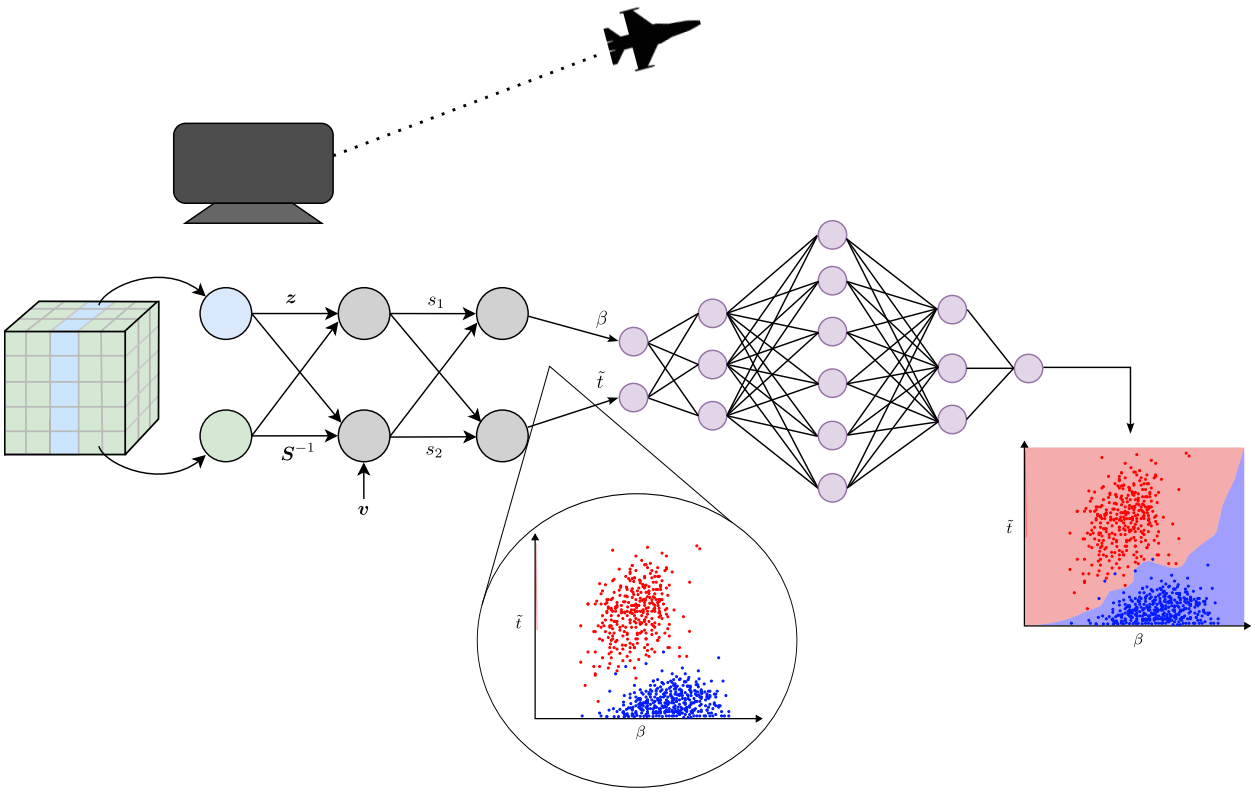
CHALMERS
UNIVERSITY OF TECHNOLOGY



Real-time Target Detection Using a CFAR Feature Plane in an Embedded System

Master's thesis in Embedded Electronic System Design

CAROLINA WEBER
NICKLAS WRIGHT

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Master's thesis 2025

Real-time Target Detection Using a CFAR Feature Plane in an Embedded System

CAROLINA WEBER
NICKLAS WRIGHT

Department of Microtechnology and Nanoscience
Chalmers University of Technology
Gothenburg, Sweden 2025

Real-time Target Detection Using a CFAR Feature Plane in an Embedded System
CAROLINA WEBER
NICKLAS WRIGHT

Supervisor: Per Larsson-Edefors, Department of Microtechnology and Nanoscience
Company advisors: Christoffer Krull & Patrik Dammert, Saab AB
Examiner: Lena Peterson, Department of Microtechnology and Nanoscience

Master's Thesis 2025
Department of Microtechnology and Nanoscience
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Conceptual block diagram of the proposed radar detector.

Typeset in LaTeX
Gothenburg, Sweden 2025

Real-time Target Detection Using a CFAR Feature Plane in an Embedded System
CAROLINA WEBER
NICKLAS WRIGHT
Department of Microtechnology and Nanoscience
Chalmers University of Technology

Abstract

Technologically advanced weapon systems – such as drones and high-precision ballistic missiles – have become defining features of modern warfare. This development has increased the demand for radar systems capable of detecting threats quickly and accurately, even in dynamic and cluttered environments, while maintaining a constant false-alarm rate (CFAR) independent of the interference model. In such settings, signal mismatches between expected and measured target steering vectors are common, requiring detectors that can be tuned for desired selectivity or robustness. The CFAR Feature Plane (CFAR-FP) is a recently proposed method for evaluating different CFAR detectors. It maps radar echoes to a two-dimensional feature space using invariant detection principles, forming distinct groups of clusters for the target and noise hypotheses. Within this plane, traditional CFAR detectors appear as linear or non-linear decision boundaries that separate the clusters depending on the desired selectivity or robustness. However, in scenarios with low signal-to-noise ratio (SNR) or significant signal mismatches, these detectors may suffer from degraded performance. To address this issue, a neural network (NN) can be used as a binary classifier to learn complex and data-driven detection thresholds, which otherwise would not be possible with traditional detectors. This thesis explores the design and implementation of a robust and tunable CFAR detector based on the CFAR-FP framework and an NN directly in the Xilinx Versal AI core series VCK190 FPGA. The board's combination of reconfigurable logic and embedded AI engines (AIE) has the potential of greatly accelerating NN-based classification in real time, making it a viable candidate for edge AI applications. A complete system model was developed in software, including a MATLAB program for generating the CFAR-FP with customizable target injections into experimental radar data, and an NN model implemented in Python. Experimental results demonstrate that quantizing the NN for deployment on resource-constrained platforms – such as the VCK190 – significantly improves inference speed, albeit with a reduction in prediction accuracy on highly mismatched and low SNR datasets. In parallel, VHDL-based modules have been developed for executing advanced complex-valued linear algebra operations required by the CFAR-FP mapping chain on an FPGA. The results show that a fully pipelined hardware implementation – from processing chain to NN inference – is feasible, enabling high-speed signal processing and detections at the cost of higher design complexity and loss in computational precision.

Keywords: edge AI, CFAR, feature plane, FPGA, edge, radar detector, neural network, GLRT, mismatched signals

Acknowledgements

This challenging and multi-disciplinary project would not have gotten as far as it did without the help of a few people that deserve our deepest gratitude. We would like to especially thank our company advisors at Saab AB. Thank you, Christoffer Krull, for always making time to meet with us and provide valuable feedback about the project's developments, as well as making sure we got everything we needed in terms of resources and hardware. Thank you, Patrik Dammert, for taking the time to answer our questions regarding radar signal processing. We would also like to thank Daniel Wallström for entrusting us with this interesting project, and Mattias Ekström for providing us with the experimental radar data. Finally, we would like to thank our project supervisor at Chalmers, Per Larsson-Edefors, for helping us plan and structure the project and Lena Peterson at Chalmers for her role as examiner during the project.

Carolina Weber & Nicklas Wright, Gothenburg, June 2025

Contents

List of Abbreviations

Below is the list of acronyms that are used throughout this report listed in alphabetical order.

ADC	Analog-to-digital converter
AI	Artificial intelligence
AIE	Artificial intelligence engine
AMF	Adaptive matched filter
ANN	Artificial neural network
AOA	Angle of arrival
API	Application programming interface
AXI	Advanced eXtensible interface
BCE	Binary cross-entropy loss
CA	Cell-averaging
CDIV	Complex divider
CFAR	Constant false alarm rate
CFAR-FP	CFAR feature plane
CMUL	Complex multiplier
CORDIC	Coordinate rotation digital computer
CPI	Coherent processing interval
CPU	Central processing unit
CUT	Cell under test
DNN	Deep neural network
DOA	Direction of arrival
DPU	Deep learning processing unit
DSP	Digital signal processing
DUT	Design under test
ED	Energy detector
FFT	Fast Fourier transform
FN	False negative
FP	False positive
FP32	Single-precision (32-bit) floating point representation
FPGA	Field-programmable gate array
FSM	Finite-state machine
GJ	Gauss-Jordan (elimination)
GLRT	Generalized likelihood ratio test

GPU	Graphics processing unit
HLS	High-level synthesis
INT8	8-bit integer representation
IoT	Internet of things
IQ	In-phase and quadrature
LLM	Large language model
LO	Local oscillator
LOS	Line-of-sight
LRT	Likelihood ratio test
LSB	Least significant bit
LUT	Lookup table
MAC	Multiply-accumulate
MCP	Multi-channel processing
MF	Matched filter
MLE	Maximum likelihood estimation
MSB	Most significant bit
NN	Neural network
NoC	Network on a chip
NP	Neyman-Pearson (rule)
ONNX	Open neural network exchange
PLL	Phase-locked loop
PPV	Positive predicted value
PRF	Pulse repetition frequency
PRI	Pulse repetition interval
RAM	Random access memory
ReLu	Rectified linear unit
RF	Radio frequency
RMB	Reed-Mallet-Brennan (rule)
RTL	Register-transfer level
SIMD	Single instruction, multiple data
SNR	Signal-to-noise ratio
SINR	Signal-to-interference-plus-noise ratio
SoC	System on a chip
STAP	Space-time adaptive processing
STALO	Stable local oscillator
2-D	Two-dimensional
3-D	Three-dimensional
TN	True negative
TP	True positive
ULA	Uniform linear array
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
VLIW	Very long instruction word

List of Symbols

Below is the nomenclature of the parameters and variables that are frequently used throughout this report.

c_0	Speed of light in vacuum
t_p	Pulse repetition interval
τ	Pulse width
f_c	Carrier frequency
f_d	Doppler frequency
λ_c	Wavelength
k	Wavenumber
R	Radial range to target
v	LOS (radial) velocity
δ_R	Range resolution
δ_v	Velocity resolution
θ_B	Antenna beam width
ω_R	Angular velocity of antenna
θ	Mismatch angle or angle from DOA to boresight of radar
θ_B	Antenna beam width
φ	Phase angle
G_0	Maximum gain
g	Amplitude function
α	Complex scalar for target signal
σ_n^2	Noise variance (power)
σ_s^2	Signal variance (power)
H_0	Null-hypothesis (only noise)
H_1	Alternative hypothesis (noise and target)
$\boldsymbol{z}$	Cell under test

$\boldsymbol{z}_k$	Secondary data cell
$\boldsymbol{n}$	Noise vector
$\boldsymbol{v}$	Estimated steering vector
$\boldsymbol{p}$	Measured steering vector
$\boldsymbol{R}$	Noise covariance matrix
$\boldsymbol{R}_{\mathrm{n}}$	Thermal noise covariance matrix
$\boldsymbol{R}_{\mathrm{c}}$	Clutter covariance matrix
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{S}$	Scatter matrix
$\boldsymbol{C}$	Clutter and noise covariance matrix
$\dagger$	Complex conjugate (Hermitian) transpose
$\|\| \cdot \|\|$	Euclidean norm of a vector (energy)
$\|\| \cdot \|\|_{\mathrm{F}}$	Frobenius norm of a matrix
$\otimes$	Kronecker product
γ	SNR
$\cos^2 \theta$	Signal mismatch
η	Detection threshold
t	Decision statistic
s_1	Maximal invariant statistic used in CFAR-FP mapping chain
s_2	Maximal invariant statistic used in CFAR-FP mapping chain
β	Normalized statistic/feature in CFAR-FP (x-axis)
$\tilde{t}$	Normalized statistic/feature in CFAR-FP (y-axis)
$\hat{y}$	Neural network prediction
FP	Number of false positives
FN	Number of false negatives
TP	Number of true positives
TN	Number of true negatives
P_{D}	Probability of detection
P_{FA}	Probability of false alarm
F_1	F1-score
ϵ_{r}	Relative error
ϵ_{F}	Frobenius norm error

1

Introduction

Target detection in radar systems is continuously evolving to adapt to developments in the current battle space. In order to stay one step ahead of potential threats, radar detection methods need to quickly adapt to the current situation. A commonly used detection method in radar development is called *constant false-alarm rate* (CFAR), which is used to set an appropriate and adaptable threshold for when a radar response should be classified as a target and when it should be considered noise or clutter. These two possible classifications are known as *test hypotheses*. The main benefit of CFAR is that it generates a constant probability of false alarm that is independent of the interference. Since the interference in radar data can vary greatly depending on the environment, an adaptable threshold is highly desirable. The methods for calculating this threshold have evolved over time to minimize false positives and find target echoes that might be hidden within the noise and clutter.

A new method to calculate this threshold was recently developed called the CFAR Feature Plane (CFAR-FP), which shares many similarities to unsupervised neural network (NN) models within the field of artificial intelligence (AI) [1]. The CFAR-FP processing chain maps radar data cells to a two-dimensional (2-D) plane in which the hypotheses will form distinct groups of clusters. These clusters share many similarities of so-called *features* in machine learning (ML), which transform datasets into individually measurable characteristics or properties represented by points in a plane. Between these clusters a decision region can be identified, which separates false positives from true positives. Existing CFAR detectors mapped to the plane appear as linear or non-linear continuous functions depending on the characteristics of the detector. Furthermore, *any* decision region drawn in the plane realizes a detector which exhibits the CFAR property. Once a CFAR-FP has been generated, an NN could theoretically be trained to find different ways to split this space to improve detection capability while still maintaining the CFAR property.

In collaboration with Saab Surveillance in Gothenburg, this thesis investigates the feasibility of implementing the CFAR-FP mapping chain together with an NN binary classifier in a real-time embedded system for radar target detection. While previous research has explored the theoretical benefits of CFAR-FP, its real-time application on hardware platforms remains somewhat uncharted territory. This project aims to bridge this gap by leveraging Xilinx's Versal AI Core series field-programmable gate array (FPGA) [2] to extract the CFAR-FP from raw radar data and deploy an NN detector directly in the hardware using its embedded AI engines (AIEs).

1.1 Background

Modern radar systems often utilize antennas with multiple array sensors to measure incoming radar echoes. This is beneficial in many aspects as it enables advanced signal processing techniques to be applied such as beamforming, which uses constructive or destructive interference to filter out return echoes arriving from spatial angles other than the direction desired. Furthermore, multichannel processing is an effective method for limiting the effects of clutter or jamming signals. However, the increasing number of antenna elements in modern radars has resulted in huge datasets that need to be processed in real-time, which necessitates fast and precise digital signal processing (DSP) systems.

The exponential growth of the Internet of Things (IoT) technology has revolutionized various industries. Traditionally, these technologies rely on cloud servers for data collection, storage, and processing. Shifting all computing tasks to the cloud has proven to be an efficient approach for data processing, as the cloud offers far more computational power than edge devices. However, while the speed of data processing has significantly improved, the bandwidth of networks connecting devices to the cloud has not seen similar growth [3]. As a result, with an ever-increasing volume of data being generated by edge devices, the network is becoming a bottleneck for cloud computing. Additionally, the constant threat of cyber attacks makes cloud storage of data risky, particularly in military applications where it is crucial that gathered intelligence is kept secret.

Edge devices and edge computing significantly enhance the autonomy of IoT systems by enabling localized data processing and interpretation, even in scenarios with limited or no network connectivity. The integration of AI within these environments enables real-time, efficient, and autonomous analysis and decision-making. Edge computing platforms typically incorporate a range of processing units – including central processing units (CPUs), graphics processing units (GPUs), AI accelerators, and hardware such as FPGAs [4] – that are optimized to perform AI computations at the edge. Embedded ML, often referred to as TinyML, involves deploying ML models directly onto resource-constrained devices like microcontrollers and FPGAs. FPGAs are particularly useful in embedded ML scenarios due to their ability to perform efficient parallel computations, which is advantageous for accelerating machine learning tasks – especially in applications that require real-time data processing. Additionally, by combining sensors, signal processing, and ML models directly in the hardware, the devices may process data and make decisions locally without relying on cloud-based computations [5], [4].

1.2 Problem Description

The core issue in radar detection is the trade-off between robustness and accuracy in environments containing high levels of noise and clutter, as well as potential jamming signals. In such environments, potentially mismatched signals are likely, and it is crucial to ensure that the detector maintains a constant false-alarm rate. For defense systems, it is essential that the radar detection is fast and accurate, as it needs to identify and react to incoming threats, such as ballistic missiles or drones, in a very short time. Additionally, radar echoes from some threats have very low signal strength, which makes detection capability in low signal-to-noise ratio (SNR) environments important as well. The challenge arises when the actual signal does not match the expected signal model, or *steering vector*, of the target because of inaccuracies in the model assumptions.

Since radar systems are deployed in various environments, the characteristics of the noise and clutter will differ greatly, and it is therefore desirable that the systems can adapt to these varied conditions and produce reliable results. A tunable detector based on the CFAR-FP framework in an embedded system could, in theory, be an adaptable and reliable detector out in the field without the need for external computers to analyze the incoming radar signals. Furthermore, embedded AIEs leverage the parallelism of the FPGA architecture. Since NNs are intrinsically parallel structures, this will significantly accelerate the computations of the NN compared to traditional CPUs or even GPUs. Thus, any loss in precision due to quantization could possibly be outweighed by the increased efficiency with reduced latency and power consumption. It also eliminates the need to transmit the data to external sources for postprocessing. The main drawbacks are storage size and computational accuracy, since the inputs to the FPGA are limited by a certain number of bits.

1.3 Related Work

The theoretical framework for this thesis is based on prior research regarding radar systems, CFAR detectors, and the CFAR-FP as well as ML models for similar classification problems.

The CFAR-FP was first proposed in [1], where theoretical analysis and simulations proved that it can be used as an intuitive tool in certain radar detection applications. In addition, [6] includes a more comprehensive overview of adaptive detectors and the CFAR-FP. These sources constitute the theoretical foundation of this work.

As for the hardware implementation of the CFAR-FP, we rely on previous published designs for complex-valued linear algebra operations implemented on FPGAs. The main source for this is a proposed design method for matrix inversion with complex floating-point numbers [7]. This work covers many needed operations for the CFAR-FP mapping, such as complex multiplication, division, and matrix inversion which provided great insight for handling complex numbers in hardware.

A prior study [8] has demonstrated the effectiveness of edge AI in low-power forest monitoring systems, where real-time fire detection is achieved without relying on

cloud-based infrastructure. Similar edge-driven principles are found in some military radar systems. In [9], a maritime domain application is proposed where edge machine learning enabled underwater drones and robots to locally process sensor data and make autonomous decisions, which is essential for time-sensitive and safety-critical tasks such as sea mine detection, where dependence on remote human control is often unsafe or impossible. In defense applications, edge devices' ability to process data and act locally even in scenarios where communication channels might have been disrupted could be extremely valuable.

To get an understanding of how NNs can be implemented on FPGA platforms we used a master's thesis conducted at Saab Surveillance in 2021 which explores this topic [10]. That work also used the VCK190 evaluation board [11] that is used in this thesis, which made it highly relevant as a source and baseline for the project.

1.4 Purpose and Goal

The purpose of this thesis is to investigate the feasibility of implementing a radar detector from an NN classifier within the CFAR-FP framework as an alternative to traditional radar detection thresholding methods. With an NN-based approach, we explore the capability of defining an optimal threshold based on a training dataset generated from a ground-based rotating experimental radar.

As data volumes in radar systems continue to increase, the demand for efficient and high-speed signal processing systems has become more critical. This work aims to explore the implementation of such a system in hardware, specifically targeting FPGAs, in order to achieve low-latency processing without compromising detection performance too severely.

The primary goal is to explore the potential of NN-driven thresholding for high-speed radar signal processing applications in edge devices. We will develop and evaluate a prototype radar detector based on the CFAR-FP concept, where an NN will be trained to set the optimal decision boundary within the plane. Furthermore, for full edge capabilities we aim to implement the CFAR-FP mapping chain on the hardware. A complete system aims to be implemented on Xilinx Versal AI Core series FPGA VCK190, leveraging its processing capabilities for real-time operation.

To reach this goal, the project is divided into several intermediate milestones:

1. **System model:**
   Develop a complete CFAR-FP detector system in software, including CFAR-mapping based on radar data with configurable SNR and mismatch levels as well as the integration of a NN for adaptive thresholding.
2. **CFAR-FP in hardware:**
   Implement the CFAR-FP mapping-chain within an FPGA.
3. **NN in hardware:**
   Investigate how the NN model trained in floating point format performs after quantization.

4. **Complete system in hardware:**
   Integrate the CFAR-FP mapping-chain hardware design with the NN into a full system implementation on the VCK190 board.

These milestones will serve as the building blocks toward achieving a high-speed, hardware-accelerated radar detection system based on modern machine learning techniques.

1.5 Delimitations

The goal is to test if an implementation of a real-time target detector, combining the CFAR-FP detection with an NN detection approach is possible on an FPGA. Further performance optimization such as resource utilization, data rate, or power dissipation are secondary and will not be the main focus of the project.

There exist multiple different CFAR detectors. However, when comparing the performance of the proposed tunable detector, we use Kelly's detector (see Section 2.3.4) for benchmarking as it is a foundation for many other detectors as well as rather simple to implement in the CFAR-FP.

In order to train the NN we need accurate labels on huge datasets with many targets of different characteristics. As it is difficult to verify with certainty from the raw radar data where targets were located – as well as the limited amount of targets available – we will make the assumption that the data mainly consist of a noise and clutter background. Artificial targets are instead injected using an ideal steering vector model into the collected radar data, as it simplifies the labeling of the data for classification purposes. This enables full control of the expected and measured targets in order to accurately generate desired signal mismatches and SNR. However, this means that the proposed system will not be tested on real targets.

Finally, the data used is collected only from one beamformed channel, which means we will not account for spatial dependencies in the target steering vector, only temporal. The experimental radar data is preprocessed by matched filtering, which means we will not perform any pulse compression as part of our design.

1.6 Use of LLMs

Large language models (LLMs), such as ChatGPT, were used conservatively in certain aspects of this project. The main usage included:

- asking for recommendations for academic literature when researching the theoretical parts of the project,
- help with debugging Python code,
- speeding up the process of configuring generated plots in MATLAB and Python,
- formatting equations and references in LaTeX,
- minor grammatical fixes in this report, such as correct usage of commas, hyphens, and semicolons.

LLMs took no part in the method, design, or overall structure of this work.

1.7 Thesis Outline

After the introduction, this report starts off with an overview of the technical background of this project. The first sections of this chapter provide the reader unfamiliar with pulsed radar systems with the fundamental theory required to understand the radar signal processing methods and CFAR-FP presented in the following sections. The chapter ends with sections covering the architecture and functionality of supervised NNs. In Chapters 3–4 we present our approach to implement the proposed system, with sections covering system modeling in MATLAB and Python as well as the hardware implementation. The results of the project are summarized in Chapter 5, which are then evaluated and discussed in Chapter 6. Finally, our conclusions drawn from the results, as well as our ideas of possible future research, are detailed in Chapter 7.

2

Technical Background

This chapter provides an overview of the technical and theoretical framework, which was needed in order to motivate the design choices made in the project. The following sections cover the basics of radar signal processing and detection theory, the mapping chain of the CFAR-FP, as well as the architecture of NNs.

2.1 Pulse-Doppler Radar

A pulsed-Doppler radar system transmits high-energy electromagnetic (EM) waves as pulses and receives their respective echoes after they have been scattered by objects in the environment. These EM pulses are typically modulated with local oscillators (LOs) using mixers to radio-frequency (RF) bands. This modulation results in narrow-band signals which provide several benefits.

The antenna's area determines the effective absorption area of a received plane wave [12, p. 151], and is directly proportional to the squared wavelength of the transmitted pulses. The wavelength is given by

$$\lambda_{\mathrm{c}} = \frac{c_0}{f_{\mathrm{c}}}, \tag{2.1}$$

where c_0 is the speed of light in vacuum and f_{c} the carrier frequency of the pulse. Thus, if low-frequency pulses are used, the antenna required for effective reception would require a diameter in the scale of kilometers, which of course is not realistic. Typically, the assumption for antennas is $\lambda_{\mathrm{c}} \ll D$, where D is the diameter of the antenna [13, p. 11]. As will be described later in this section, the minimum and maximum detectable radial velocity of a target are also dependent on the wavelength.

