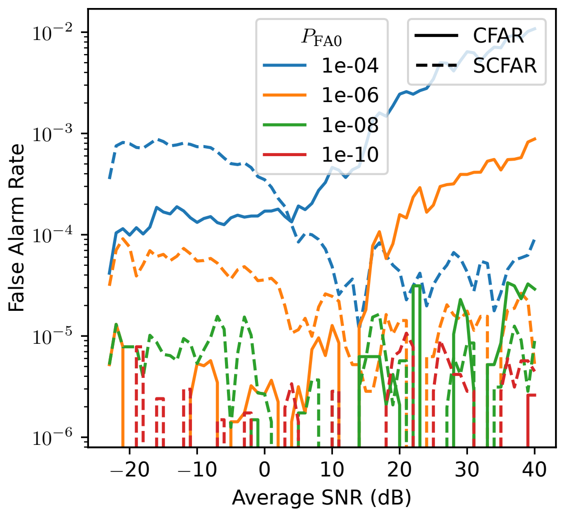


(b) P_{FA} ROC

Figure B.15: Normalised Input $u_{\text{static}} = 0\text{V}$ (IF, reset)



(a) P_{D} ROC



(b) P_{FA} ROC

Figure B.16: Normalised Input $u_{\mathrm{static}} = 1\mathrm{V}$ (IF, reset)

Bibliography

[1] Jabran Akhtar and Karl Erik Olsen. A neural network target detector with partial CA-CFAR supervised training. In *2018 International Conference on Radar (RADAR)*, pages 1–6, August 2018.

[2] W Baldygo, R Brown, M Wicks, P Antonik, G Capraro, and L Hennington. Artificial intelligence applications to constant false alarm rate (CFAR) processing. In *The Record of the 1993 IEEE National Radar Conference*, pages 275–280, April 1993.

[3] David Knox Barton. *Radar Equations for Modern Radar*. Artech House, 2013.

[4] S Blake. OS-CFAR theory for multiple targets and nonuniform clutter. *IEEE Trans. Aerosp. Electron. Syst.*, 24(6):785–790, November 1988.

[5] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris, Jr, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P Davison, Sami El Boustani, and Alain Destexhe. Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comput. Neurosci.*, 23(3):349–398, December 2007.

[6] J H Bryant. The first century of microwaves-1886 to 1986. *IEEE Trans. Microw. Theory Tech.*, 36(5):830–858, May 1988.

[7] M Vizcarro i Carretero, R I A Harmanny, and R P Trommel. Smart-CFAR, a machine learning approach to floating level detection in radar. In *2019 16th European Radar Conference (EuRAD)*, pages 161–164, October 2019.

[8] P P Gandhi and S A Kassam. Analysis of CFAR processors in nonhomogeneous background. *IEEE Trans. Aerosp. Electron. Syst.*, 24(4):427–445, July 1988.

[9] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, November 2016.

[11] Hananel Hazan, Daniel J Saunders, Hassaan Khan, Devdhar Patel, Darpan T Sanghavi, Hava T Siegelmann, and Robert Kozma. BindsNET: A machine Learning-Oriented spiking neural networks library in python. *Front. Neuroinform.*, 12:89, December 2018.

[12] A L Hodgkin and A F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.*, 117(4):500–544, August 1952.

[13] Sung Won Hong and Dong Seog Han. Performance analysis of an environmental adaptive CFAR detector. *Math. Probl. Eng.*, 2014(6):1–7, May 2014.

[14] Cesar Iovescu and Sandeep Rao. The fundamentals of millimeter wave sensors. *Texas Instruments*, pages 1–8, 2017.

[15] M Jankiraman. *FMCW Radar Design.* Artech House, July 2018.

[16] Miao Kang, Xiangguang Leng, Zhao Lin, and Kefeng Ji. A modified faster R-CNN based on CFAR algorithm for SAR ship detection. In *2017 International Workshop on Remote Sensing with Intelligent Processing (RSIP)*, pages 1–4, May 2017.

[17] Nikola K Kasabov. *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence.* Springer, Berlin, Heidelberg, 2019.

[18] Chia-Hung Lin, Yu-Chien Lin, Yue Bai, Wei-Ho Chung, Ta-Sung Lee, and Heikki Huttunen. DL-CFAR: A novel CFAR target detection method based on deep learning. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–6, September 2019.

[19] Javier López-Randulfe, Tobias Duswald, Zhenshan Bing, and Alois Knoll. Spiking neural network for fourier transform and object detection for automotive radar. *Front. Neurorobot.*, 15:688344, June 2021.

[20] Mark A Richards. *Fundamentals of radar signal processing.* McGraw-Hill Education, 2014.

[21] Christoph Schroeder and Hermann Rohling. X-band FMCW radar system with variable chirp duration. In *2010 IEEE Radar Conference.* IEEE, 2010.

[22] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. May 2017.

[23] Marcel Stimberg, Romain Brette, and Dan Fm Goodman. Brian 2, an intuitive and efficient neural simulator. *Elife*, 8, August 2019.

[24] Leiou Wang, Donghui Wang, and Chengpeng Hao. Intelligent CFAR detector based on support vector machine. *IEEE Access*, 5:26965–26972, 2017.

[25] Jiafei Zhao, Rongkun Jiang, Xuetian Wang, and Hongmin Gao. Robust CFAR detection for multiple targets in K-Distributed sea clutter based on machine learning. *Symmetry*, 11(12):1482, December 2019.