Additionally, the receiver uses in-phase and quadrature (IQ) mixers to demodulate the received signal to baseband. This produces two channels: one real and one imaginary which are out of phase by 90°. By expressing the signals as complex phasors using Euler's identity, mathematical operations performed during signal-processing steps are greatly simplified – as multiplication of sine and cosine functions are reduced to addition of exponents of exponentials. Furthermore, the IQ-demodulation provides phase information of the received signal which allows for Doppler frequency processing to determine the velocity of targets. We will return to this subject later in this section.

A simplified block diagram of a radar system model is shown in Fig. 2.1.

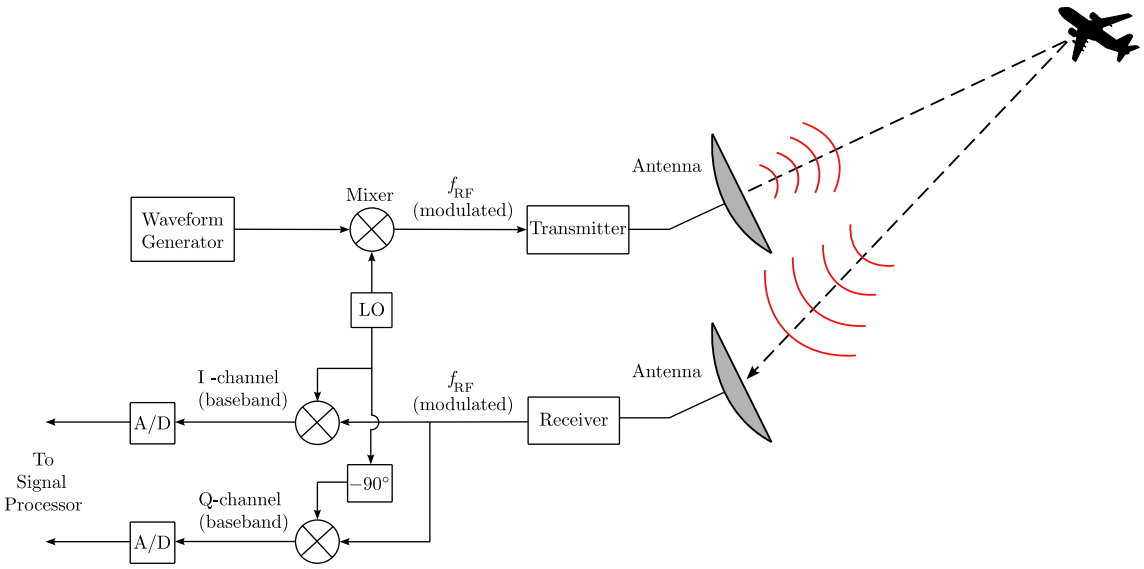


Figure 2.1: Simplified radar system block diagram.

Note that the figure shows one antenna for transmitting and another for receiving the radar echoes. This is only the case for bistatic radars. Monostatic radars instead use the same antenna for both transmission and reception with a control circuit, such as a duplexer, which switches between the two modes [13, p. 35]. Furthermore, other blocks such as amplifiers and intermediate mixers with different oscillators have been omitted from Fig. 2.1 for simplicity.

After the analog-to-digital converters (ADCs) a matched filter (MF) is commonly used to pulse compress the received signal by convolving it with the time-reversed complex conjugate of the transmitted signal; effectively cross-correlating the two signals. This allows for wider pulse width which maximizes the SNR, while still maintaining a fine range resolution otherwise only obtainable by a short pulse width [13, pp. 113–120].

2.1.1 Range

In pulsed radar systems the pulses are characterized by a certain peak power, pulse width, τ, and a pulse repetition interval (PRI), t_p [13, p. 9], as shown in Fig. 2.2. The reciprocal of the PRI is called the pulse repetition frequency (PRF). The relative time delay and phase difference of the transmitted and received pulses allow for measurements of the distance and velocity of the objects. Given a phase velocity of the transmitted wave (often approximated to c_0 in free-space), the time delay Δt between transmission of the leading edge and reception of the trailing edge of a single pulse is

$$\Delta t = \frac{2R}{c_0} + \tau, \tag{2.2}$$

where R is the distance to the target.

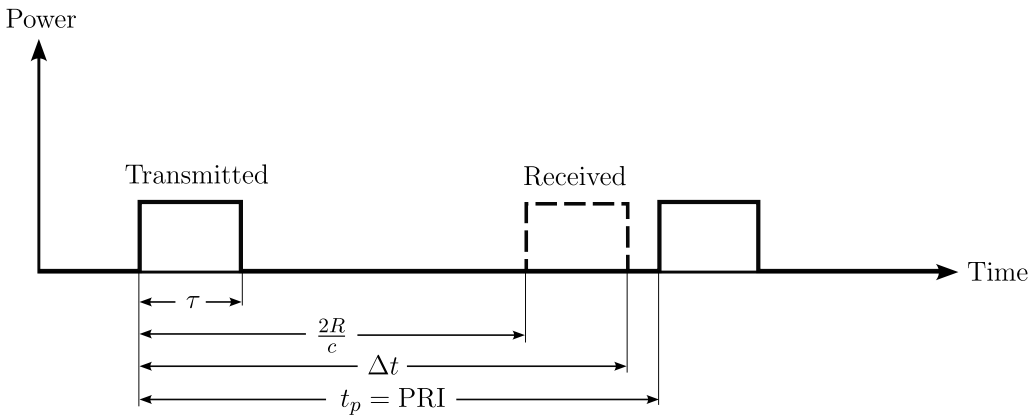


Figure 2.2: Transmitted and received radar pulses.

Since c_0 is a constant, and $\tau \ll \frac{2R}{c_0}$, the range can easily be calculated by rewriting (2.2). The maximum detectable range before the next pulse is sent is known as the *maximum unambiguous range*, and is given by

$$R_{\text{u}} = \frac{c_0(t_{\text{p}} - \tau)}{2} \approx \frac{c_0 t_{\text{p}}}{2}, \tag{2.3}$$

and the range resolution, δ_R, is determined by the pulse width:

$$\delta_R = \frac{c_0 \tau}{2}. \tag{2.4}$$

2.1.2 Velocity

As previously mentioned, using IQ-mixers for the demodulation at the receiver enables measurements of the relative phase of the received pulses and a stable LO (STALO) within the radar [13, p. 20]. Radars with this capability are known as *coherent* radars. A pulse reflected off a target located at range R will have its phase, φ, delayed by

$$\Delta\varphi = -\frac{4\pi R}{\lambda_{\text{c}}}. \tag{2.5}$$

If the target is stationary, $\Delta\varphi$ will be constant over multiple transmitted pulses. However, if the target is moving with a constant line-of-sight (LOS) velocity towards or away from the radar, the relative phase of each consecutive pulse will have a constant rate-of-change, or *frequency*. This phenomenon is known as a Doppler shift, or Doppler frequency. The Doppler frequency, f_{d}, relates to the LOS velocity of the target, v, as

$$f_{\text{d}} = -\frac{2v}{\lambda_{\text{c}}}. \tag{2.6}$$

The negative sign in front of the expression is a result of the fact that the LOS velocity can be both positive or negative, with positive velocity indicating that the target is moving away from the radar[1].

[1]This convention differs in some sources, where negative velocity instead represents the target moving away from the radar. In those cases, (2.6) does not include the negative sign.

The Doppler frequency is measured by computing the fast Fourier transform (FFT) of M received pulses. The maximum Doppler shift that can be measured is determined by Nyquist sampling theorem:

$$f_{\mathrm{d,max}} = \frac{\mathrm{PRF}}{2} = \frac{1}{2t_{\mathrm{p}}}, \tag{2.7}$$

which gives a maximum unambiguous velocity, v_{u}, of

$$v_{\mathrm{u}} = \pm\frac{\lambda_{\mathrm{c}}}{2t_{\mathrm{p}}}. \tag{2.8}$$

The frequency resolution, δ_f, is of course determined by the minimum detectable rate of change in phase, and is dependent on both M and the PRI as

$$\delta_f = \frac{1}{Mt_p}, \tag{2.9}$$

which yields a velocity resolution, δ_v, of

$$\delta_v = \frac{\lambda_{\mathrm{c}}}{2Mt_{\mathrm{p}}}. \tag{2.10}$$

The product Mt_{p} is known as the dwell time, or coherent processing interval (CPI). Clearly, selecting a long CPI is beneficial in terms of resolution but it will require significantly more computational resources to process the data. The minimum CPI required depends on the application, although selecting a value of base two is preferable as it allows for efficient FFT computations.

2.2 Multichannel Processing

In the previous section we described how range and velocity measurements are performed using pulsed-Doppler radar and a single antenna element, where each received echo from a certain direction of arrival (DOA) is characterized by its reception time. A further improvement frequently applied in modern radar systems is to use multiple antennas – so-called *subapertures* – to measure incoming radar echoes. The received signal is then not only characterized by the reception time and its phase, but also the subapertures' physical location in space [13, p. 349]. This way of simultaneously processing a set of spatial samples from multiple subapertures and temporal samples from multiple pulses within a CPI is known as *multichannel processing* (MCP) and is utilized in, for example, *space-time adaptive processing* (STAP) in airborne radars.

Compared to single-aperture processing, STAP provides improved detection performance of targets obscured by jamming signals or clutter [14]. This is particularly important in airborne radar systems, where the radar platform's motion may cause the clutter to be spread out as a narrow ridge across the entire Doppler space [14, Fig. 1]. The way this is achieved is by computing adaptive weight vectors based on interference estimations to filter out jamming signals and stationary ground clutter [14].

It should come as no surprise that these benefits provided by MCP make it a foundation in many modern radar signal processing applications – with this work being no exception. In this section, we will lay out the main framework required for MCP which is relevant for this project. For a more in-depth explanation of MCP and STAP we refer the reader to [13, Ch. 12], [15].

2.2.1 Array antennas

Antennas with multiple subapertures are often referred to as *array antennas*. In many applications it is desirable to also be able to control the phase of the radiating elements. It is then possible to steer the beam direction to a desired location without physically moving the antenna [13, pp. 53–54]. These types of antennas are known as *phased-array antennas*, and are crucial in beamforming applications.

Multiple different architectures for phased-array antennas exist, but we will here only consider the simple uniform linear array (ULA) antenna for demonstration purposes. A line array consists of N subapertures located a distance d apart[2], as shown in Fig. 2.3.

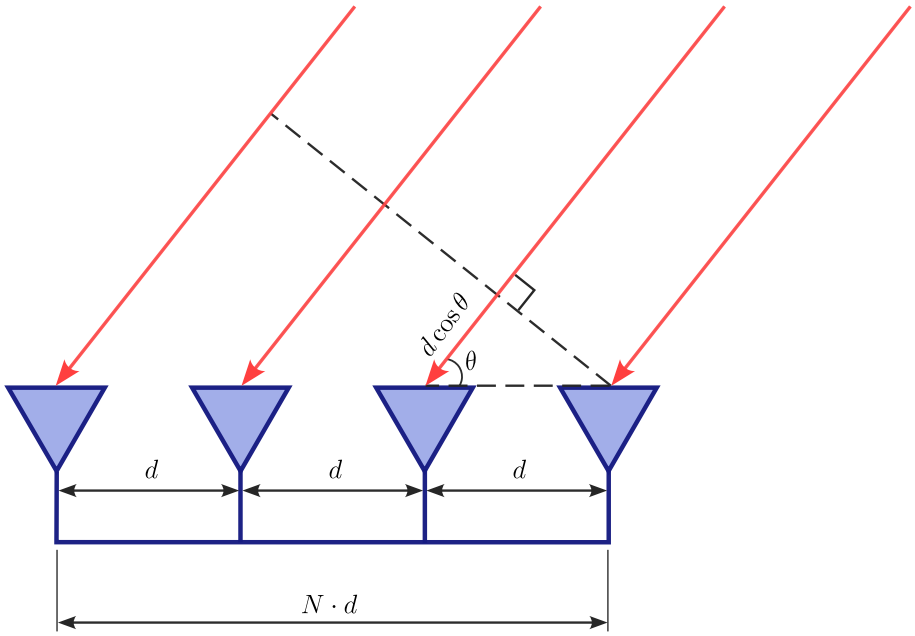


Figure 2.3: Uniform linear array antenna with N radiating elements.

Each subaperture receives an incoming radar echo (red arrows in Fig. 2.3) at the same incidence angle, θ. For aperture n, where $n = 0, 1, \ldots, N-1$, the radar pulse will have to travel $nd\cos\theta$ longer than the right-most subaperture ($n = 0$). This longer path will result in a time delay between each subaperture's reception of the same pulse, which provides information regarding the pulses spatial frequency associated with the angle of arrival (AOA) [15]. Thus, as opposed to the single

[2]Typically, this distance is defined as $d \leq \lambda_c/2$ [13, p. 56].

element radar system we assumed in Section 2.1, the received pulses for array antennas contain both temporal and spatial dependencies related to the DOA and AOA respectively.

2.2.2 Radar data cube

In array processing, a CPI of M pulses is transmitted and collected from N subapertures. Each pulse is sampled at a very high sampling rate compared to the PRI (often in the scale of GHz) and pulse compressed into L discrete *range bins*. This high sampling rate is why this dimension is often referred to as *fast time*, and the pulse dimension is referred to as *slow time* [13]. The difference in range between two adjacent range bins is given by the range resolution in (2.4).

In other words, for each CPI we have LMN complex radar echoes, which can be visualized as a three-dimensional (3-D) data cube, as depicted in Fig. 2.4. The squares marked in blue constitute an $M \times N$ data matrix, typically restructured for signal processing as an $MN \times 1$ vector referred to as a *cell*.

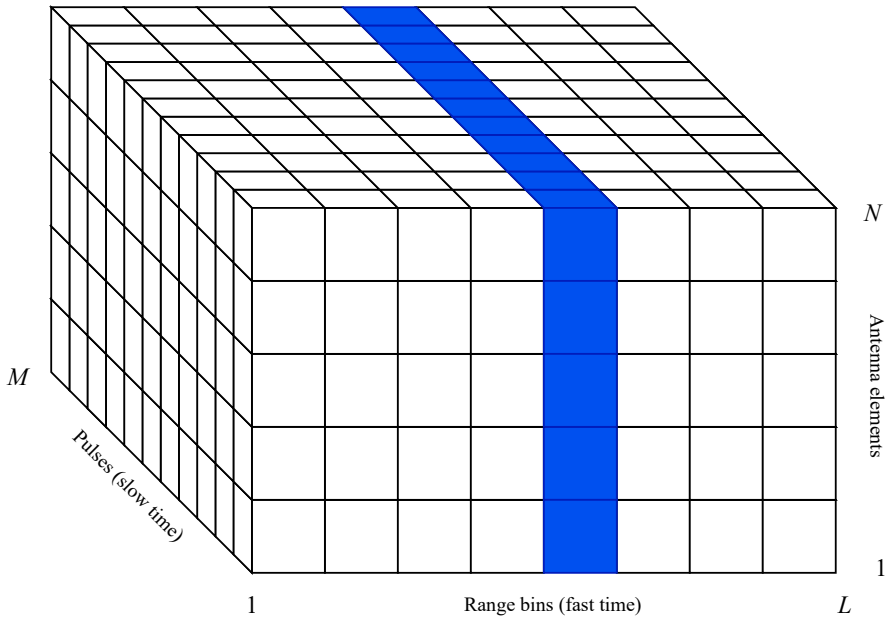


Figure 2.4: CPI data cube.

2.3 Target Detection

Radar detection fundamentally involves determining whether a target is present in a radar echo, or if the signal only contains noise and clutter. Thus, any radar detector should output a binary decision given an observed radar data cell – a so-called *cell under test* (CUT). Either the target is present in the signal, or it is not. In radar signal processing theory, this is typically expressed as a binary hypothesis test. The null-hypothesis (target is not present in signal) is denoted by H_0, and the alternative hypothesis (target is present in signal) is denoted by H_1 [6, p. 5]. These two hypotheses can be defined as

$$\begin{cases} H_0 : \boldsymbol{z} = \boldsymbol{n} \\ H_1 : \boldsymbol{z} = \alpha \boldsymbol{v} + \boldsymbol{n}, \end{cases} \tag{2.11}$$

where $\boldsymbol{z} \in \mathbb{C}^{MN \times 1}$, $\boldsymbol{n} \in \mathbb{C}^{MN \times 1}$, and $\boldsymbol{v} \in \mathbb{C}^{MN \times 1}$ are the column vectors representing the CUT, noise, and signal steering vector respectively [1]. Note that $\boldsymbol{n}$ includes all disturbances such as thermal noise, radar clutter, and possible jamming signals. The scalar $\alpha \in \mathbb{C}$ represents an amplitude and phase shift caused by the physical channel. This parameter depends on the physical constraints of the radar system and the characteristics of the scattering object such as radar cross-section (RCS), transmitter antenna gain, and multi-path channel effects. Dimension MN is the product of the number of antenna array elements and the number of transmitted pulses within a CPI in order to include both spatial and temporal dependencies [14].

A complete decision rule must then be defined, wherein each observed $\boldsymbol{z}$ can be assigned to either H_0 or H_1. This classification of the observations is never perfect, and there will always be a risk for false alarms – meaning that a given observation will incorrectly classify interference as a target. Solving the hypothesis test in (2.11) is done by applying a threshold, η, to some decision statistic of the receiver output, t as a function of $\boldsymbol{z}$:

$$t \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{2.12}$$

As emphasized in [12, p. 266], most radar detectors perform this comparison of the receiver output with some threshold η in one way or another when applying the hypothesis test in (2.11). Methods for determining η is therefore a heavily researched topic and a fundamental part of this thesis, which will be revisited several times throughout this report.

2.3.1 Energy detector

A possible and simple decision statistic is to measure the energy of a received signal $z(t)$ after matched filtering and comparing it with a threshold. This type of detector is known as an *energy detector* (ED) [6, pp. 9–10]. Assuming that the noise and clutter, $n(t)$, and useful signal, $s(t)$, are linearly separable, the energy of $z(t)$ can be expressed as

$$E(z) = \begin{cases} E(n) & , \text{if } H_0 \\ E(s) + E(n) & , \text{if } H_1 \end{cases} \tag{2.13}$$

The energy of a continuous signal is given by $E(z) = \int_{-\infty}^{+\infty} |z(t)|^2 \, dt$, which for a discrete system containing M samples of the analog signal $z(t)$ can instead be described by $E(z) = \sum_{i=1}^{M} |z_i|^2$. These M samples can then be conveniently expressed as a vector for the two hypotheses as in (2.11). The energy of the vector is simply the squared magnitude, and the decision rule for the ED can thus be defined as

$$||\boldsymbol{z}||^2 = \boldsymbol{z}^\dagger \boldsymbol{z} \underset{H_0}{\overset{H_1}{\gtrless}} \eta, \tag{2.14}$$

where η is some threshold value and $\boldsymbol{z}^\dagger$ denotes the complex conjugate transpose, or Hermitian, of $\boldsymbol{z}$. Eq. (2.14) highlights why the ED is also often referred to as a *square-law detector* [6, p. 11]. Achieving a desired probability of false alarm is done by setting η to an appropriate value.

2.3.2 Neyman-Pearson approach

With a probabilistic model of the noise and useful signal defined, by denoting $\mathcal{R}_1$ as the M-dimensional region wherein all observations $\boldsymbol{z}$ that have been assigned to H_1 lie, the probability for detection, P_D, and the probability of false alarm, P_FA, can be described as the integrals of the joint probability density functions (PDFs) over $\mathcal{R}_1$ [16]:

$$\begin{aligned} P_\mathrm{D} &= \int_{\mathcal{R}_1} p_{\boldsymbol{z}}(\boldsymbol{z}|H_1) \, d\boldsymbol{z} \\ P_\mathrm{FA} &= \int_{\mathcal{R}_1} p_{\boldsymbol{z}}(\boldsymbol{z}|H_0) \, d\boldsymbol{z}, \end{aligned} \tag{2.15}$$

where $p_{\boldsymbol{z}}(\boldsymbol{z}|H_1)$ is the PDF of $\boldsymbol{z}$ given that a target was present, and $p_{\boldsymbol{z}}(\boldsymbol{z}|H_0)$ is the PDF of $\boldsymbol{z}$ given that a target was absent.

Fig. 2.5 shows an example of the PDFs of a simple radar detector which compares the voltage level of the measured signal with some threshold voltage, V_T. The PDF of the measured voltage given that the target was absent is denoted by $p_0(V)$ and the PDF given that the target was present is denoted by $p_1(V)$. As can be seen, the P_FA and P_D are indeed the integrals of $p_0(V)$ and $p_1(V)$ respectively, evaluated in region $\mathcal{R}_1 \in [V_T, +\infty)$.

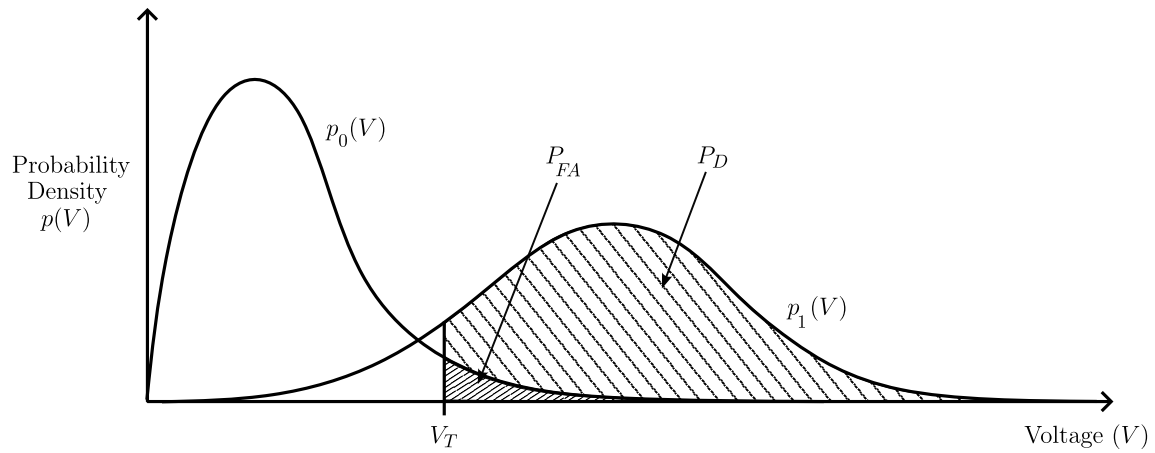


Figure 2.5: Probability density functions of $p_0(V)$ and $p_1(V)$.

Since PDFs are strictly positive, (2.15) and Fig. 2.5 lead to the conclusion that P_{D} and P_{FA} must increase or decrease concomitantly if the region $\mathcal{R}_1$ grows or shrinks (V_T is lowered or increased). Given this fact, a popular detection rule is often applied known as the Neyman-Pearson criterion (NP) [16]. This approach aims to find the threshold η for a detector, as in (2.14), that achieves the highest possible P_{D} while ensuring that the P_{FA} never exceeds a predefined acceptable value, ψ. The NP rule can therefore be viewed as selecting the region $\mathcal{R}_1$ in such a way that P_{D} is maximized, under the constraint $P_{\mathrm{FA}} \leq \psi$.

Maximizing P_{D} under a given constraint can be done by using Lagrange multipliers [16]. Define the Lagrangian for maximizing P_{D}, constrained by P_{FA} as a function F, given by

$$\mathcal{L}(\boldsymbol{z}, \psi) = F \equiv P_{\mathrm{D}} + \lambda \cdot (P_{\mathrm{FA}} - \psi). \tag{2.16}$$

Maximizing F and then choosing the multiplier λ for which the constraint $P_{\mathrm{FA}} = \psi$ is satisfied guarantees the optimum solution[3]. Substituting (2.15) into (2.16) gives

$$\begin{aligned} F &= \int_{\mathcal{R}} p_{\boldsymbol{z}}(\boldsymbol{z}|H_1)\, d\boldsymbol{z} + \lambda \left(\int_{\mathcal{R}} p_{\boldsymbol{z}}(\boldsymbol{z}|H_0)\, d\boldsymbol{z} - \psi \right) \\ &= -\lambda\psi + \int_{\mathcal{R}} p_{\boldsymbol{z}}(\boldsymbol{z}|H_1) + \lambda p_{\boldsymbol{z}}(\boldsymbol{z}|H_0)\, d\boldsymbol{z}. \end{aligned} \tag{2.17}$$

The only variable which can be defined arbitrarily is $\mathcal{R}_1$. Maximizing the integral over region $\mathcal{R}_1$ maximizes F as a result. The sign of the integral will depend on the sign of λ, $p_{\boldsymbol{z}}(\boldsymbol{z}|H_0)$, and $p_{\boldsymbol{z}}(\boldsymbol{z}|H_1)$. F is maximized if one selects only the points in $\mathcal{R}_1$ for which the integral will be greater than zero:

$$\begin{aligned} & p_{\boldsymbol{z}}(\boldsymbol{z}|H_0) + \lambda p_{\boldsymbol{z}}(\boldsymbol{z}|H_1) > 0 \\ \Leftrightarrow\ & p_{\boldsymbol{z}}(\boldsymbol{z}|H_1) > -\lambda p_{\boldsymbol{z}}(\boldsymbol{z}|H_0). \end{aligned} \tag{2.18}$$

Rewriting (2.18) leads to the decision rule known as the *likelihood ratio test* (LRT):

$$\frac{p_{\boldsymbol{z}}(\boldsymbol{z}|H_1)}{p_{\boldsymbol{z}}(\boldsymbol{z}|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} -\lambda. \tag{2.19}$$

The LRT states that if, and only if, the ratio of the two PDFs evaluated at an observation $\boldsymbol{z}$ exceeds a threshold determined by λ should $\boldsymbol{z}$ be classified as hypothesis H_1 – otherwise hypothesis H_0 should be selected. Note that the LRT can be expressed as (2.12), with a decision statistic $t = \frac{p_{\boldsymbol{z}}(\boldsymbol{z}|H_1)}{p_{\boldsymbol{z}}(\boldsymbol{z}|H_0)}$ and a threshold $\eta = -\lambda$.

As demonstrated in [6, p. 11], assuming σ_{n}^2 and the signal power σ_{s}^2 are known, the LRT can be applied to the presented square-law detector in (2.14):

$$\frac{||\boldsymbol{z}||^2}{\sigma_{\mathrm{n}}^2} \underset{H_0}{\overset{H_1}{\gtrless}} \eta, \tag{2.20}$$

where η now is a modification of the original threshold.

[3]Note that the use of λ here is the conventional notation of the Lagrangian multiplier and should not be confused with the wavelength λ_{c}.

2.3.3 CFAR

The NP approach previously discussed shows that the probability of detection can be maximized if the probability of false alarm is constrained to a constant value, ψ. If the P_{FA} for a given detector is also independent of any parameters in the H_0 model, the detector is said to have the CFAR property.

From (2.15) and (2.12), we know that the detection probability depends on the threshold η. The simplest approach when designing a detector is to set a constant threshold. This will result in a trade-off between P_{D} and P_{FA} as described in Section 2.3.2. However, this leads to a significant problem. If the interference level and characteristics are known, setting an appropriate threshold to ensure a constant false-alarm rate would be trivial using the NP rule. Unfortunately, in real-world radar applications the noise and clutter are rarely known beforehand. Although it is possible to define a model for H_0 and then design detectors with the CFAR property under that model assumption, these models are not always a good predictor of the conditions under H_0. As a result, it is difficult, if not impossible, to guarantee the CFAR property if η is fixed.

To illustrate this, Fig. 2.6 shows an example plot of the normalized received power across different range bins for a specific pulse index together with a fixed threshold for an ED detector. Targets have been injected at range bins 355, 519, and 684, and are all clearly above the threshold. However, several false alarms are also present. Evidently, an adaptable threshold is desired.

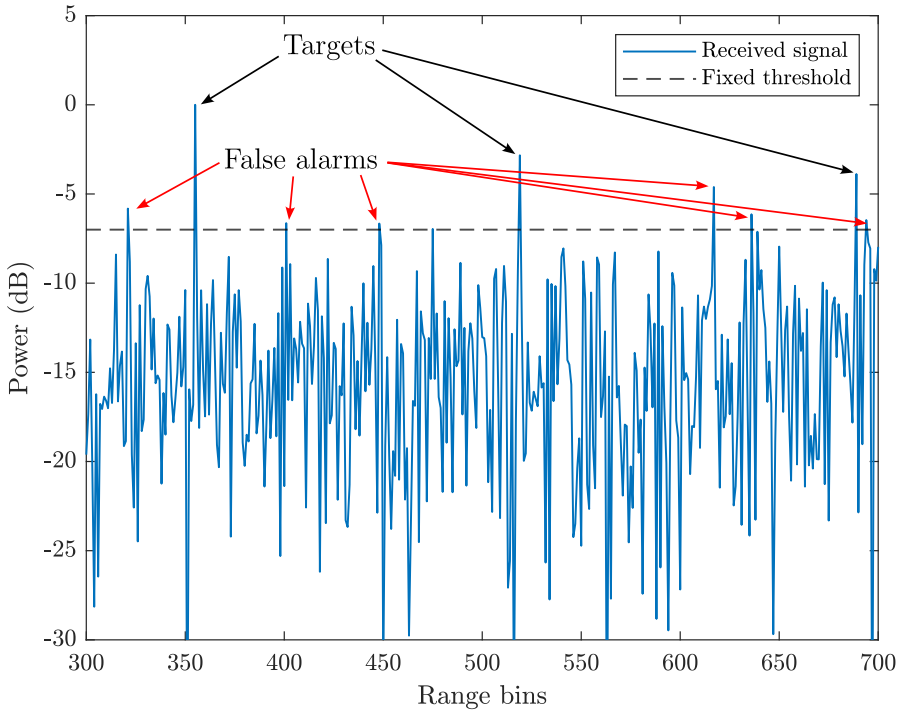


Figure 2.6: Normalized range-power plot of one pulse per range bin with fixed threshold. Targets injected at range bins 355, 519, 684.

In (2.20), the LRT is applied to the ED assuming σ_n^2 is known, which achieves the desired CFAR property. In the common case, where the interference characteristics are not known completely, the interference can instead be approximated by measuring neighboring cells surrounding the CUT, with the assumption that only the CUT contains a target and that the neighboring cells only contain independent and identically distributed noise [6, p. 13]. These cells are often referred to as *secondary data* and are selected from K range bins surrounding the CUT.

The size of K depends on the desired performance. We want K to be high enough to get a good estimation of the noise, but not too high as that would violate the homogeneity assumption [17]. In Reed, Mallet, and Brennan's (RMB) pioneering work [18], it was proven that given a CPI of M pulses, setting $K = 2M$ will achieve a minimum 3 dB SNR loss compared to the optimum case of a known noise covariance matrix. This widely accepted rule-of-thumb is, unsurprisingly, referred to as the RMB rule. To ensure the secondary data cells only contain noise, a few guard cells are also added on each side of the CUT in case the observed target stretches over multiple range bins, as this would otherwise heavily skew the noise estimation [16].

Fig. 2.7 illustrates how the CUT, guard cells and secondary data cells are selected, given a single antenna element from the data cube shown in Fig. 2.4.

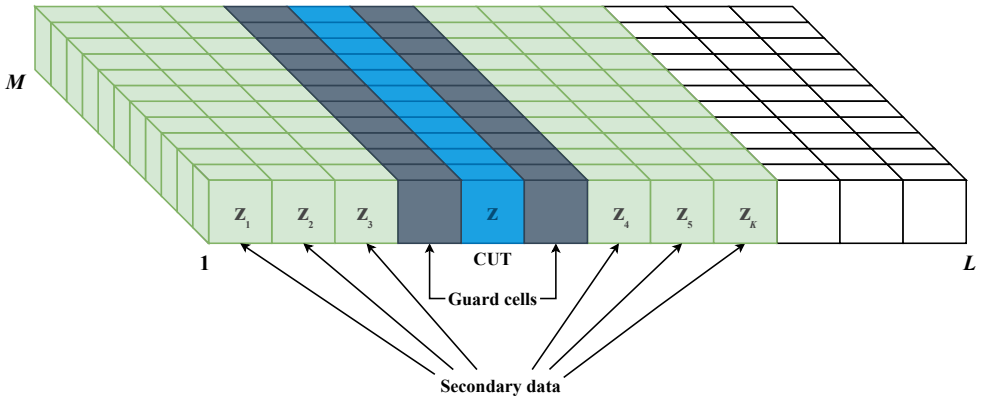


Figure 2.7: $M \times L$ data matrix showing CUT, guard cells, and secondary data.

The estimated noise power, $\hat{\sigma}_n^2$, from secondary data cells $\boldsymbol{z}_k$, $k = 1, 2, ..., K$ is calculated as:

$$\hat{\sigma}_n^2 = \frac{1}{K} \sum_{k=1}^{K} |\boldsymbol{z}_k|^2. \tag{2.21}$$

Substituting σ_n^2 with $\hat{\sigma}_n^2$ in (2.20) the decision statistic for the adaptive ED can thus be set as [6, p. 13]:

$$\frac{||\boldsymbol{z}||^2}{\hat{\sigma}_n^2} \equiv \frac{||\boldsymbol{z}||^2}{\frac{1}{K}\sum_{k=1}^{K} |\boldsymbol{z}_k|^2} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{2.22}$$

Rewriting (2.22) yields

$$||\boldsymbol{z}||^2 \underset{H_0}{\overset{H_1}{\gtrless}} \frac{\eta}{K} \sum_{k=1}^{K} |\boldsymbol{z}_k|^2 \Leftrightarrow t \underset{H_0}{\overset{H_1}{\gtrless}} \eta \hat{\sigma}_n^2. \tag{2.23}$$

Evidently, the adaptive threshold is simply the average magnitude of the secondary data scaled by a constant η. Note that, as opposed to (2.20), in (2.23) it is the threshold – not the decision statistic – that is adaptive. Sweeping over each range bin in the radar data matrix and computing a threshold in this manner for each CUT is known as a cell-averaging (CA) CFAR detector, and is the simplest known adaptive detector [17].

Fig. 2.8 shows the resulting adaptive threshold created by the CA-CFAR algorithm on the radar signal in Fig. 2.6.

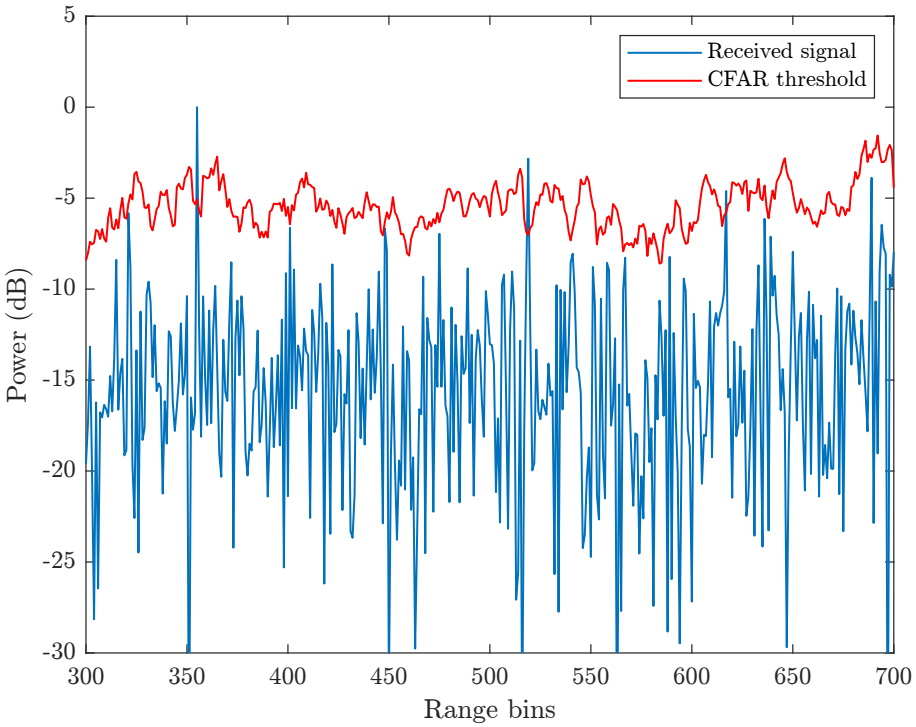


Figure 2.8: Normalized range-power plot of one pulse per range bin with CFAR threshold from cell-averaging.

2.3.4 Generalized likelihood ratio test

Setting an appropriate threshold for a given false-alarm rate requires a probabilistic model of the noise and clutter, as well as knowledge of the useful signal. In most cases, the noise can be considered to be zero-mean Gaussian, as supported by the Central Limit Theorem, with a variance (or noise power) of σ_{n}^2 [6, pp. 10–11]. In coherent radar processing, full knowledge of $\alpha\boldsymbol{v}$ is limited by the physical channel effects, expressed by α. Assuming coherent processing and complex Gaussian noise, $\boldsymbol{z}$ in the two hypotheses is distributed as [6, pp. 28–29]:

$$\begin{cases} H_0 : \boldsymbol{z} \sim \mathbb{C}\mathcal{N}(0, \boldsymbol{R}) \\ H_1 : \boldsymbol{z} \sim \mathbb{C}\mathcal{N}(\alpha\boldsymbol{v}, \boldsymbol{R}) \end{cases}. \tag{2.24}$$

Assuming an array antenna with N subapertures where the measured noise in each channel is uncorrelated, $\boldsymbol{R} = \sigma_{\mathrm{n}}^2\boldsymbol{I}$ is the unknown covariance matrix of the noise, where $\boldsymbol{I}$ is the $MN \times MN$ identity matrix [13, p. 352].

The limitations of the LRT in applications is the fact that it rarely can be applied in practice since often one or more parameters are unknown. The *generalized likelihood ratio test* (GLRT) proposed by Kelly in [19], expands the LRT to include estimations of unknown parameters within the H_0 model using *maximum likelihood estimation* (MLE). The GLRT approach can be used to perform an MLE for a decision statistic on a discrete set of estimated parameters, such as the noise and clutter, with minimal loss of performance [19]. This discovery allowed numerous new CFAR detectors with different properties to be defined [1].

The complex scalar α of the target steering vector in (2.24) can be estimated using the LRT to maximize the likelihood with respect to α. Using the GLRT it may be generalized to colored noise conditions as in (2.24). This maximum likelihood estimator, $\hat{\alpha}$, is described as [6, p. 29]:

$$\hat{\alpha} = \frac{\boldsymbol{z}^\dagger \boldsymbol{R}^{-1} \boldsymbol{v}}{\boldsymbol{v}^\dagger \boldsymbol{R}^{-1} \boldsymbol{v}}. \tag{2.25}$$

This estimation of α leads to the generalized non-adaptive MF in colored noise:

$$t_{\text{MF}} = \frac{|\boldsymbol{z}^\dagger \boldsymbol{R}^{-1} \boldsymbol{v}|^2}{\boldsymbol{v}^\dagger \boldsymbol{R}^{-1} \boldsymbol{v}} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{2.26}$$

Similarly, the ED in (2.20) may also be generalized to colored noise conditions:

$$t_{\text{ED}} = \boldsymbol{z}^\dagger \boldsymbol{R}^{-1} \boldsymbol{z} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{2.27}$$

Using GLRT, the hypothesis testing problem in (2.24) may be expanded to include the estimated noise from the secondary data cells [6, p. 30], assuming an unknown Hermitian positive-definite covariance matrix with Gaussian distribution, $\boldsymbol{R}$. The GLRT hypothesis test is described as

$$\begin{cases} H_0 \colon & \boldsymbol{z} \sim \mathbb{C}\mathcal{N}(0, \boldsymbol{R}) \\ & \boldsymbol{z}_k \sim \mathbb{C}\mathcal{N}(0, \boldsymbol{R}), \quad k = 1, \ldots, K \\ H_1 \colon & \boldsymbol{z} \sim \mathbb{C}\mathcal{N}(\alpha \boldsymbol{v}, \boldsymbol{R}) \\ & \boldsymbol{z}_k \sim \mathbb{C}\mathcal{N}(0, \boldsymbol{R}), \quad k = 1, \ldots, K \end{cases} \tag{2.28}$$

By measuring K secondary data cells, a sample covariance matrix, $\hat{\boldsymbol{R}}$, can be generated which estimates the true unknown noise covariance matrix $\boldsymbol{R}$:

$$\hat{\boldsymbol{R}} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{z}_k \boldsymbol{z}_k^\dagger = \frac{1}{K} \boldsymbol{S}, \tag{2.29}$$

where $\boldsymbol{S}$ is the Hermitian positive-definite scatter matrix [6, p. 29]. Given this definition, it has been proven that the MLE of $\boldsymbol{R}$ is exactly equal to $\hat{\boldsymbol{R}}$ [19]. Thus, the ED in (2.27) can then be expressed as an adaptive detector:

$$t_{\text{ED}} = \boldsymbol{z}^\dagger \boldsymbol{S}^{-1} \boldsymbol{z} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{2.30}$$

Note that since $\frac{1}{K}$ in (2.22) is a constant and η is an arbitrary threshold, $\frac{1}{K}$ can be incorporated into η, as has been done in (2.30).

Kelly derived his famous detector, now known as *Kelly's detector*, based on his proposed GLRT, which has become a performance benchmark for other radar detectors [1]. Kelly's detector is a *selective* receiver, ideal for specific target localization as it rejects signals differing from the expected target steering vector [6, p. 32]. It is described by the following decision statistic:

$$t_{\text{Kelly}} = \frac{|\boldsymbol{z}^\dagger \boldsymbol{S}^{-1}\boldsymbol{v}|^2}{(\boldsymbol{v}^\dagger \boldsymbol{S}^{-1}\boldsymbol{v})(1+\boldsymbol{z}^\dagger \boldsymbol{S}^{-1}\boldsymbol{z})} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{2.31}$$

Finally, this method also meant that (2.26) could be expressed as an adaptive MF (AMF) detector, first proposed by Robey et al. in [20]:

$$t_{\text{AMF}} = \frac{|\boldsymbol{z}^\dagger \boldsymbol{S}^{-1}\boldsymbol{v}|^2}{\boldsymbol{v}^\dagger \boldsymbol{S}^{-1}\boldsymbol{v}} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{2.32}$$

The AMF is a *robust* receiver, which is suitable for wide-band and low SNR conditions and therefore preferable when a radar is scanning for unknown targets as it is less sensitive to signal mismatches [6, p. 32]. The decision statistic in (2.32) performs a cross-correlation between the expected steering vector model $\boldsymbol{v}$ in the estimated noise and the CUT.

2.3.5 Steering vector

The steering vector $\boldsymbol{v}$ in (2.11) depends on the system model. For example, in the general case of STAP in airborne radar, it is possible to define both a temporal and spatial steering vector based on the data cube shown in Fig. 2.4. For a given range bin r_k, the temporal steering vector $\boldsymbol{v}_{\text{T}}$ can be defined as

$$\boldsymbol{v}_{\text{T}} = \begin{bmatrix} 1 \\ e^{j2\pi v_{\text{T}}} \\ \vdots \\ e^{j2\pi(M-1)v_{\text{T}}} \end{bmatrix} \in \mathbb{C}^M, \tag{2.33}$$

where $v_{\text{T}} = f_{\text{d}} t_{\text{p}}$ is the normalized Doppler frequency of the target in slow time [6, pp. 4–5].

A spatial steering vector, $\boldsymbol{v}_{\text{S}}$, is defined in almost the same way except that v_{T} is replaced with the spatial frequency of the target, which depends on the apertures' location in space. Thus, for N subapertures, the spatial steering vector can be described as

$$\boldsymbol{v}_{\text{S}} = \begin{bmatrix} 1 \\ e^{j2\pi v_{\text{S}}} \\ \vdots \\ e^{j2\pi(N-1)v_{\text{s}}} \end{bmatrix} \in \mathbb{C}^N, \tag{2.34}$$

where v_{S} depends on the DOA of the target, carrier frequency or array geometry. In the case of ULA antennas (see Section 2.2.1), then $v_{\text{S}} = \frac{d}{\lambda}\sin\theta$, where θ is the angle between the boresight of the antenna and the DOA [6, pp. 5–6].

In order to get the correct $MN \times 1$ dimension type as in (2.11), the *space-time steering vector* $\boldsymbol{v}$ can be formulated as

$$\boldsymbol{v} = \boldsymbol{v}_{\mathrm{T}} \otimes \boldsymbol{v}_{\mathrm{S}} \in \mathbb{C}^{MN}, \tag{2.35}$$

where $\otimes$ denotes the Kronecker product operator [6, p. 7].

For a rotating and ground-based radar, as the one used in this work, the temporal steering vector is rather different compared to the general airborne radar case. The radiation pattern of a single antenna element, is characterized by a main lobe, multiple side lobes and a beam width θ_{B} [13, p. 11], as depicted in Fig. 2.9.

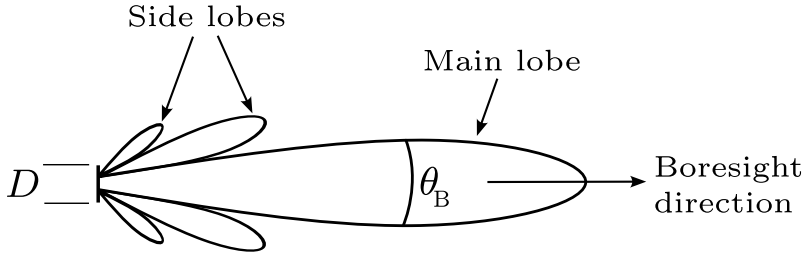


Figure 2.9: Antenna beam pattern of single radiating element with diameter D and beam width θ_{B}.

In order to simplify calculations on the radiation pattern, the main and side lobes are often approximated as a well-defined function, such as the absolute value of a sinc function or a Gaussian. The sinc function is beneficial if precise modeling of the side lobes is necessary. However, as described below, we only consider the beam width of the main lobe when defining our steering vector. Therefore, in this work we use the Gaussian approximation for simplicity. The choice of antenna model is also not crucial in our case, as we will synthetically inject targets. As long as the ideal steering vector has the same model as the measured targets, this method is valid.

Assume that for pulse m a certain target of radial velocity v is located exactly in the LOS of the antenna, meaning that the return echo is received at the center of the main lobe. If the antenna then rotates with a rotational velocity of ω_{R} rad/s, the DOA of pulse $m + 1$ will differ from the boresight direction, as illustrated in Fig. 2.10.

The change in angle, $\Delta\theta$,of the boresight relative the DOA is determined by ω_{R} and the PRI as

$$\Delta\theta = \omega_{\mathrm{R}} t_{\mathrm{p}}. \tag{2.36}$$

Thus, the angle θ can be determined deterministically for each consecutive pulse as

$$\theta_{m+1} = \theta_m + \Delta\theta. \tag{2.37}$$

As a result of this rotation, the received signal will have a varying amplitude dependent on θ, which is also Gaussian [21]:

$$g(\theta) = \sqrt{G_0} e^{-2\ln(2)\left(\frac{\theta}{\theta_{\mathrm{B}}}\right)^2}, \tag{2.38}$$

where G_0 is the maximum gain.

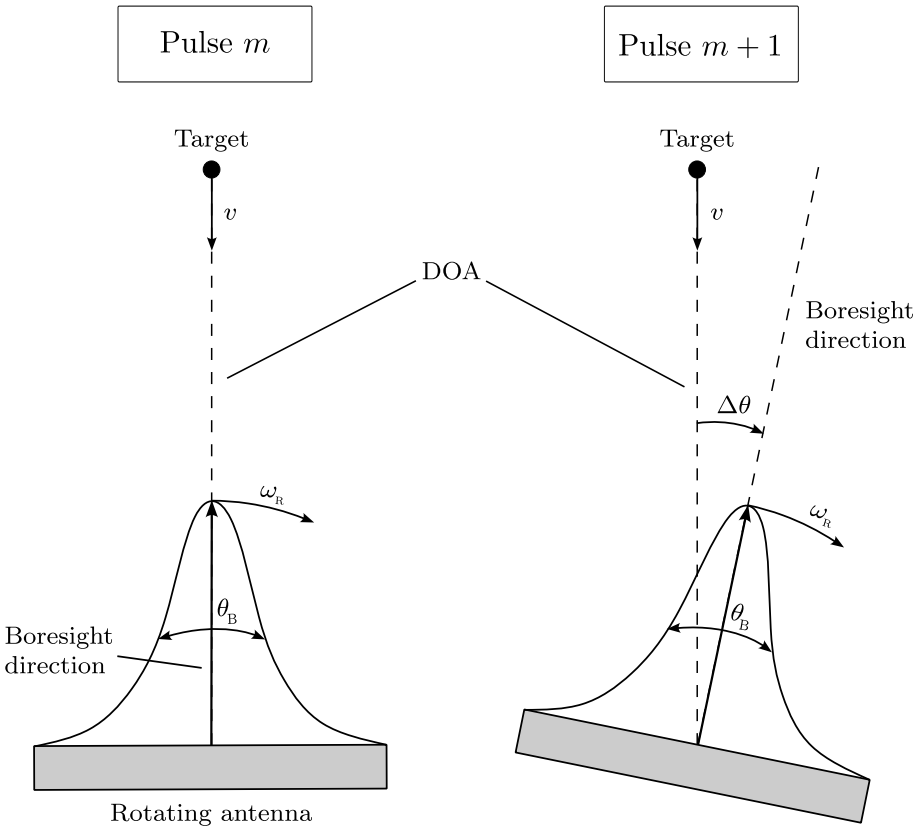


Figure 2.10: Model of rotating antenna.

The peak in the received power will occur when $\theta = 0°$, since the boresight then is perfectly in line with the DOA of the echos from the target. When instead $\theta = \pm\theta_B/2$, the received power is $g^2(\pm\theta_B) = G_0/2$ since the echoes then reach the parts of the antenna lobe constituting the $-3\,\text{dB}$ azimuth beam width.

A phase model of the target is necessary as well. The phase φ of pulse m can be determined by

$$\varphi_m = 2kvt_p m, \tag{2.39}$$

where $k = \frac{2\pi}{\lambda_c}$ is the wavenumber.

Combining (2.38) and (2.39), we can describe each element of the temporal steering vector of a target as complex phasors:

$$\boldsymbol{v} = \begin{bmatrix} g(\theta_1)e^{j\varphi_1} \\ g(\theta_2)e^{j\varphi_2} \\ g(\theta_3)e^{j\varphi_3} \\ \vdots \\ g(\theta_M)e^{j\varphi_M} \end{bmatrix}. \tag{2.40}$$

Once the antenna has rotated past θ_B, the received signal will be heavily attenuated and difficult to distinguish from the clutter and noise surrounding the target.

Therefore, it is practical to define the phase and amplitude to be centered around the middle of a CPI. It is then assumed that for a given CPI the antenna has rotated from $-\theta_B/2$ to $\theta_B/2$, with the observed target being located at exactly $\theta = 0°$. Given (2.37), this can be achieved in software simply by multiplying each integer in the range $[-M/2, M/2-1]$ with $\Delta\theta$, with the minimum M pulses required for the CPI to include the full sweep of the $-3\,\mathrm{dB}$ beam width being

$$M_{\min} = \frac{\theta_B}{\omega_R t_p}. \tag{2.41}$$

Due to this shift, the linear phase will have a dynamic swing of $\pm\frac{4\pi}{\lambda_c}\frac{M}{2}vt_p$.

Fig. 2.11 depicts the amplitude and phase plots of the described steering vector given a CPI consisting of M pulses high enough to show most of the Gaussian shape.

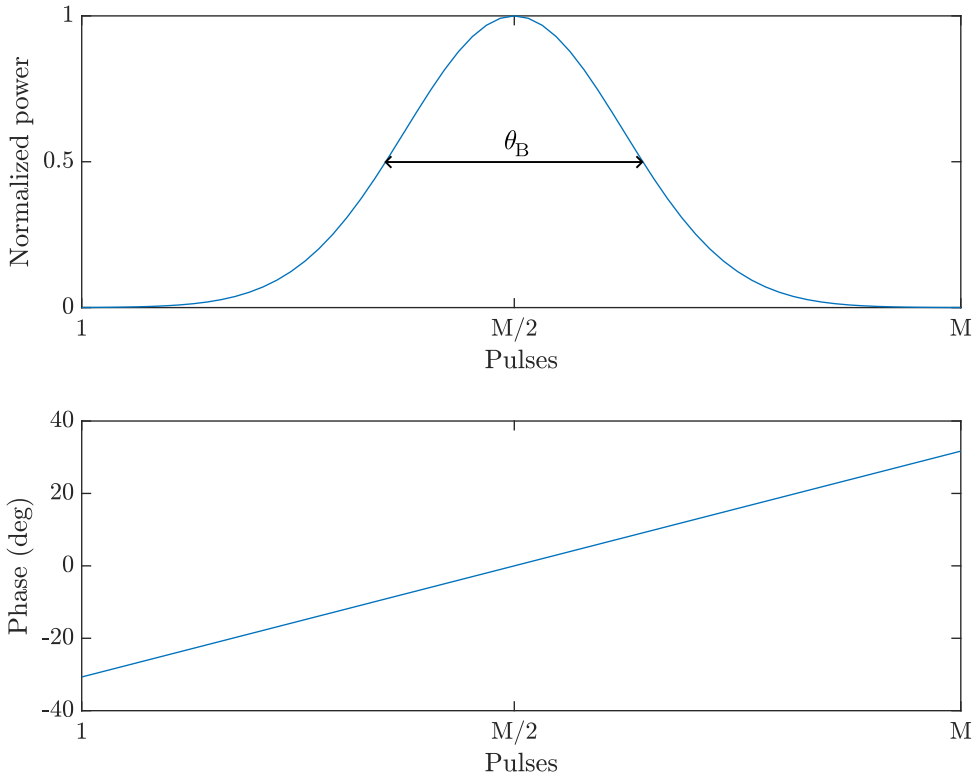


Figure 2.11: Magnitude and phase of steering vector. Note that if $M = M_{\min}$ the width of the Gaussian will be the beam width θ_B, as (2.38) is dependent on the number of pulses used when determining θ.

2.3.6 Signal mismatch

Before moving on to describing the CFAR-FP mapping chain, a more in-depth explanation as to what exactly constitutes a signal mismatch is warranted.

Consider an expected steering vector $\boldsymbol{v}$, which is estimated based on the expected pulse response of a received radar echo from a specific target given a particular radar system model, for example, the rotating antenna system explained in Section 2.3.5. Recalling the hypothesis test in (2.11), we can visualize the two hypotheses as vectors projected on a noise subspace, as shown in Fig. 2.12 [6, p. 23]. Note that the expected steering vector $\boldsymbol{v}$ is orthogonal to the noise space, since it is the ideal model of a target.

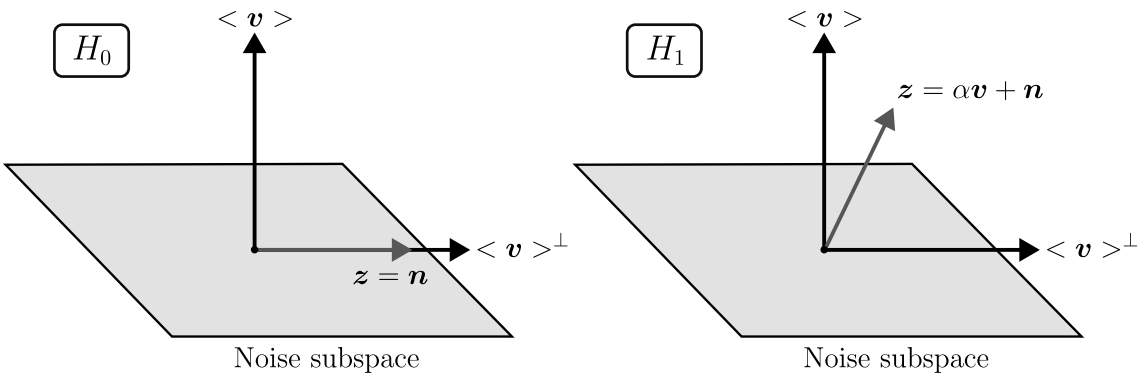


Figure 2.12: Geometrical representation of the hypothesis test in (2.11).

By denoting the measured received signal $\boldsymbol{z}$ as a complex vector $\boldsymbol{p}$, in Fig. 2.13 we see that $\boldsymbol{p}$ deviates from the expected steering vector $\boldsymbol{v}$ by an angle θ. If $\boldsymbol{p}$ contains *exactly* the same phase and amplitude information as $\boldsymbol{v}$, then $\theta = 0°$ and we have a perfect signal match. If, on the other hand, $\boldsymbol{p}$ deviates from $\boldsymbol{v}$, then $\theta \neq 0°$ and we have a signal *mismatch*.

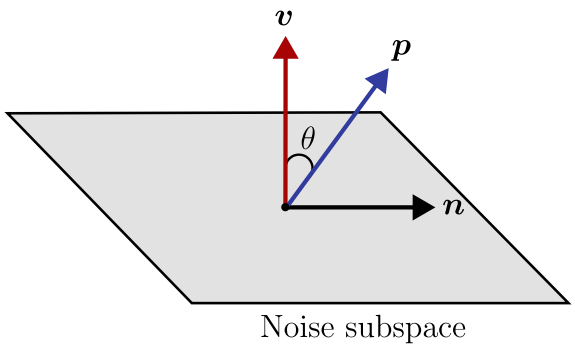


Figure 2.13: Geometrical representation of signal mismatch, indicated by the angle θ between expected steering vector $\boldsymbol{v}$ and actual steering vector $\boldsymbol{p}$. Note that this θ is different from the angle mentioned in Section 2.3.5.

This mismatch is typically quantified in terms of the squared cosine of θ [1], where

matched conditions imply $\boldsymbol{p} = \boldsymbol{v}$ and $\cos^2\theta = 1$:

$$\cos^2\theta = \frac{|\boldsymbol{p}^\dagger \boldsymbol{C}^{-1}\boldsymbol{v}|^2}{(\boldsymbol{v}^\dagger \boldsymbol{C}^{-1}\boldsymbol{v})(\boldsymbol{p}^\dagger \boldsymbol{C}^{-1}\boldsymbol{p})} \in [0,1], \tag{2.42}$$

where $\boldsymbol{C} \in \mathbb{C}^{MN\times MN}$ is the covariance matrix of the clutter and thermal noise, defined as

$$\boldsymbol{C} = \boldsymbol{R}_\mathrm{c} + \boldsymbol{R}_\mathrm{n}, \tag{2.43}$$

with $\boldsymbol{R}_\mathrm{c} \in \mathbb{C}^{MN\times MN}$ being the clutter covariance matrix, and $\boldsymbol{R}_\mathrm{n} = \sigma_\mathrm{n}^2\boldsymbol{I}$ the thermal noise covariance matrix. It is important to note here that $\boldsymbol{R}_\mathrm{c}$ is *not* a scaled identity matrix, such as $\boldsymbol{R}_\mathrm{n}$, and may contain nonzero elements outside its diagonal.

The SNR γ can also be determined by $\boldsymbol{p}$ and $\boldsymbol{C}$:

$$\gamma = |\alpha|^2 \boldsymbol{p}^\dagger \boldsymbol{C}^{-1}\boldsymbol{p} \in \mathbb{R}_+, \tag{2.44}$$

where the amplitude α is chosen deterministically for a desired SNR.

2.4 CFAR Feature Plane

The CFAR-FP, first proposed in [1], has its foundation in *invariant detection*. Invariant detection is a branch within detection theory which involves mathematical transformations on the hypothesis test in (2.28) in a manner which ensures that parameters in the statistical distribution significant for detection, such as the statistical mean of the data, remain unchanged after the transformations. Any change to the covariance is deemed irrelevant for detection performance [22].

In the case of the AMF and Kelly's CFAR detectors, both assume the covariance matrix is completely unknown and unstructured. However, in many scenarios the antenna array geometry as well as partial knowledge of the noise environment are known and therefore provide a rough structure to the covariance matrix [22]. Invariant testing utilizes this semi-structure to significantly improve the probability of detection, given a fixed false-alarm rate. As a result of these transformations, any pair of decision statistics with a 1-1 relation to t_Kelly and t_AMF may be defined in terms of a *maximal invariant* test. Given the relation to the AMF and Kelly's decision statistics, these invariant tests notably also possess the CFAR property [1].

The CFAR-FP framework provides a new method for analyzing different CFAR detectors' performance in terms of their maximal invariant statistics directly in a two-dimensional plane. Any given CUT and collection of secondary data cells are compressed using the maximal invariant statistics to a single point in the plane, defined by two statistics, β and $\tilde{t}$, which are independent under the H_0 assumption and have well-defined distributions [6, p. 151], [1]. After multiple cells have been processed, clusters will form in the plane, whose region depend on the characteristics of the received signal.

The maximal invariants in the CFAR-FP processing chain are defined by the variables s_1 and s_2. In this work we define them as in [1] to be equal to t_ED and t_AMF

respectively:

$$(s_1, s_2) = (t_{\mathrm{ED}}, t_{\mathrm{AMF}}) = \left(\boldsymbol{z}^{\dagger} \boldsymbol{S}^{-1} \boldsymbol{z}, \frac{|\boldsymbol{z}^{\dagger} \boldsymbol{S}^{-1} \boldsymbol{v}|^2}{\boldsymbol{v}^{\dagger} \boldsymbol{S}^{-1} \boldsymbol{v}} \right) \in \mathbb{R}_+, \tag{2.45}$$

where $\mathbb{R}_+ \in (0, +\infty)$. These are then decomposed in terms of the $\tilde{t}$ and β statistics:

$$(\beta, \tilde{t}) = \left(\frac{1}{1 + s_1 - s_2}, \frac{s_2}{1 + s_1 - s_2} \right) \in (0, 1) \times \mathbb{R}_+. \tag{2.46}$$

This mapping chain can be modeled as a multilayered learning machine similar to an unsupervised NN, as shown in Fig. 2.14.

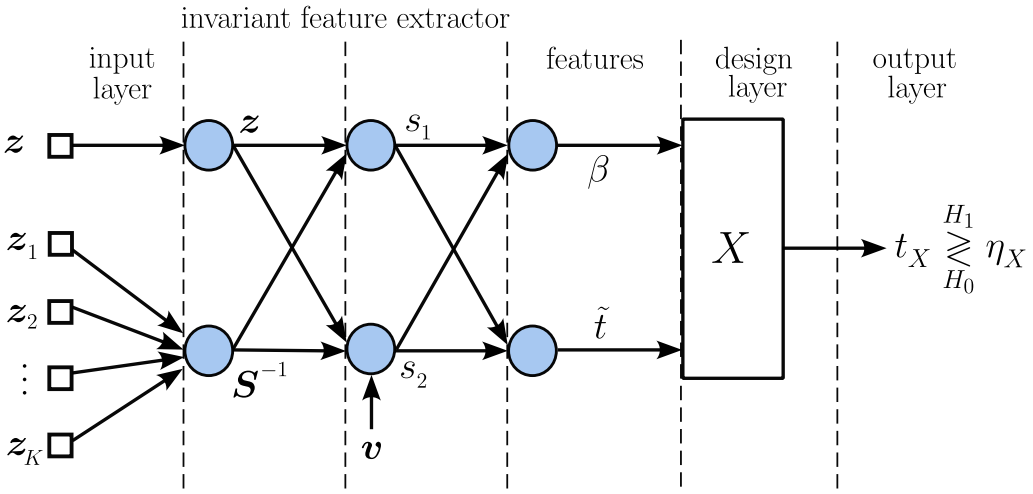


Figure 2.14: Network model of the CFAR-FP processing chain based on [1, Fig. 1] with detector X, decision statistic t_X, and threshold η_X.

In Fig. 2.14, the raw radar data $\{\boldsymbol{z}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_K\}$ are passed through the input layer, followed by two hidden layers which apply two stages of data compression, in order to extract the invariant features. The first hidden layer ensures that the secondary data is only used to construct $\boldsymbol{S}^{-1}$, while the second layer guarantees that the only permissible transformations are the ones which depend on the energy of the received signal (s_1) and its correlation with the energy-normalized steering vector (s_2) [1]. The final layer then maps (s_1, s_2) to $(\beta, \tilde{t})$ for a more convenient statistical representation of the features.

In the design layer, the block denoted by X indicates the radar detector used to determine the output decision statistic t_X, which is then compared to a threshold η_X (determined by the detector X and desired P_{FA}). Given the maximal invariant compression, any well-known CFAR detector can often have its decision statistic $t_X(\boldsymbol{z}, \boldsymbol{S})$ expressed in terms of $(\beta, \tilde{t})$ only [6, p. 154]. For instance, for the detectors (2.31) and (2.32), the corresponding reparameterization of their decision statistics are

$$t_{\mathrm{Kelly}} = \frac{\tilde{t}}{1 + \tilde{t}} \tag{2.47}$$

$$t_{\mathrm{AMF}} = \frac{\tilde{t}}{\beta}. \tag{2.48}$$

Thus, a decision boundary can then be defined for the CFAR-FP by analyzing the equation $t_X(\beta, \tilde{t}) = \eta_X$. For the detectors described in Section 2.3.4, their respective decision boundaries are expressed as the curve equations in the CFAR-FP and shown in Table 2.1 [1].

Table 2.1: Decision boundaries for different CFAR detectors in the CFAR-FP.

Name	**Curve Equation**
Kelly's detector (2.31)	$\tilde{t} = \eta_{\text{Kelly}}$
AMF (2.32)	$\tilde{t} = \eta_{\text{AMF}}\beta$
ED (2.30)	$\tilde{t} = (\eta_{\text{ED}} + 1)\beta - 1$

A multitude of different well-known CFAR detectors can be expressed as linear or non-linear classifiers in the β-$\tilde{t}$ plane, just as in Table 2.1 [1]. This enables an intuitive method for tuning detectors based on the desired selectivity or robustness. If robustness is desired, the classifier should aim to separate the H_0 cluster from the union of matched and mismatched H_1 clusters. Conversely, if selectivity is desired, the classifier should separate the union of the H_0 and mismatched H_1 clusters from the matched H_1 cluster [6, p. 151]. In the CFAR-FP, robust detectors have a positive slope, while selective detectors have a negative slope (with the exception of Kelly's detector, which has a zero slope) [1]. Fig. 2.15 shows an example illustration of what the clusters and detectors in Table 2.1 could look like in the CFAR-FP.

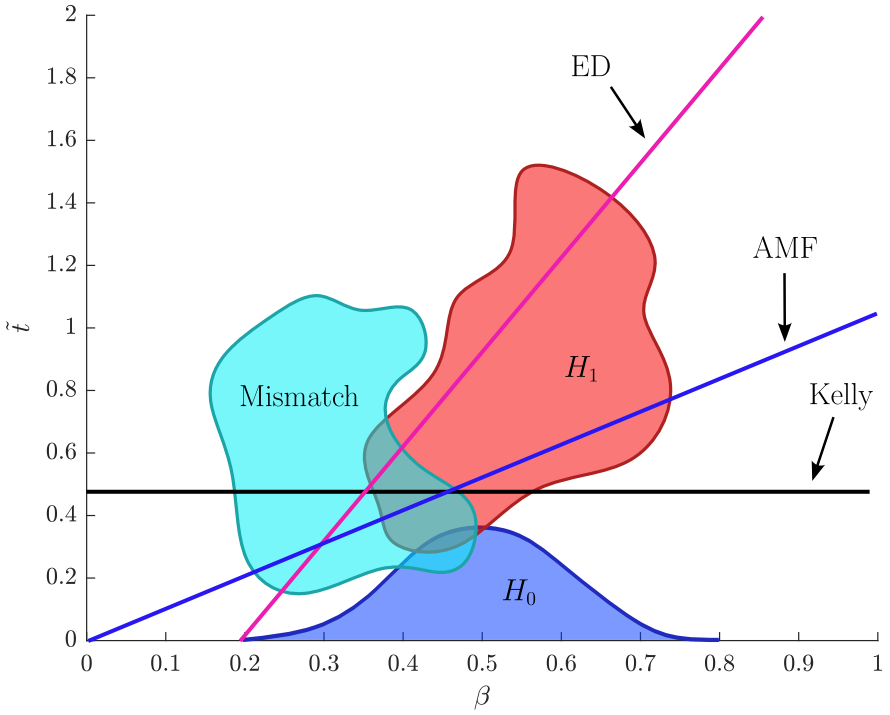


Figure 2.15: Illustration of the CFAR-FP with drawn decision boundaries of Kelly, ED, and AMF.

The CFAR-FP also provides intuitive insight to the SNR and mismatch levels since the mapping chain is based on t_{ED} and t_{AMF}. The clusters' positions change dependent on a fixed γ as a function of $\cos^2\theta$, which allows for drawing trajectories describing the location of the center of a cluster, given certain conditions [1]. Fig. 2.16 depicts a simplified illustration based on the example in [1, Fig. 8] of how the mismatched clusters migrate for a fixed SNR of 20 dB. The dashed horizontal line indicates full mismatch and the vertical line indicates perfect match. The diagonal dashed lines represent different levels of mismatch for the given SNR.

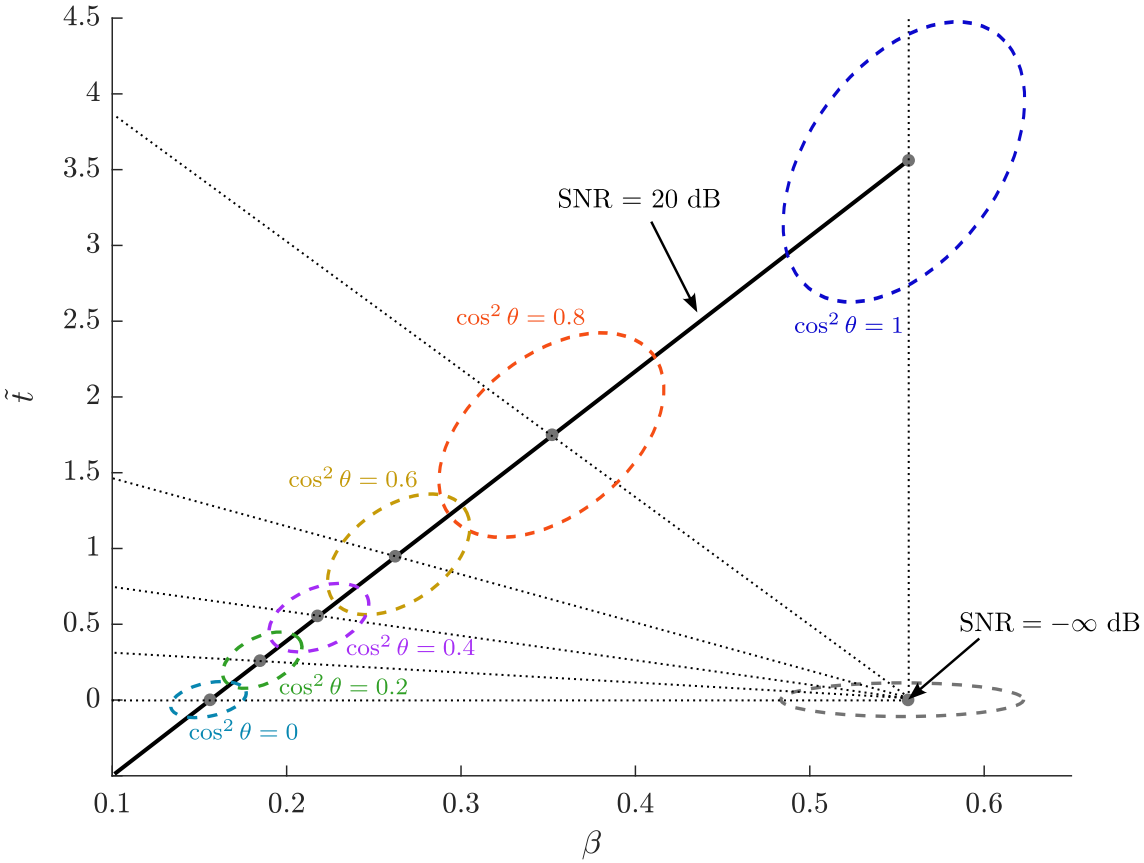


Figure 2.16: Illustration of cluster migration for fixed SNR.

Migration patterns can be drawn for any given SNR. In cases of higher SNR, the black diagonal line in Fig. 2.16 will have its end point higher on the dashed vertical line. Conversely, for lower SNR it will be closer to the $-\infty$ dB point.

2.5 Artificial Neural Networks

An ANN is an ML-model that mimics the complex functionality of the human brain in a simplified way. This allows the model to be trained to recognize patterns and make decisions based on input features provided by data sets. The model used in this thesis uses a method called supervised learning, where the input training data is labeled which allows the model to tweak its parameters based on the loss between the predicted label in the output with the actual label in the training set. In other words, the training algorithm aims to identify patterns and the relationship between input and the desired output. The goal is to train the network well enough that it can reliably predict the output from new data.

ANNs consist of interconnected layers of nodes, also called artificial neurons [23]. Each node in each layer receives input data, x_i, which is scaled with a weight, ω_i, and summed together with a bias, b, as described by

$$\sum_{i=1}^{K}(\omega_{i,j} \cdot x_i) + b_j, \tag{2.49}$$

where K is the number of inputs x to each node j [24]. The weights' function is to tune the network to ensure that the paths from input to output which lead to the highest probability of correct predictions are prioritized. The weighted sums described in (2.49) are then used in an activation function, $a_j(\bullet)$:

$$\hat{y}_j = a_j\left(\sum_{i=1}^{K}(\omega_{i,j} \cdot x_i) + b_j\right), \tag{2.50}$$

as illustrated in Fig. 2.17. The output $\hat{y}_j$ is connected to all nodes in the next layer, making the outputs of one layer the inputs to the next layer. This is called a *feed-forward network*.

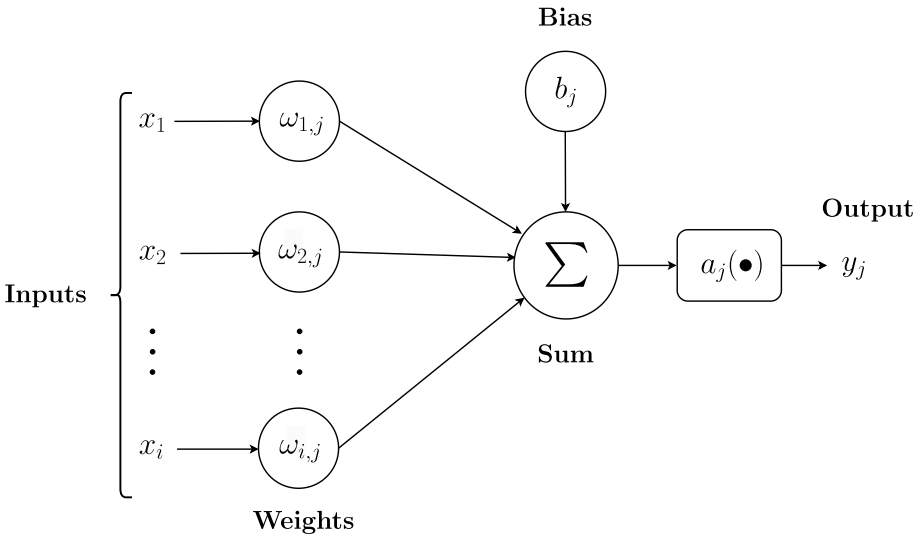


Figure 2.17: Node j of a feed-forward NN.

Another way to connect the inputs and outputs of the nodes is using a *recurrent network* that uses its own outputs as inputs [24]. This type of network is a dynamic system, where the network's current output is not only determined by the present input but also by its previous value. They are designed to process sequential data, where the order of the inputs is important.

In this thesis we focus on *feed-forward networks*. The network consists of an input layer and an output layer, in some cases there are hidden layers in between. When there are hidden layers it is called a *multilayer feed-forward neural network*, while when there is only an input and an output layer, it is called a *single-layer neural network* or a *perceptron network* [24].

Unlike the nodes in the hidden and output layers, the input layer nodes do not perform any computations. Each input feature corresponds to one node in the input layer. Their sole function is to pass raw input data directly to the next layer without applying any weights, biases, or activation functions [25], [24]. The nodes in the hidden layer process the input data by calculating weighted sums and apply an activation function, as described in (2.50). In some cases there can be multiple hidden layers in a network, this is typically called a *deep neural network* (DNN) [25]. Each hidden layer passes the results from their activation function to the next layer, as seen in Fig. 2.18.

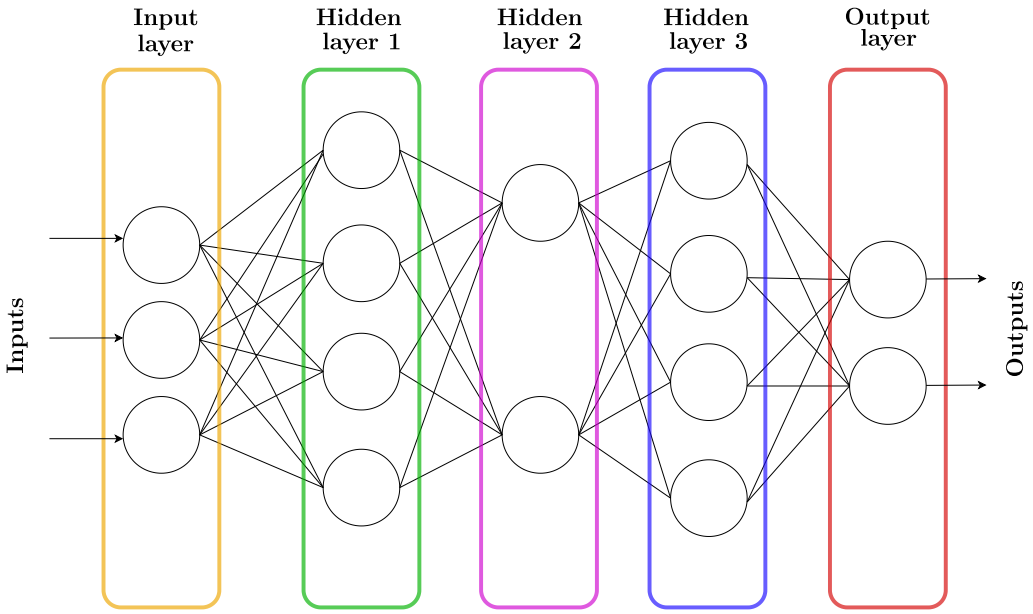


Figure 2.18: DNN with multiple hidden layers.

Selecting an appropriate activation function is essential as it significantly influences the behavior and performance of the NN. These functions determine how the output from each node is calculated and passed to the next layer. The activation function commonly used in hidden layers play a vital role in learning non-linear patterns in the data and extracts and transforms the features in a proactive way, making the prediction step in the output layer easier [24]. In other words, it serves as a filter that reshapes and enhances the input data. This refinement helps the next

layer process the data more efficiently, leading to better decision-making. Among the most commonly used activation functions for hidden layers is the *rectified linear unit* (ReLU) [26]. The ReLU outputs a 0 for all negative input values and propagates the input if it is positive. Thus, the ReLu activation function can be described by

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}. \tag{2.51}$$

The simplicity of the ReLu function makes it efficient to use for training an NN [24]. Even though it seems to be a bit-wise linear function, the ReLU function is actually a non-linear function. This enables the network to learn more complex data patterns [25]. As we can see in (2.51), all zero input values are output as negative values. This introduces *sparsity* in the network, i.e., only a small number of nodes are activated at each time instance, leading to more efficient and faster calculations.

Another widely used activation function is called the *Leaky ReLU*, which is similar to the ReLU function, but introduces a small slope for negative input values rather than outputting zero. This slope is scaled by a constant α, as described by

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}, \tag{2.52}$$

where α is a small positive constant, typically a small value like 0.01 [25].

The output layer uses the same data flow as the hidden layer, calculating a weighted sum of the input values and applying an activation function. The purpose of the activation function in the output layer is to produce a final prediction in the desired format. It usually consists of one node or as many nodes as there are classes in a classification problem [25], [24]. Classification problems involve assigning inputs to specific categories. For instance, in binary classification problems, there are two discrete categories to which an input can be assigned. In such cases, only one node is used in the output layer, resulting in one probability value, $\hat{y}$. The predicted probability that the input belongs to Class 1 of the two different possible classes [24]. In the binary classification problem the output node typically uses a *sigmoid* activation function, described by

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.53}$$

The probability, $\hat{y}$, produced by the output layer is then compared to the true label, y, of the input. Typically, the model outputs a probability value (after applying the sigmoid function) representing the likelihood that a sample belongs to class 1 [25]. For example, if the output is 0.9, it indicates the model is 90 % confident that the sample is in Class 1. A loss function calculates the difference between the predicted probabilities and the actual label, providing a numerical representation of the model's error to guide the learning process. The loss function helps the model generate probabilities that align closely with the true labels. When the model is highly confident but wrong, the loss becomes large, sending a strong signal to update the model's weights [23]. The loss can generally be described as:

$$L = \frac{1}{N} \sum_{i=1}^{N} \ell(y_i, \hat{y}_i), \tag{2.54}$$

where $\hat{y}_i$ and y_i are the predicted and true labels of sample i respectively, and N is the total number of samples.

The choice of loss function in a classification problem depends on several factors, including how the model is built, the number of classes, how the output layer is structured, and the activation function used in the output layer. The loss function must be consistent with both the activation function and the intended probabilistic interpretation of the model's output [25].

In binary classification, the model usually uses a sigmoid-activated output node to produce probabilities, with binary cross-entropy (log loss) measuring how well these align with the true labels. The *binary cross-entropy* loss (BCE) is derived from the MLE principle applied to a Bernoulli distribution, where the true labels y_i are binary (0 or 1), and the predicted values $\hat{y}_i$ are the probabilities that the label is 1 [25]. The BCE loss can be expressed as

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right]. \tag{2.55}$$

2.5.1 Back-propagation

The loss is then used to compute gradients during the back-propagation phase of training. Back-propagation applies the chain rule to efficiently compute the partial derivatives of the loss with respect to each parameter (weights and biases) in the network. These gradients indicate how much each parameter contributes to the error [25]. This gradient vector points in the direction of the greatest rate of increase of the function. Each component indicates how sensitively the function responds to changes in the corresponding input variable [24].

The gradient is used in a method called *gradient descent*, which minimizes the loss function by adjusting the model's parameters in the direction that reduces the loss most rapidly and optimizes the model, using only the first derivative (gradient) of the function [25]. After each gradient-based update, the network's parameters are slightly adjusted to more effectively minimize the loss function. The training then returns to the forward propagation step, now utilizing the updated parameters. This process repeats over multiple training iterations, more commonly known as *epochs*, gradually decreasing the loss and enhancing the model's performance [24].

2.5.2 Evaluation of model

After the training process is finished, the next step is to assess the model's performance on unseen data. This involves using the trained model to make predictions without updating its parameters, and is commonly referred to as *inference*. Depending on the specific task, appropriate evaluation metrics such as accuracy, precision, recall, and F1-score are calculated.

Accuracy is the ratio of correct classifications (positive and negative) and all predictions the model makes. Precision, also known as the positive predictive value (PPV), represents the proportion of actual positives among all examples the model classified as positive. Recall is the rate of positive classifications made by the model, out of all actual instances of the positive class [27].

In classification tasks, a confusion matrix is used to provide a detailed overview of the model's predictions compared to the true labels. It results in four fundamental outcomes: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), as seen in Fig. 2.19.

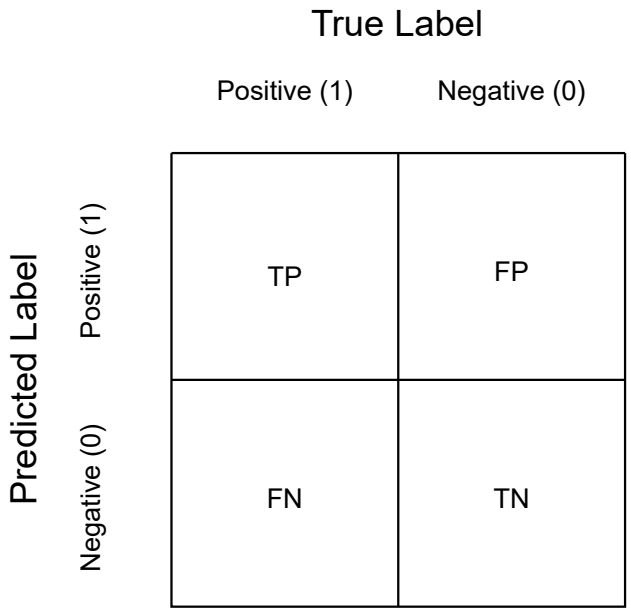


Figure 2.19: Confusion matrix.

TP occurs when the model correctly predicts the positive class, while TN occurs when the model correctly predicts the negative class. FP occur when the model incorrectly predicts the positive class when it actually was negative and FN is when the model predicts negative, but it was actually positive [27]. The accuracy, precision,

and recall of the model can then be calculated as

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.56}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.57}$$

$$\text{Recall (Sensitivity)} = \frac{TP}{TP + FN}. \tag{2.58}$$

To analyze overall performance of the model, the precision and recall parameters are often combined to form the F1-score. The F1-score is the harmonic mean of precision and recall, and provides a single metric that indicates how good the balance between precision and recall is [27]. The F1-score is computed as

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.59}$$

Furthermore, when used in the realm of radar detection, P_{D} and P_{FA} may also be expressed in terms of TP, FP, TN, and TP as

$$P_{\text{D}} = \frac{TP}{TP + FN} = \text{Recall} \tag{2.60}$$

$$P_{\text{FA}} = \frac{FP}{FP + TN}. \tag{2.61}$$

3

Methods

The approach taken for this project will consist of three main steps. Firstly, a system model of the CFAR-FP mapping chain shall be created in MATLAB. In order to create the classifier model, suitable datasets need to be created both for training and validation, with accurate labels for each observed CUT. Since the provided experimental radar data contains primarily noise and clutter, with no certain target locations known beforehand, accurate signal models of targets – using the defined steering vector $\boldsymbol{v}$ in Section 2.3.5 – will be used to synthetically inject a target $\boldsymbol{p}$ at any given range bin or CPI index in the radar data matrix. This will allow for full control of targets' location, gain, and signal mismatches. This will enable us to sweep over the radar data matrix and perform the CFAR-FP mapping chain – explained in Section 2.4 – on each cell to generate desired clusters. Additionally, we will only use data from one already beamformed channel, which means all matrices and vectors used in the CFAR-FP mapping chain will have their dimensions defined only by the M pulses within a CPI, since $N = 1$.

Secondly, a binary classifier NN will be built and trained using the `PyTorch` library in Python. The classifier should then effectively realize detector X of the design layer in Fig. 2.14, and outputs a design statistic $t_{\mathrm{NN}} \in [0, 1]$ that is compared with a threshold of 0.5. Once trained, the NN ultimately creates a decision region which aims to distinguish the H_0 from the matched and mismatched H_1 clusters, realizing a robust detector. This decision region may be visualized by passing a grid of coordinates in the β-$\tilde{t}$ plane through the network and color coding the regions which get classified as H_0 and H_1 respectively. Using the designed MATLAB model, different datasets will be generated with varying mismatch and SNR levels to evaluate the performance of the NN model. The metrics evaluated shall include P_{D}, P_{FA}, accuracy, and F1-score. The performance of the NN detector will then be compared with Kelly's detector, optimized for minimum P_{FA}. The trained NN model from Python shall then be compiled and deployed to the VCK190 using Vitis-AI.

Finally, we will aim to implement the CFAR-FP mapping chain in hardware using a hierarchical approach in VHDL, with multiple modules performing various complex linear algebraic operations on a given CUT and secondary data stored in block RAMs (BRAMs). These modules will primarily be implemented and verified using the rather simple Nexys A7-100T [28] board as a target device, as it simplifies synthesis and implementation steps. The available resources for the Nexys board are summarized in Table 3.1.

Table 3.1: Available resources for the Nexys A7-100T.

Logic Slices	LUT Slices	Slice Registers	BRAM	DSP Slices
15,850	63,400	126,800	4,860 kb	240

As mentioned, the target device for deploying the proposed NN design is Xilinx Versal AI Core Series VCK190 Evaluation Kit [11], which is a powerful development platform built to accelerate AI inference and signal processing tasks in edge and embedded environments. At its core is the Versal AI Core XCVC1902 Adaptive System-on-Chip (SoC), which combines AIEs, DSP engines, and programmable logic to provide high computational performance [29], [30]. The Versal adaptive SoC includes the Arm Cortex-A72 64-bit dual-core processor and the Cortex-R5F dual-core real-time processor [31].

The Versal adaptive SoC is built to handle demanding NN workloads with high efficiency. Its 400 AIEs are designed for maximum throughput while conserving power, making them suitable for tasks like AI inference and complex signal processing. These engines utilize arrays of very long instruction word (VLIW) processors with single instruction, multiple data (SIMD) vector units, allowing them to execute multiple operations in parallel across both instructions and data. This is particularly useful for achieving high performance in compute-heavy applications [32]. The AIEs can run up to 1.3 GHz and have 32 kB local memory and faster interconnects using the advanced extensible interface (AXI) protocol.

The AMD Xilinx Versal Deep Learning Processing Unit (DPU) is a flexible computation engine tailored for AI tasks on Versal-based platforms like the VCK190 evaluation board. It allows designers to configure the level of parallelism to suit the specific performance and resource constraints of their target device and application, making it well-suited for embedded deep learning solutions [33].

Xilinx provides two different ways for programming the AIEs: C/C++ code, compiling it, and creating AIE kernels using the `aiecompiler`; or using Vitis AI to deploy and run pre-trained NNs directly on the board using DPU blocks, with either C++ application programming interfaces (APIs) or Python APIs.

4

Design and Implementation

The design and implementation process was divided into two main parts: system development and hardware implementation. The former involved generating comprehensive datasets from the experimental radar data, which included synthetically injecting targets in the raw radar data and then creating clusters using the CFAR-FP mapping chain. This was all implemented in MATLAB and is covered in Sections 4.1–4.2 of this chapter. Additionally, Section 4.3 describes the proposed NN model implemented and trained using Python, which was then used as a target classifier. The latter included the register-transfer level (RTL) design for the CFAR-FP mapping chain and the deployment of the pre-trained NN model to the hardware, which are both covered in Sections 4.4 and 4.5 respectively.

4.1 Target Injection

The data used for the CFAR-FP mapping chain was gathered from a ground-based rotating experiment radar. The radar antenna consists of multiple subapertures and rotates with a certain angular velocity ω_{R}[1]. Thousands of pulses are transmitted each rotation and each received pulse can be used to infer the rotated angle of the radar in relation to true north.

The data had been stored in a video file together with the parameters used during sampling. From the video file, a 3-D matrix could be accessed in MATLAB containing the range bins, pulses, and beamformed antenna elements. This matrix has similar format to that of Fig. 2.4, although it contains all received pulses during a rotation instead of just one CPI of M pulses and instead of antenna elements it contains already beamformed channels. The beamformed channel selected concentrated received echoes originating from areas high above the ground, in order to mitigate the effects of ground-based clutter. An image plot of this matrix from one rotation of the radar is shown in Fig. 4.1; the received power is color coded from blue to yellow, blue indicating low power (thermal noise) and yellow higher power (clutter).

[1]Exact value cannot be disclosed.

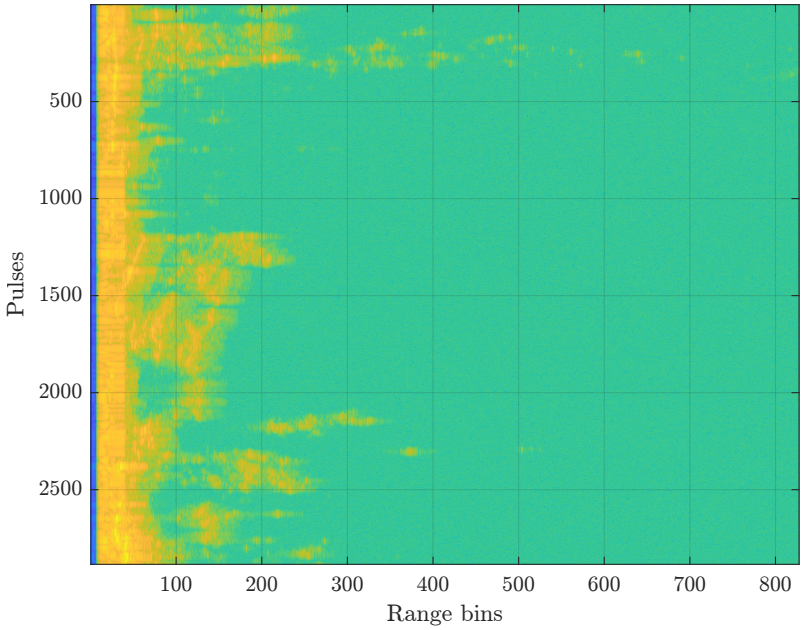


Figure 4.1: Pulse compressed raw radar data matrix from one antenna rotation.

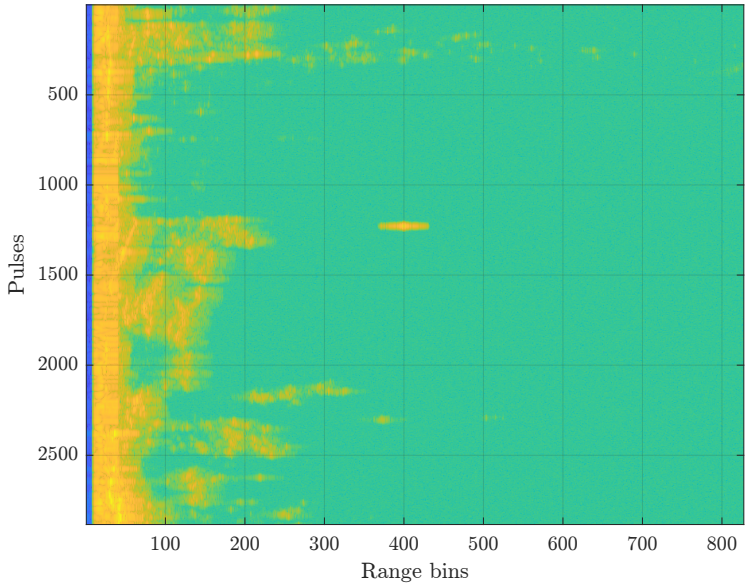
In order to train the proposed NN it is necessary to know for certain which radar cells contain targets using prior knowledge in order to accurately label the data. Additionally, training a high-performing network requires huge datasets which would entail generating large H_0, H_1, and mismatch clusters. Since knowledge of the surrounding environment was limited, and from the video data it was observed that the data mainly contained clutter and noise, it was decided that the optimal approach for benchmarking would be to synthetically inject targets into the dataset.

A synthetic target was defined as the steering vector described by (2.40). Note that this vector is a temporal steering vector, and therefore is missing any spatial dependencies. The motivation for only using a temporal steering vector was to simplify the processing chain as much as possible, as the main focus of this project is not space-time adaptive processing but rather target detection using an NN on the CFAR-FP. As a result of this lack of spatial steering vector, targets hidden in clutter will be difficult to detect. How this was handled is covered in Section 4.2.

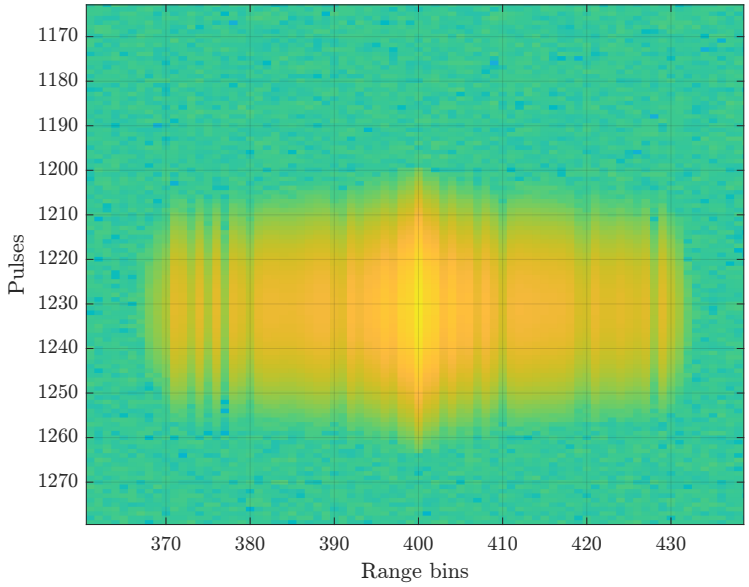
The steering vector could be configured to generate different targets simply by changing the maximum gain G_0 and the radial velocity v (i.e. Doppler frequency). Given the angular velocity ω_{R} and the beam width θ_{B} from the video data, the minimum required number of pulses for one CPI was calculated from (2.41). The closest rounded up value of base two in our case is 32, which was used as the CPI.

A typical pulse response at a range bin r_k will appear in the image plot as stretched over a few bins surrounding r_k, with its power maxima located at bin r_k. This is a result from the beamforming where each subaperture in the antenna receives a CPI of pulses with slightly different time delay which will be interpreted as small variations in range.

Such a pulse response with normalized gain was provided by Saab as a `mat`-file in MATLAB. The pulse response could then be modulated using (2.40) along the pulse dimension in Fig. 4.1 for one CPI. Fig. 4.2a shows the same data as in Fig. 4.1, but with a synthetic target injected at range bin 400 and starting pulse index 1200. A CPI of 64 pulses was used to clearly observe the Gaussian envelope. In Fig. 4.2b, a zoomed-in version of the same plot is shown with the synthetic target clearly visible.



(a) Plot of the pulse compressed raw radar data matrix from one antenna rotation with injected target at range bin 400.



(b) Zoomed in plot of the pulse compressed raw radar data matrix from one antenna rotation with injected target at range bin 400.

Figure 4.2: Pulse compressed raw radar data matrix from one antenna rotation with injected target at range bin 400 and starting pulse index 1200.

4.2 CFAR-FP Mapping

The CFAR-FP mapping chain depicted in Fig. 2.14 was developed and tested in MATLAB in order to generate datasets for the NN. Since MATLAB is structured around matrix computations, with many linear algebra functions available as well as a capability to handle complex numbers, it made for a suitable choice given the nature of the CFAR-FP processing chain.

Implementing the necessary equations for the processing chain, expressed in (2.45) and (2.46) is straight-forward in MATLAB. The main design considerations were rather to ensure customizable parameters for the generated datasets such as sweep range, number of secondary data and guard cells, steering vector parameters, pulses per CPI, and SNR and mismatch levels.

All three clusters were generated by sweeping over the range bins in the radar data and performing the CFAR-FP algorithm on each CUT from a specified starting range bin up to the last cell which still fits half the secondary data and guard cells above it[2]. The number of guard cells was set to a value which ensured that the entire pulse response of the synthetic target (see Fig. 4.2b) was excluded from the secondary data. As mentioned in the previous section, given the high levels of clutter in provided radar data, and the fact that our steering vector has no spatial dependencies, the starting range bin was set to 500 since from that point the data contains very low levels of clutter (see Fig. 4.1).

To generate H_0, no targets were injected in the data so that the steering vector $\boldsymbol{v}$ is correlated with only the noise measurements. For H_1, the steering vector $\boldsymbol{p} = \boldsymbol{v}$ was injected on all selected range bins. Finally, to generate the mismatched H_1 cluster, the injected target $\boldsymbol{p}$ was equal to $\boldsymbol{v}$, but with a small Doppler frequency error to generate desired mismatch.

SNR and mismatch levels were set deterministically by using (2.44) and (2.42) to compute their respective values for a given steering vector and radar dataset. For the mismatch we assumed Gaussian noise since the range bins containing clutter are mostly omitted for the sweep range. As a result, the interference covariance matrix $\boldsymbol{C}$ is replaced by $\boldsymbol{R}_{\text{n}}$, resulting in it being canceled out in (2.42). Thus, the expression simplifies to

$$\cos^2\theta \approx \frac{|\boldsymbol{p}^\dagger \boldsymbol{v}|^2}{(\boldsymbol{v}^\dagger \boldsymbol{v})(\boldsymbol{p}^\dagger \boldsymbol{p})}. \tag{4.1}$$

Using this approximation to compute the mismatch, we could derive a list of Doppler frequency offsets for $\boldsymbol{p}$ which resulted in certain mismatch levels.

For the SNR calculation the Gaussian noise approximation was also applied. However, unlike in the mismatch case the covariance matrix does not cancel out in (2.44). Instead $\boldsymbol{R}_{\text{n}}$ was approximated by measuring the mean value of the noise power in all cells in the radar data starting from range bin 600 and multiplying it with an $M \times M$ identity matrix. SNR was computed for the matched case of ideal steering vector $\boldsymbol{p} = \boldsymbol{v}$, with the complex scalar α was set to unity gain. Then, similarly to

[2]The exact range in kilometers that these range bins correspond to cannot be disclosed.

our method for deriving mismatch levels, we could define a list of different values for G_0 which resulted in a certain SNR given a specific $\boldsymbol{p}$ and $\boldsymbol{R}_{\text{n}}$.

It should be noted that this is quite a coarse approximation of the SNR and does not include other quantities typically included in radar signal processing, such as signal-to-interference-plus-noise ratio (SINR) which also includes clutter and jamming signals. However, since we do not sweep over the range bins with high clutter levels we deemed this SNR approximation as sufficient.

Finally, the MATLAB implementation generates scatter plots of β-$\tilde{t}$ after the mapping process has completed, offering insight into the geometric distribution of clusters under different hypotheses. These visualizations helped assess the class-separating capability of the CFAR-FP mapping under varying signal conditions, and ensured that the plots were consistent with what was expected from the theory covered in Section 2.4.

Fig. 4.3 shows a CFAR-FP generated in MATLAB from three rotations worth of gathered pulse compressed radar data. The gain and frequency error of the steering vector were set to achieve an SNR of 20 dB and average mismatch $\cos^2\theta \approx 0.75$. In Fig. 4.4, we have instead swept across all range bins in order to include the ground-based clutter close to the radar shows the affects of clutter on the CFAR-FP. As can be seen, the clutter has heavily affected the clusters and has caused both the mismatched and matched H_1 clusters to spread down to the H_0 region, making classification difficult as not clear decision boundary can be seen. Since our work does not include a method for filtering clutter, all datasets were generated by sweeping from starting range bin 500 as in Fig. 4.3 in order to mitigate the effects of clutter.

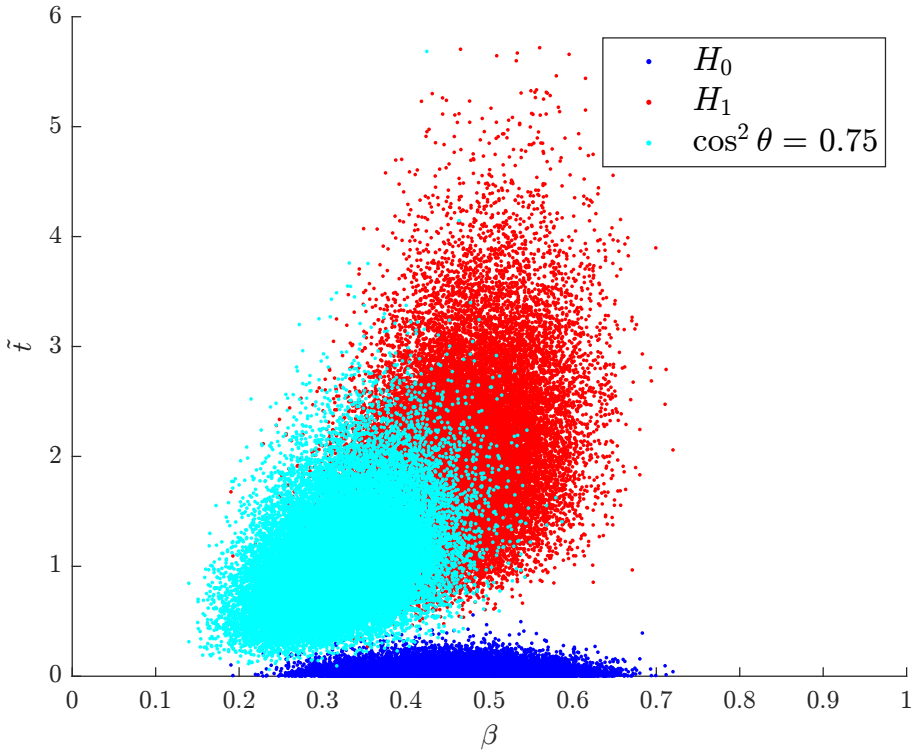


Figure 4.3: CFAR-FP generated by sweeping from range bin 500 to end using three rotations of radar data with 20 dB SNR and $\cos^2\theta \approx 0.75$.

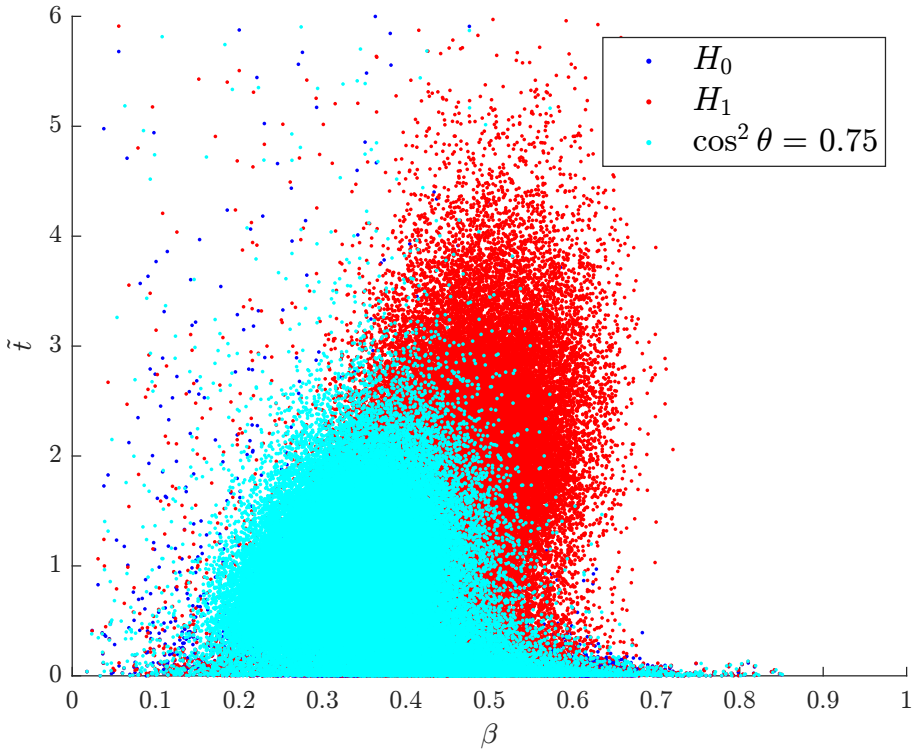


Figure 4.4: CFAR-FP generated by sweeping across all range bins from three rotations of radar data with 20 dB SNR and $\cos^2\theta \approx 0.75$.

4.3 NN Model Development

The CFAR-FP data was stored as a CSV file to make it easier to handle in Python. The CSV file consisted of three columns: β, $\tilde{t}$ and labels (H_0, H_1, or mismatch). To enable efficient model training, the dataset was randomly split into 80 % training and 20 % test sets using the `train_test_split` function provided by the `scikit-learn` library. The features were normalized using `StandardScaler` to scale them to have zero mean and unit variance. For the NN model, an input layer was designed with two nodes: one for β values and another one for $\tilde{t}$ values.

Various configurations of hidden layers and nodes were tested to determine the optimal model, as depicted in Fig. 4.5. The goal was to strike a balance between model complexity and performance, aiming for the smallest possible model that could still achieve a low P_{FA}. This optimization is critical for implementing the model efficiently on hardware while maintaining the best possible performance. The model's hidden layers utilize the ReLu activation function. This function introduces non-linearity into the network, allowing it to capture complex patterns and relationships within the data. It is also computationally less expensive than other non-linear activation functions, such as the sigmoid or hyperbolic tangent (tanh), making it suitable for hardware applications.

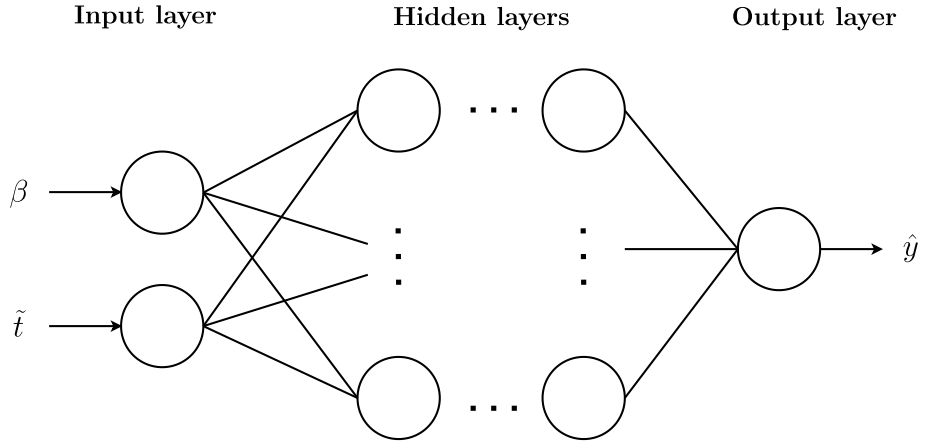


Figure 4.5: NN design with various amount of nodes and hidden layers

To train the model with the dataset, data loading and batching were handled using `PyTorch`'s `DataLoader` class. Instances of this class, representing the training and validation datasets, were then created. These objects, referred to as the training loader and validation loader, efficiently iterate through the respective datasets, providing batches of data for each training epoch to reduce the training time. The input features are passed through the model producing output values. These values are then compared to the true labels, obtained from the CSV file, to calculate the loss. Batch normalization was also applied to the inputs of each layer to enable faster and more stable training.

The loss is calculated using `BCEWithLogitsLoss` which is a binary cross-entropy (BCE) loss function with logits that also applies the sigmoid activation function

to the output nodes. The sigmoid function transforms the network's output into probabilities between 0 and 1, indicating the likelihood that the input belongs to the positive class (H_1 or mismatch in this context); making it an ideal activation function for a binary classifier.

The loss function is then used to enable backpropagation, as described in Section 2.5.1. `PyTorch` automatically computes the gradients of the loss function with respect to the model's weights. These gradients indicate the direction and magnitude of change needed in the weights to reduce the loss. Basic gradient descent uses the same learning rate for all parameters, which can slow down training. More advanced optimizers, such as *Adam* which was used in this project, improve this by adjusting the learning rate for each parameter separately. The Adam optimizer keeps track of the average of past gradients and their squares in order to make these adjustments. It has the benefit of being fast, uses little memory, and works especially well for large models or datasets [25]. Adam utilizes the gradients to adjust the model's weights iteratively, with the goal of minimizing the loss. This iterative process of gradient computation and weight adjustment continues throughout the training process, gradually improving the model's ability to make accurate predictions. Since the matched and mismatched H_1 clusters account for twice as many samples as the H_0 cluster, the positive weights in the loss function were initialized to 0.5 so that the model does not become too biased towards the positive class.

During training, the model's performance was also evaluated on a separate set of validation data. The validation process involves propagating each batch of validation data through the trained network. For each batch, predictions are generated by performing a forward pass through the model. The loss is then calculated by comparing these predictions to the true labels of the validation data. This process is repeated for all batches in the validation set, providing an overall measurement of the model's performance on unseen data over each epoch. Throughout the training, the validation loss is closely monitored. If the validation loss fails to improve after a predetermined number of epochs, training is halted prematurely. This early stopping mechanism helps prevent overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data.

The model's prediction accuracy was assessed by assigning the output of the model, $\hat{y}$, to class H_1 if $\hat{y} \geq 0.5$, and class H_0 if $\hat{y} < 0.5$ and calculating the percentage of correct predictions compared to the true labels. Both training and validation losses and accuracies are tracked during each epoch of the training process. This data was then used to create graphs that show how well the model learned over time.

Once trained, the decision boundary could be drawn by passing coordinates of a grid through the network; coloring the regions classified as H_1 in red, and the regions classified as H_0 in blue. Finally, the model is evaluated on a separate CFAR-FP test dataset. The values for TP, TN, FP, and FN are calculated using the `confusion_matrix` function from Python's `sklearn` module. These values could then be used to determine P_{D}, P_{FA}, accuracy, and the F1-score for the model's predictions on the test dataset using (2.56)–(2.61). The new CFAR-FP dataset was then plotted together with the decision boundary to provide a visual result that clearly illustrates how the clusters are separated.

The results of the NN detector was compared with the traditional Kelly's detector for performance benchmarking. Its threshold was determined by looping through each value of $\tilde{t}$ and calculating the P_{FA} and P_{D}. The best P_{FA}, i.e. its lowest value, was saved and the corresponding $\tilde{t}$ value determines the optimum threshold.

4.4 RTL Design

In this section we describe the proposed RTL design for implementing the CFAR-FP mapping chain described in Section 2.4 using VHDL. The mapping-chain involves linear algebraic operations on complex-valued vectors and matrices with dimensions defined by the M pulses constituting a CPI. In order to calculate s_1 and s_2 in (2.45), several operations are needed.

The scatter matrix $\boldsymbol{S}$ has to first be computed by accumulating K complex column-row vector multiplications from the secondary data cells, followed by a complex matrix inversion to obtain $\boldsymbol{S}^{-1}$. Next, we note that the expressions $\boldsymbol{z}^\dagger\boldsymbol{S}^{-1}\boldsymbol{z}$, $\boldsymbol{z}^\dagger\boldsymbol{S}^{-1}\boldsymbol{v}$, and $\boldsymbol{v}^\dagger\boldsymbol{S}^{-1}\boldsymbol{v}$ in (2.45) require first a matrix-vector multiplication to obtain a new column vector, followed by a dot product. Finally, a square-root operation is needed to compute the absolute value of the complex scalar $\boldsymbol{z}^\dagger\boldsymbol{S}^{-1}\boldsymbol{v}$ in the numerator of s_2. To realize these linear algebraic operations we additionally require a complex multiplier and complex divider.

Realizing this in gate-level logic requires careful design considerations. Recall that a complex number is expressed as $z = a + jb$, where $a, b \in \mathbb{R}$ and $j = \sqrt{-1}$. Of course, there is no way of implementing j directly in hardware. Instead, we utilize the fact that a complex number always is represented by one real and one imaginary part and define all modules in the RTL design to always have one real and one imaginary channel for each data input and output.

Furthermore, the MATLAB implementation uses double precision floating point representation, which should if possible be avoided in hardware as it complicates mathematical operations and rounding since the exponent and mantissa must be handled separately [34]. Instead signed 2's complement fixed-point representation was used for the hardware design. The required number of integer and fractional bits was determined by looking at the results from computations in the CFAR-FP mapping chain in MATLAB and finding the minimum and maximum values represented. It was decided that a 32-bit data path would be sufficient for the proof-of-concept design, with 18 fractional bits and 14 integer bits (including sign bit). This is represented by the notation s13.18, where s denotes the sign bit.

The value, v, of a given 2s complement signed fixed-point word is calculated as

$$v = r\left(-b_{N-1}\cdot 2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i\right), \tag{4.2}$$

where N is total number of bits, r is the resolution, and b_i the bit at position i [34]. The resolution is the smallest representable value and is determined by the number of fractional bits. Given our s13.18 format the resolution is

$$r = 2^{-18} \approx 3.815 \cdot 10^{-6}, \tag{4.3}$$

and the total range of representable numbers are $[-2^{13}, \quad (2^{13} - 2^{-18})]$.

4.4.1 Complex multiplier

Multiplying two complex numbers $z_1 = a_1 + jb_1$ and $z_2 = a_2 + jb_2$ results in a product, p:

$$p = z_1 \cdot z_2 = (a_1 + jb_1)(a_2 + jb_2) = (a_1a_2 - b_1b_2) + j(a_1b_2 + a_2b_1). \tag{4.4}$$

Note that $\mathrm{Re}(p) = (a_1a_2 - b_1b_2)$ and $\mathrm{Im}(p) = (a_1b_2 + a_2b_1)$ require two multipliers and one adder circuit (either for addition or subtraction) each. Thus, we can realize the complex multiplication by performing four multiplications in parallel followed by one subtraction and one addition in parallel [7], [35], as shown in Fig. 4.6.

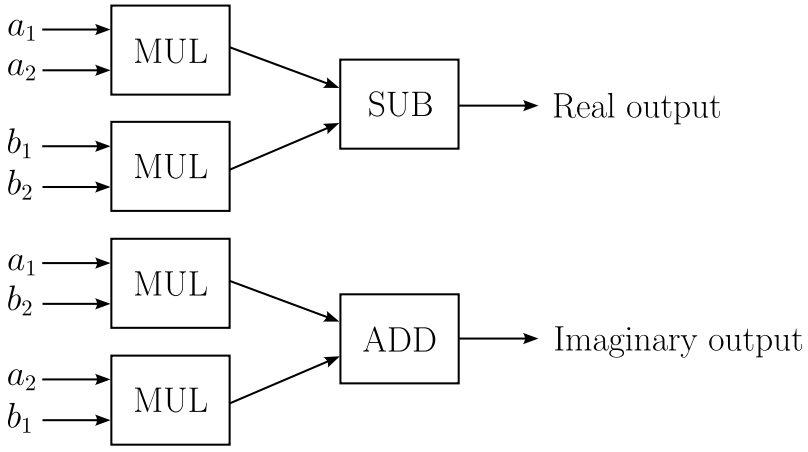


Figure 4.6: Complex multiplier (CMUL) block diagram.

The multiplier blocks were realized using Vivado's multiplier IP core [36] to enable efficient configuration of area or speed optimization as well as pipelining registers. These blocks were optimized for area with seven pipelined registers; the recommended amount from the IP documentation. Input data is stored in initial registers before the multipliers to enable full control of the data path. This proposed design has a throughput of 1/clock cycle after the initial propagation delay of ten clock cycles.

The product of the fixed-point multiplication between two factors of the format s13.18 results in a 64-bit output of the format s27.36. This is a known consequence of fixed-point multiplication that the format of the product has double the number of integer and fractional bits compared to two factors of equal format [34]. In order to read the correct results, the products are left-shifted by the number of fractional bits. In our case, this results in the format s13.50. The result is then truncated to keep the 32 most significant bits (MSBs) and discarding the extra 32 fractional bits in order to maintain a 32-bit data size. Finally, the result is rounded by using the common approach of rounding up if the MSB of the discarded bits is a 1, and down if it is a 0. Rounding down is easily achieved by simply writing a 0 to the least significant bit (LSB) of the saved bits, while rounding up requires an addition operation to increment the result by 1 which requires an additional clock cycle [34].

4.4.2 Complex divider

The complex division (CDIV) of z_1 and z_2 in polar form is calculated by multiplying the fraction with the complex conjugate of the divisor. This results in a real-valued divisor which is used to divide each real and imaginary term independently:

$$\frac{z_1}{z_2} = \frac{a_1 + jb_1}{a_2 + jb_2} = \frac{(a_1 + jb_1)(\overline{a_2 + jb_2})}{a_2^2 + b_2^2} = \frac{a_1a_2 + b_1b_2}{a_2^2 + b_2^2} + j\left(\frac{a_2b_1 - a_1b_2}{a_2^2 + b_2^2}\right). \tag{4.5}$$

Again using a parallel structure we can perform this complex division in hardware using six multipliers, three adders, and two dividers [7]. A block diagram of the complex divider is shown in Fig. 4.7.

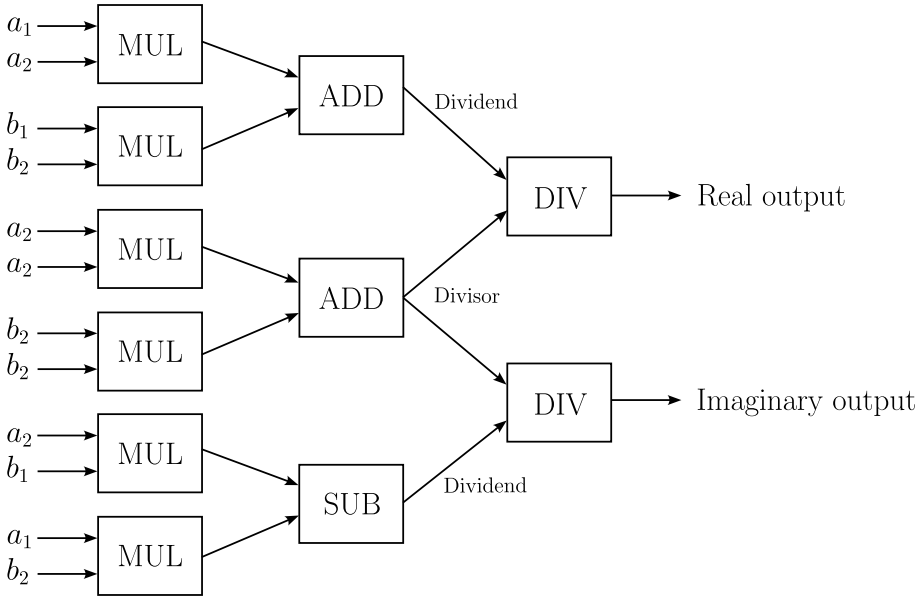


Figure 4.7: Complex divider block diagram.

The multipliers were again configured using Vivado's multiplier IP core, and the dividers were configured using the divider IP core [37]. This divider IP has the benefit of being able to be set up to generate the result in fractional format, which retained our fixed-point format. The output of the block consists of 56 bits, so we truncate it to only include the 32 LSBs. The throughput of the divider is 1-in-4, meaning we get one new computed result every four clock cycles. Therefore, in order to keep one clock cycle throughput for mathematical operations we use Vivado's `Clocking Wizard` IP [38] to generate two phase-locked-loop (PLL) clocks from our 100 MHz system clock with frequencies 100 MHz and 400 MHz respectively to enable the complex divider to propagate data four times faster than the other blocks in the system, which achieves the same overall throughput. The 100 MHz PLL clock is used as the main clock in other parts of the entire system to ensure synchronicity between these two clocks.

4.4.3 Dot product

As is known from basic linear algebra, the dot product of two vectors is a scalar value. Given two arbitrary vectors, $\boldsymbol{x}$ and $\boldsymbol{y}$, with N elements defined as

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix},$$

the resulting dot product of these two vectors is calculated as

$$\boldsymbol{x}^{\mathrm{T}} \cdot \boldsymbol{y} = \sum_{i=1}^{N} x_i y_i. \tag{4.6}$$

For our proposed design we use a CPI of 32 pulses, which means the vectors to be computed contain 32 elements. From (4.6) we see that we require 32 consecutive multiply-accumulate (MAC) operations to obtain the dot product of two vectors.

The dot product operation was implemented in VHDL using the created complex multiplier module (CMUL) previously described in Section 4.4.1 together with two adders for the real and imaginary channels respectively. A Mealy-type finite-state machine (FSM) was used in conjunction with a synchronous counter to control the program flow. A simplified block diagram of the dot product module is shown in Fig. 4.8. Note that the clock signal and intermediate registers have been omitted for simplicity, however, the module is pipelined and controlled synchronously by the clock.

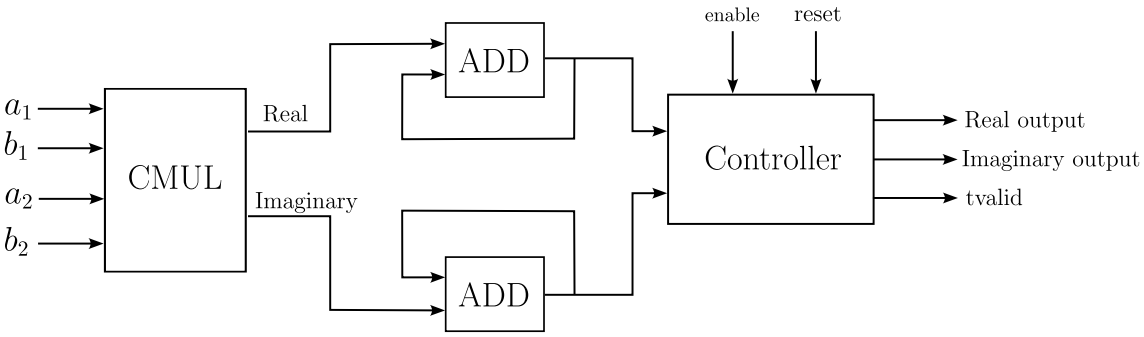


Figure 4.8: Simplified dot product module block diagram.

While in its idle state, the FSM waits for the enable signal to be asserted, after which one 32-bit complex element from each of the two vectors is stored in input registers each clock cycle. Once 32 MAC operations have been performed (determined by the counter), the `tvalid` control flag is asserted to indicate that the computed dot product is available at the output and the accumulate registers are reset to ensure the accumulated sums always start at 0 for each new pair of vectors. After the initial propagation delay to fill the pipeline, it takes 32 clock cycles for the module to compute the dot product.

Ideally, we would have liked to fully utilize the parallelism of the FPGA by using 32 CMUL blocks in parallel to perform the multiplications. After the initial propagation delay, all required multiplications are performed within the same clock cycles and the products can be added to form the dot product. However, this would require high usage of hardware which is quite limited in the Nexys A7-100T board. Since we expected the other matrix operations, such as the matrix inverse, to require significant amount of hardware we had to limit the resources used by each module as much as possible.

4.4.4 Scatter matrix

As seen in (2.29), the scatter matrix $\boldsymbol{S}$ for K secondary data cells $\boldsymbol{z}_k$ is computed as

$$\boldsymbol{S} = \sum_{k=1}^{K} \boldsymbol{z}_k \boldsymbol{z}_k^{\dagger}. \tag{4.7}$$

Since $\boldsymbol{z}_k \in \mathbb{C}^{M\times 1}$, this operation requires column-row vector multiplication. Given an arbitrary column vector $\boldsymbol{x}$ and an arbitrary row vector $\boldsymbol{y}$, each with M elements, this multiplication results in an $M \times M$ matrix $\boldsymbol{A}$:

$$\boldsymbol{A} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \times \begin{bmatrix} y_0 & y_1 & \cdots & y_{N-1} \end{bmatrix} = \begin{bmatrix} x_0y_0 & x_0y_1 & \cdots & x_0y_{N-1} \\ x_1y_0 & x_1y_1 & \cdots & x_1y_{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-1}y_1 & x_{N-1}y_2 & \cdots & x_{N-1}y_{N-1} \end{bmatrix}. \tag{4.8}$$

Thus, to form row i in $\boldsymbol{A}$ we need to multiply x_i with each element in $\boldsymbol{y}$.

Furthermore, for $\boldsymbol{S}$ we need to accumulate the results of each row-column vector multiplication to form the final scatter matrix. To realize this in hardware, a VHDL module was designed which uses a Mealy-type FSM to control the sequence of complex multiplications and accumulate operations. A block diagram of this module is shown in Fig. 4.9.

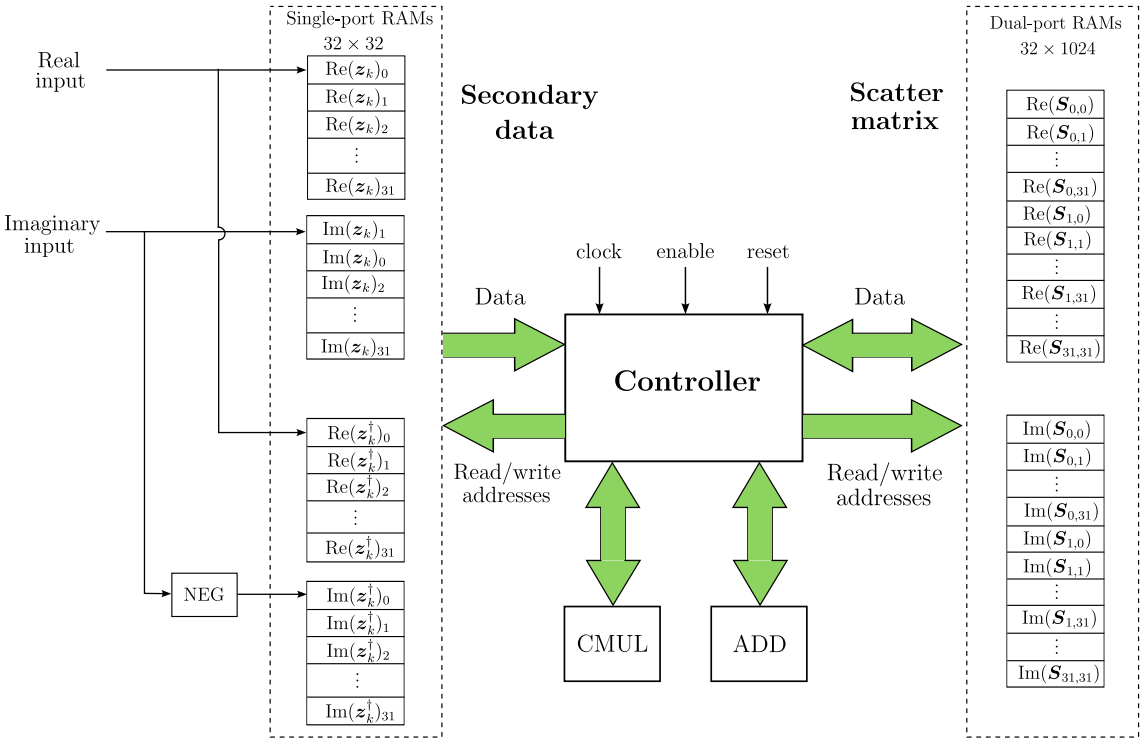


Figure 4.9: Scatter matrix module block diagram.

The FSM starts by loading one secondary data cell of 32 elements to two pairs of single-port BRAMs if the `enable` signal is asserted. These contain the real and imaginary parts of $\boldsymbol{z}_k$, and the real and imaginary parts of the complex conjugate of $\boldsymbol{z}_k$. The conjugate was achieved simply by negating the imaginary channel. Note that both $\boldsymbol{z}_k$ and $\boldsymbol{z}_k^\dagger$ are stored as column vectors in the RAMs, but we have to remember that the $\boldsymbol{z}_k^\dagger$ RAMs should be treated as a row vector.

Two synchronous counters read the addresses from the two pairs of input RAMs. One is the column counter of $\boldsymbol{z}_k^\dagger$, which increments every clock cycle, and the other is the row counter of $\boldsymbol{z}_k$ which increments once all elements in $\boldsymbol{z}_k^\dagger$ have been accessed. This is done to achieve the complex divisions for one row of the $\boldsymbol{S}$ matrix each iteration, as shown in (4.8).

Two dual-port BRAMs are used to contain the real and imaginary parts of $\boldsymbol{S}$, stored as stacked rows. All elements in these RAMs are initialized to zero at the start of the program. Each resulting product from the complex multiplications are added to the previous value stored at the corresponding address of the $\boldsymbol{S}$ RAMs before being stored at that address.

This process repeats until all K secondary data cells have been processed, after which an output flag `tvalid` is asserted and the finished $\boldsymbol{S}$ matrix will be transmitted one element at a time to the output ports of the module.

4.4.5 Matrix inverse

There exists multiple different methods for determining the inverse of a square matrix. In our design we decided to use the Gauss-Jordan (GJ) elimination method. The reasoning behind this selection is that the algorithm only uses the standard mathematical operations such as multiplication, division, and subtraction and avoids introducing further complexities such as linear matrix transformations that are common in other methods, such as the Cholesky decomposition. Before explaining the hardware design, an overview of the GJ method is warranted in case the reader is not familiar (or recalls) the algorithm.

As an example, consider an invertible square 3×3 matrix $\boldsymbol{A}$. Augmenting $\boldsymbol{A}$ with its identity matrix $\boldsymbol{I}$ creates a 3×6 augmented matrix:

$$[\boldsymbol{AI}] = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & 1 & 0 & 0 \\ a_{1,0} & a_{1,1} & a_{1,2} & 0 & 1 & 0 \\ a_{2,0} & a_{2,1} & a_{2,2} & 0 & 0 & 1 \end{bmatrix}. \tag{4.9}$$

The GJ elimination algorithm performs operations on the augmented matrix to row-reduce matrix $\boldsymbol{A}$ to its identity matrix. Once it has finished, the remaining matrix (previously constituting $\boldsymbol{I}$) will be $\boldsymbol{A}^{-1}$. The number of iterations required for the algorithm to converge for any given $N \times N$ matrix is equal to N. In each iteration i the following operations are performed:

1. If pivot element in row i is a 0, swap the row with another row containing a nonzero element in the same position.
2. Divide row i of $[\boldsymbol{AI}]$ by the pivot element, which results in a 1 in the pivot position.
3. In order to eliminate all elements above and below the pivot, multiply all elements above and below the pivot element one at a time with row i and subtract the results from the corresponding multiplier's row.

Steps 1–3 are repeated until $i = N$.

For an $N \times N$ matrix, each iteration of the GJ elimination requires $2N$ complex division operations, $2N(N-1)$ complex multiplication operations, and $2N(N-1)$ complex subtraction operations; making it a costly algorithm in terms of computational complexity. However, as proven in [7], the inverse matrix can be obtained by only performing operations on the critical elements of the matrix. By avoiding division and multiplication operations which would result in a 0 or a 1, the number of operations can be cut in half.

Additionally, since we are only computing the inverse of $\boldsymbol{S}$, which is Hermitian positive-definite, we can neglect Step 1 since the matrix by definition does not contain any zero elements in the pivot positions. The only way for a 0 to appear in a pivot position is if it is the result of the subtraction in the previous iteration. However, since the scatter matrix consists of noise samples it is unlikely that the previous multiplication would result in a subtrahend exactly equal to the minuend at that pivot position.

If we again consider the augmented matrix in (4.9), the first iteration of the improved GJ algorithm is shown in Fig. 4.10

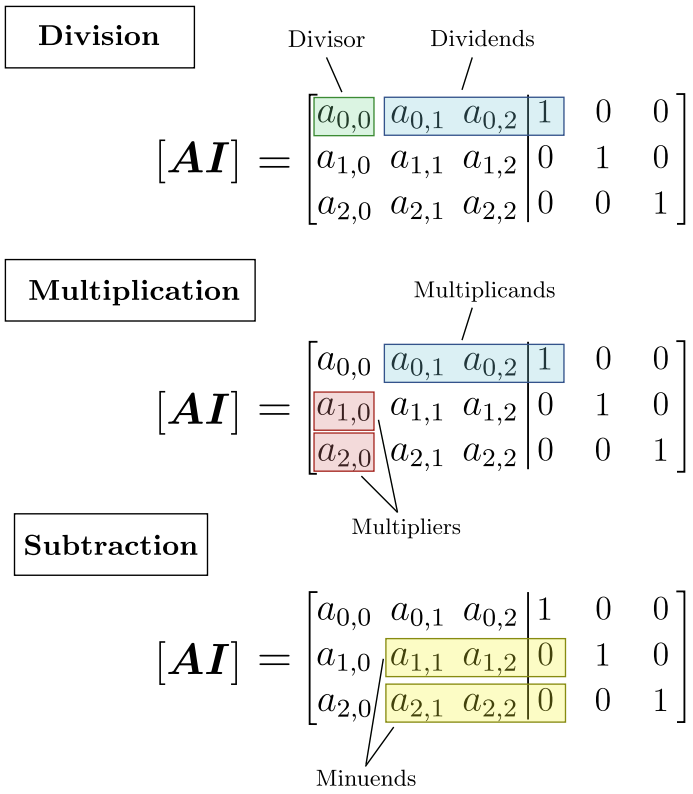


Figure 4.10: First iteration of the improved GJ elimination algorithm.

As can be seen from the figure, we ignore operations on the pivot element $a_{0,0}$ and only perform division, multiplication, and subtraction on elements up to and including the pivot column of $\boldsymbol{I}$. In the next iteration we move to the next pivot element, so the divisor (green box in Fig. 4.10) will be $a_{1,1}$, the dividends and multiplicands (blue boxes) will start at $a_{1,2}$, the multipliers (red boxes) will be $a_{0,1}$ and $a_{2,1}$, and the minuends (yellow boxes) will start from $a_{0,2}$ and $a_{2,2}$. This process is then repeated for the final pivot to obtain the inverse matrix. Thus, this improved version also takes N iterations to compute the inverse matrix, but requires half the operations each iteration.

The described improved GJ elimination algorithm was first functionally verified in MATLAB before being implemented in VHDL by using a Mealy-type FSM to control the flow of the program. Given that in our case $\boldsymbol{S}$ is a 32×32 matrix, it takes 32 iterations to obtain its inverse. At the beginning of the process, the real and imaginary parts of $\boldsymbol{S}$ are loaded to two separate BRAMs. Due to the row-based nature of the GJ algorithm, $\boldsymbol{S}$ is stored as stacked rows in the BRAMs for easier indexing, which required a RAM depth of 1024. Additionally, two other BRAMs are loaded with the 32×32 complex identity matrix $\boldsymbol{I}$ during startup. A block diagram of the GJ inverse module is shown in Fig. 4.11.

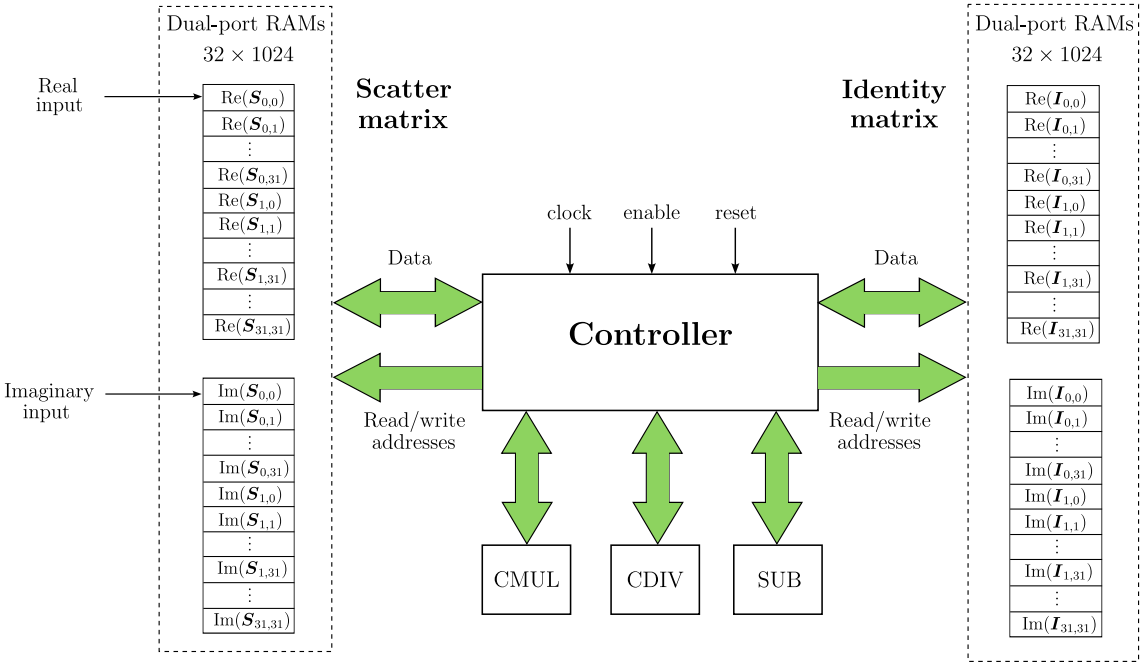


Figure 4.11: Matrix inverse module block diagram.

The controller utilizes synchronous counters to access different read and write addresses of the four BRAMs depending on the state of the FSM. For easier control, lookup tables (LUTs) were defined which contain the addresses for all pivot elements, the first element in each row, and last element in each row. This allows the controller to keep track of which pair of RAMs should be accessed when reading or writing data.

Each iteration the correct elements are read from the RAMs and complex division, multiplication, or subtraction operations are performed depending on the state. To avoid possible memory corruption, each time a row is being multiplied the products are stored in separate RAMs of depth 32 which allowed easier indexing when performing the subtraction step.

When all 32 iterations have been completed, an output flag `tvalid` is asserted and the finished $\boldsymbol{S}^{-1}$ will be transmitted to the output ports one element at a time.

4.4.6 Verification

In order to verify the functionality of each design, test vectors had to be created to be used as stimuli in test benches. For each module, pseudo-random complex test vectors with their expected output for the given operation were generated. Before storing the results as binary words in text files to be used for verification, each test set was converted to same s13.18 fixed-point format used in the VHDL design using MATLAB's `Fixed-point Designer` toolbox.

This quantization was done in order to ensure that any possible errors produced by the RTL modules were not due to the difference between the default double-precision floating point representation in MATLAB and the fixed-point format of the

modules. Any observed errors during verification process could then be concluded only be caused by the logic used in the RTL designs, not the data format.

Once all modules passed behavioral verification, their resulting outputs were stored in text files and compared to the floating point results in MATLAB in order to determine the error of the RTL modules compared to the results produced by MATLAB.

Different metrics for the error were used depending on the design under test (DUT). For single outputs such as the complex multiplication or division, we used the relative error ϵ_{r}, defined as:

$$\epsilon_{\mathrm{r}} = \left| \frac{y - \hat{y}}{y} \right|, \tag{4.10}$$

where y is the true result and $\hat{y}$ is the approximated result [34].

For computed matrices, such as $\boldsymbol{S}$ or $\boldsymbol{S}^{-1}$, the Frobenius norm error was used in order to get a single quantifiable metric of how the estimated and actual matrix differ as a whole. The Frobenius norm of an $N \times N$ matrix $\boldsymbol{A}$ is defined as

$$||\boldsymbol{A}||_{\mathrm{F}} = \sqrt{\sum_{i=1}^{N} \sum_{j=1}^{N} a_{i,j}^2}. \tag{4.11}$$

The Frobenius norm error, denoted by e_{F}, of the actual matrix $\boldsymbol{A}$ and the estimated matrix $\hat{\boldsymbol{A}}$ is then

$$\epsilon_{\mathrm{F}} = \frac{||\boldsymbol{A} - \hat{\boldsymbol{A}}||_{\mathrm{F}}}{||\boldsymbol{A}||_{\mathrm{F}}}. \tag{4.12}$$

Additionally, the relative element-wise error of the two matrices can be computed iteratively using (4.10) to compute the average element error.

4.5 DPU Implementation

The pre-trained NN model was saved as an *open neural network exchange* file (ONNX), which is an open source standard framework for representing machine learning models. ONNX allows models developed in frameworks such as `PyTorch` or TensorFlow to be converted into a unified representation that can be efficiently deployed across a wide range of platforms, including embedded systems and custom inference accelerators.

In this project, the ONNX model served as a necessary intermediate step to enable deployment on the Xilinx VCK190 evaluation board using Vitis AI – a development environment designed for optimized AI inference on Xilinx FPGAs and adaptive SoCs. Vitis AI requires that models be pre-trained and saved in ONNX format for before they can be compiled for deployment.

By applying integer quantization, deploying NNs on AMD Deep Learning Processing Units, or DPUs, becomes more efficient as this reduces energy consumption, memory requirements, and the data bandwidth needed for inference. Post-training quantization is carried out by the Vitis AI Quantizer, which functions as a component within `PyTorch`. During this process, a small portion of the original training

data – typically ranging from 100 to 1000 unlabeled samples – is used to forward propagate through the network. This step allows the quantizer to examine the activation distributions at each network layer. Once this analysis is done, the network's weights and activations are converted into 8-bit integer format (INT8). Once quantization is complete, the model's accuracy is tested again with the validation dataset. If the accuracy stays within an acceptable range, the quantization process is considered finished. This objective was not successfully accomplished because, despite thorough testing and investigation, the quantization package for PyTorch utilized by the Vitis AI framework was incompatible with the implemented PyTorch model. Attempts to use it were unsuccessful, preventing effective implementation of the quantization process.

Vitis AI Compiler converts the model into an internal computation graph using the Xilinx Intermediate Representation (XIR). The compiler schedules instructions efficiently to leverage the DPU's parallelism and data reuse. During compilation, `PyTorch` models are transformed into XIR, with the compiler handling framework-specific differences, optimizing the graph, and partitioning it into subgraphs. For subgraphs compatible with the DPU, the compiler uses a DPU *arch.json* file that defines the target architecture, in this case the VCK190's DPU (DPUCVDX8G). It generates instruction streams and outputs a compiled *.xmodel* file. The compilation was carried out using the **vai-c-xir** tool, which converts the ONNX model into an optimized binary format tailored for the configured DPU on the VCK190 board. In this implementation, the model was compiled using single-precision floating point format (FP32), which is supported by the selected DPU configuration, since model quantization could not be carried out.

We investigate the effects of the NN model trained in FP32 after it has been quantized in Python to INT8 format, with the aim of simulating how the model would theoretically perform if implemented on the VCK190 board. The method is based on a standardized pipeline for static post-training quantization in PyTorch. This is a prerequisite to enable a fair comparison with models intended to run on embedded hardware. By analyzing inference time, performance metrics, and changes in the model weights, one can assess how quantization affects accuracy in relation to the improvements in speed and memory consumption. In practice, quantization means that the model's weights (and sometimes also activations) are scaled down from a large numerical range to a smaller interval with coarser resolution.

This process takes three steps:

1. Original weights (FP32) : The exact floating point values the model was trained with.
2. Quantized weight (INT8) : The weights are scaled and rounded to integers within $[-128, 127]$, which requires much less memory and is faster to calculate.
3. De-quantized weight (FP32): To understand and analyze the precision of the model after quantization, we restore the INT8 values to floating point format.

5

Results

In this chapter we present all results from our implemented designs. The performance of the trained NN detector implemented in Python compared with Kelly's detector is presented in Section. 5.1 and the results from the hardware implementation of parts of the CFAR-FP mapping chain is summarized in Section 5.2.

5.1 NN Detector

Three primary datasets were generated by the MATLAB model to serve as training and validation data for the NN. Each set consisted of one rotation's worth of gathered data, with different rotations used for each dataset. Due to the time required for one rotation of the antenna not being exactly equal for each rotation, the number of samples differs slightly in the three datasets but is roughly 13,000 samples per dataset.

Using our MATLAB model, we generated clusters of equal size for H_0, H_1, and mismatched H_1 using the method as described in Section 4.2. For each dataset two different mismatches were generated so that the mismatched cluster contained two different $\cos^2\theta$ values – one for each half of the swept range bins. The three datasets were designed in terms of detection difficulty rated from easy, medium, to hard[1]. The detection difficulty depends on both the SNR and the level of mismatch. Table 5.1 shows the parameters chosen for each dataset.

Table 5.1: Parameters for the easy, medium, and hard datasets generated in MATLAB.

Name	**SNR**	$\cos^2\theta$
Easy	20 dB	0.6 & 0.8
Medium	20 dB	0.4 & 0.7
Hard	15 dB	0.3 & 0.6

We decided to use the medium difficulty dataset as training data for the NN, as it is in the middle of two extreme scenarios, with 80 % of the set being used for training,

[1]The datasets were defined for typical SNR values in radar signals, with two levels of mismatch in each set. The mismatch levels were selected to fit a wide range of potential mismatches without being too close to the true H_0 or H_1 cases.

and 20 % for validation during the training process. Training the NN on the easy dataset resulted in it being too optimistic in its predicted decision boundary, and using the hard dataset increased the risk of overfitting. The NN was configured with two input nodes, two hidden layers with 64 nodes each, and one output node. The training and validation loss and accuracy were plotted as a function of epochs to determine model performance. A typical response while using batch normalization between the hidden layers can be seen in Fig. 5.1.

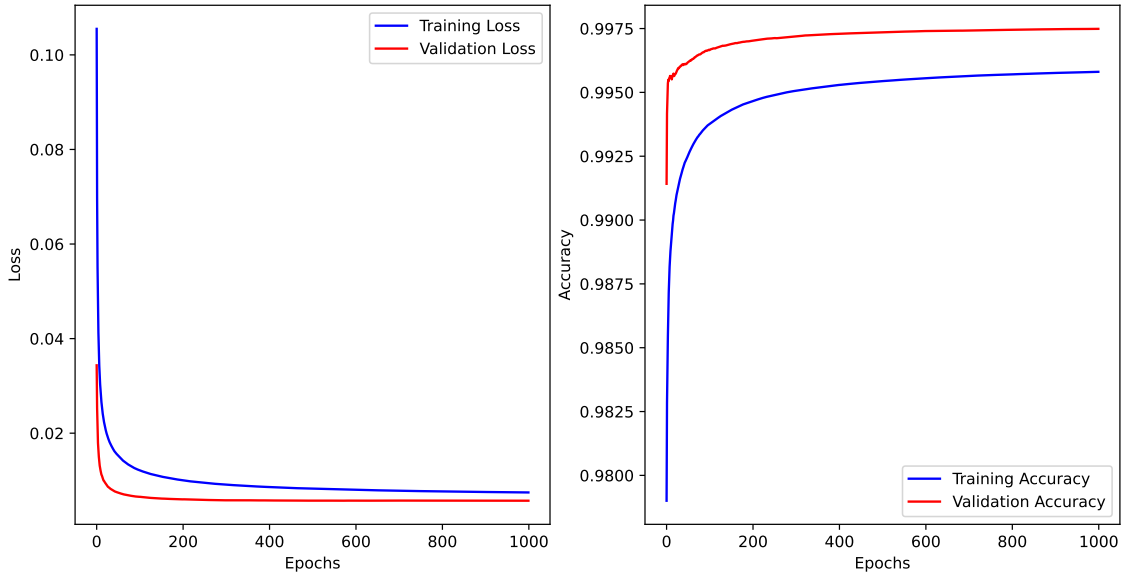
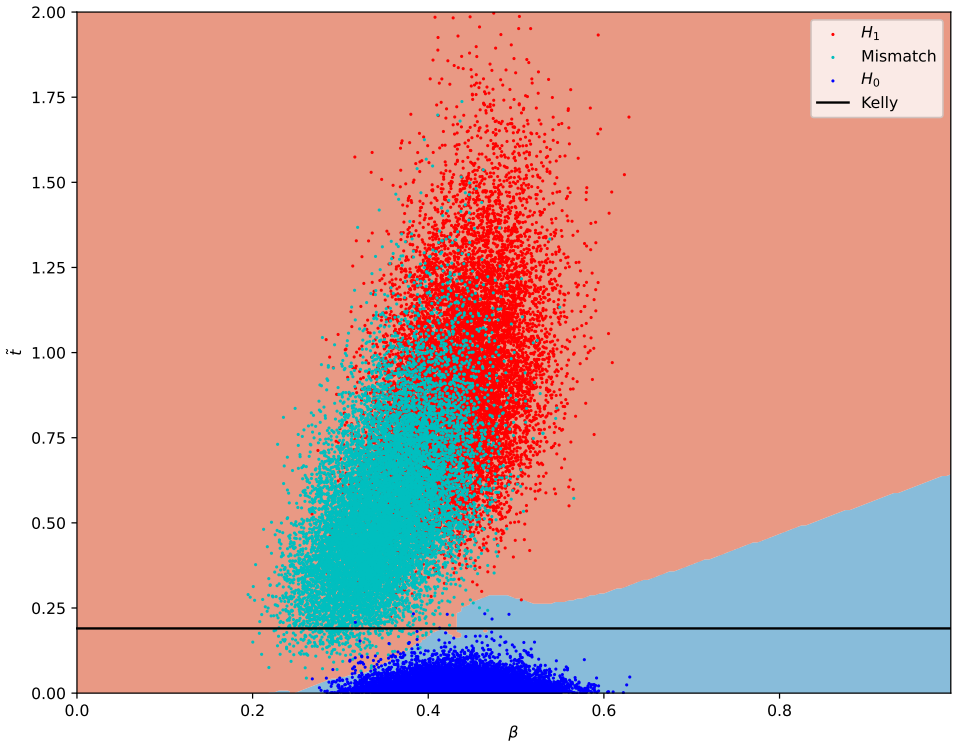
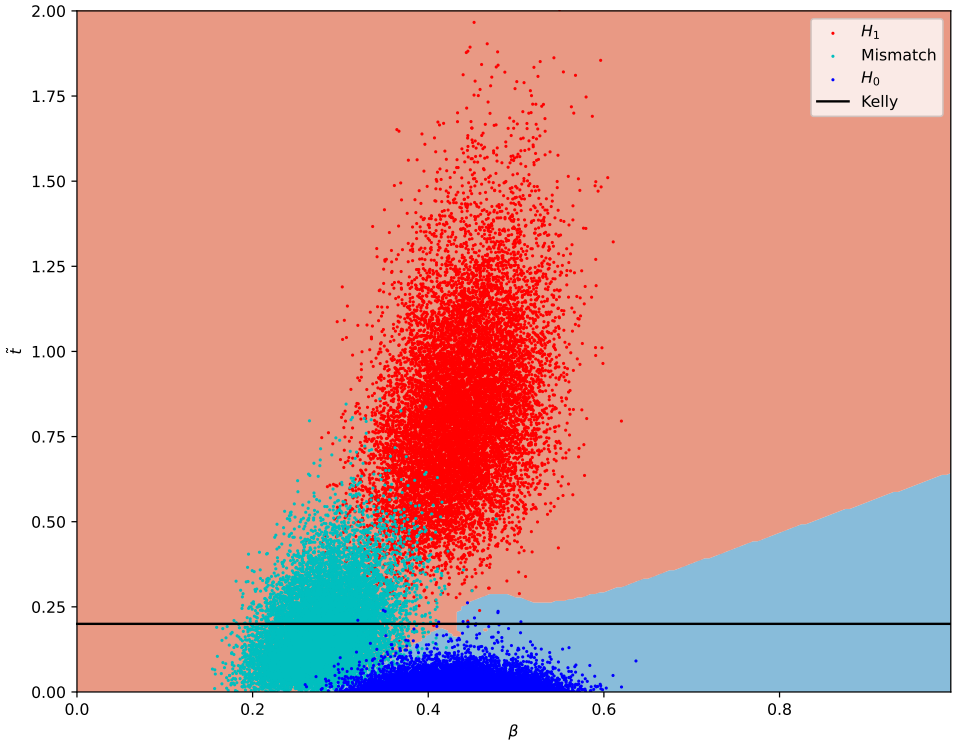


Figure 5.1: Training and validation loss per epoch (left), and training and validation accuracy per epoch (right) on the medium dataset.

After training, the easy and hard datasets were used as inference with the trained model. In Fig. 5.2a, the easy dataset is plotted together with the NN-generated decision boundary and the Kelly detector's threshold. All points in the red region of the figures are classified as H_1, while all points in the blue region are classified as H_0. For this dataset the optimal Kelly threshold for minimum P_{FA} was $\tilde{t} = 0.2$. As can be seen, the threshold produced by the NN has quite successfully separated the union of the matched and mismatched H_1 clusters from the H_0 cluster, indicating high levels of robustness. In contrast, as previously mentioned in Section. 2.3.4, the selective property of Kelly's detector is clearly demonstrated where a greater part of the mismatched H_1 cluster is classified as H_0. Fig. 5.2b shows a similar plot but for the hard dataset instead, with a Kelly threshold of $\tilde{t} = 0.17$.



(a) Decision boundaries of Kelly's detector and the NN in the CFAR-FP of the easy dataset.



(b) Decision boundaries of Kelly's detector and the NN in the CFAR-FP of the hard dataset.

Figure 5.2: Decision boundary created by the NN trained on the medium dataset (H_0 region in light blue and H_1 region in light red) together with Kelly's threshold (black line) of the easy (a) and hard (b) datasets.

The performance results of the model, using the easy and hard datasets during inference are summarized in Tables 5.2 and 5.3. The results also compare the model's performance before and after quantization, where FP32 represents the original floating-point representation and INT8 represents the quantized version with 8-bit integers. The inference time for the FP32 and quantized model was also measured to observe the improvement in computational speed for the quantized model. Inference time for the unquantized model was 41.65 ms and only 3.45 ms for the quantized NN, which means the quantized model was approximately twelve times faster than the original model.

Table 5.2: Results of the NN model and Kelly's detector on the easy dataset.

Detector	P_{FA}	P_{D}	**Accuracy**	F_1	**Data type**
NN	$3.7 \cdot 10^{-3}$	0.9997	0.9986	0.9989	FP32
Kelly ($\tilde{t} = 0.2$)	$4.6 \cdot 10^{-4}$	0.9874	0.9914	0.9935	FP32
NN (quantized)	$2.29 \cdot 10^{-4}$	0.9964	0.9073	0.9982	INT8

Table 5.3: Results of the NN model and Kelly's detector on the hard dataset.

Detector	P_{FA}	P_{D}	**Accuracy**	F_1	**Data type**
NN	$4.3 \cdot 10^{-3}$	0.9875	0.9903	0.9927	FP32
Kelly ($\tilde{t} = 0.17$)	$1.2 \cdot 10^{-3}$	0.7007	0.8000	0.8237	FP32
NN (quantized)	$1.5 \cdot 10^{-4}$	0.7082	0.6683	0.8292	INT8

Zooming in on the CFAR-FP plot (see Fig. 5.3), we can see that the complexity of this dataset emphasizes the benefits of using the decision boundary generated by the NN model. Evidently, the boundary outlines the matched and mismatched H_1 clusters closely, which would have been hard to achieve with a conventional detector.

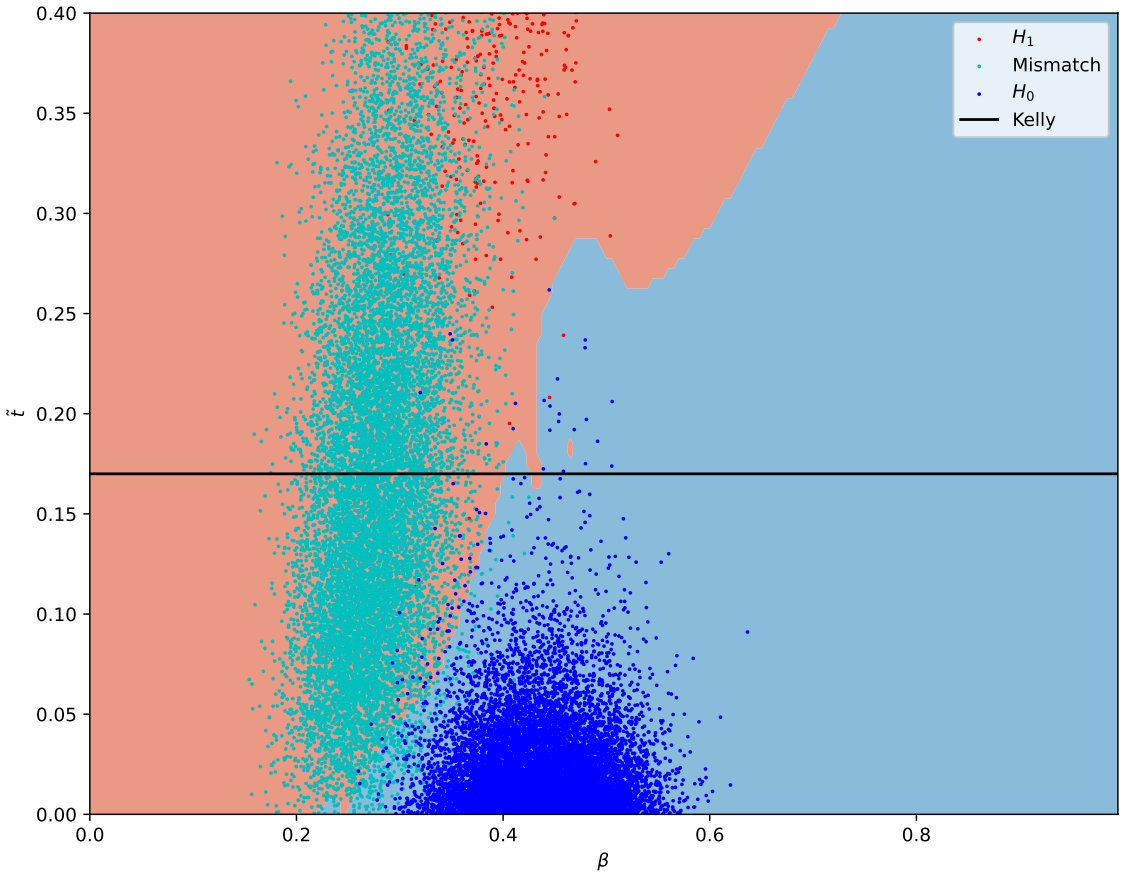


Figure 5.3: Zoomed in plot of the decision boundaries of Kelly's detector and the NN in the CFAR-FP of the hard dataset.

A list of all the NN model's weights was extracted in order to evaluate the effects of quantization. Once the weights had been quantized to INT8 they were then dequantized back to FP32 to observe the loss in precision. The first three weights of this list are presented in Table 5.4. As can be seen, the quantization resulted in small numerical errors.

Table 5.4: The first three weights before quantization, after quantization and dequantization on the hard dataset.

Weight type	**Weights**	**Data type**
Original	$[-0.2444, 0.4084, -0.6024, \dots]$	FP32
Quantized	$[-11, 18, -27, \dots]$	INT8
Dequantized	$[-0.2496, 0.4084, -0.6125, \dots]$	FP32

5.2 RTL Design Results

In this section we present the results from the implemented RTL modules described in Section 4.4. Each module was designed using VHDL and implemented in Vivado using the Nexys A7-100T as the target device.

The behavioral functionality of the CMUL and CDIV modules was verified by created test benches using 10,000 random fixed-point complex test vectors, with real and imaginary parts in the range of $[-50, 50]$. The CMUL block generated exactly the same output as the expected fixed-point golden reference, while the CDIV module sometimes had minor errors of the real and imaginary outputs. These small errors were expected as the CDIV includes six multiplier blocks, with each output being rounded before the divider blocks.

The mean relative error of the fixed-point results compared to the their corresponding floating point golden reference in MATLAB was also computed to determine any loss in precision between the two formats:

$$\begin{cases} \epsilon_{\mathrm{r}}(\mathrm{CMUL}) & \approx 7.574 \cdot 10^{-8} \\ \epsilon_{\mathrm{r}}(\mathrm{CDIV}) & \approx 1.927 \cdot 10^{-6} \end{cases}$$

Synthesis for CMUL and CDIV was performed, with a 100 MHz clock for the CMUL module and a 400 MHz PLL clock generated by the Clock Wizard for the CDIV module. Both modules passed synthesis with no timing violations.

The CMUL module depicted in Fig. 4.6 also includes input registers, pipeline registers within the multiplier blocks, registers for storing the results from the adders, and a final pair of output registers which stores the values after truncating and rounding the result. The minimum propagation delay that can be achieved with this setup is five clock cycles, when the the multiplier cores are configured with only one pipeline stage. However, for optimal performance Vivado recommended five pipeline stages for the multipliers, which then results in a total delay of ten clock cycles instead. Similarly, the CDIV module shown in Fig. 4.7 also includes the equal amount of registers as the CMUL module, with additional pipeline registers for the divider cores. To achieve the minimum throughput of one new output every four clock cycles, five pipeline stages were used for the divider cores. The minimum achievable propagation delay with this setup amounted to ten clock cycles, assuming one pipeline stage was used in the multipliers. In other words, the CDIV has twice the propagation delay as the CMUL module.

The utilization of the two designs after synthesis are summarized in Table 5.5.

Table 5.5: Utilization of the CMUL and CDIV RTL modules on the Nexys A7-100T as percentages of maximum resources.

Name	LUTs (%)	Registers (%)	BRAM Tiles (%)	DSPs (%)
CMUL	4.19	2.70	0	1.67
CDIV	8.51	4.67	1.48	11.7

The dot product module was verified for 1000 random 32×1 complex vectors, with real and imaginary parts in the range of $[-10, 10]$. Each computed dot product was compared with the expected fixed-point result, and no errors were detected. Since the module only uses adders and the CMUL block, the relative error's order of magnitude will be the determined by the CMUL block.

The module for computing $\boldsymbol{S}$ also passed functional verification, with one random set of 64 32×1 secondary data vectors with real and imaginary parts also ranging $[-10, 10]$. Similarly, the GJ inverse module was verified for one randomly generated 32×32 matrix with real and imaginary elements in the range $[-2, 2]$, while also ensuring nonzero elements in the diagonal as expected by a true scatter matrix. However, it did not pass verification for higher values such as $[-50, 50]$ for example, the reason for this will be addressed in Chapter 6. From enable being asserted up to the final computed element being available at the output ports, the scatter matrix module takes approximately 70,400 clock cycles (704 µs given a 100 MHz system clock). For the GJ module it instead takes roughly 89,900 clock cycles (899 µs given a 100 MHz system clock).

The Frobenius norm error and average relative element-wise error of the fixed-point results produced by the VHDL modules compared to corresponding floating point result in MATLAB for the scatter matrix and GJ inverse modules are summarized in Table 5.6

Table 5.6: Error of the implemented VHDL modules for computing the scatter matrix and matrix inverse.

Name	ϵ_{F}	ϵ_{r}
Scatter matrix	$1.600 \cdot 10^{-7}$	$4.336 \cdot 10^{-7}$
GJ inverse	$9.614 \cdot 10^{-5}$	$1.801 \cdot 10^{-4}$

The dot product, scatter matrix, and GJ inverse modules passed synthesis with a 100 MHz system clock. However, the GJ inverse module had some severe timing violations. This was caused because certain necessary design considerations had not been taken into account for the module's use of two different clock domains in the complex division step. Since time was limited at this stage of the project, these timing violations were never corrected. However, possible solutions are discussed in Chapter 6.

The total utilization for the dot product, scatter matrix, and GJ inverse modules are listed in Table 5.7. Note that since the CMUL and CDIV modules are part of these designs in various ways, they are included in the total utilization.

Table 5.7: Utilization of the dot product, scatter matrix, and GJ inverse RTL modules on the Nexys A7-100T as percentages of maximum resources.

Name	LUTs (%)	Registers (%)	BRAM Tiles (%)	DSPs (%)
Dot product	4.41	2.91	0	1.67
Scatter matrix	4.60	2.95	2.96	1.67
GJ inverse	13.8	8.03	5.19	13.3

6

Discussion

In this chapter we discuss and analyze the results presented in Chapter 5.

6.1 NN Detector Performance

One of the aims for this thesis was to develop an NN model trained on raw radar data, designed with hardware deployment in mind. To make the implementation effective, the goal was to minimize the number of hidden layers and nodes without significantly compromising performance. In Chapter 5, we present the performance metrics yielded by our NN model consisting of two hidden layers with 64 nodes each. Although various NN configurations were evaluated, the final architecture was chosen based on a balance between performance and hardware suitability.

The model was evaluated with two datasets with different degree of detection difficulty in terms of the level of SNR and mismatches. Its detection performance was then compared with the traditional Kelly detector optimized for minimum P_{FA}, as shown in Tables 5.2 and 5.3. The NN achieved a higher P_{D} than Kelly's detector in both cases, demonstrating its robust characteristics, while Kelly's detector achieved a lower P_{FA} in both cases. Since Kelly's threshold was chosen based on its lowest P_{FA}, and that it is a selective detector, this was somewhat expected. In terms of detection probability, the NN model's performance far exceeded Kelly's detector on the hard dataset with high P_{D}, accuracy, and F1-score. Since the NN was trained to separate the union of mismatched and matched H_1 clusters from H_0, it fundamentally realizes a robust detector, making it more suitable to mismatched targets. Since Kelly's detector is a selective detector, its P_{D} in the case of the harder dataset is far lower. Overall, the NN shows promising results. Even for the hard dataset with severe levels of mismatch and lower SNR the NN achieved a P_{D} of 98.75 % – 28 points higher than the Kelly detector, while still achieving similar P_{FA}.

The results before and after quantizing the NN to INT8 format were also observed in order to detect any degradation in detection performance. The quantized model represents the learned patterns with lower precision, which can lead to some borderline cases previously being classified as positive now falling below the threshold and instead being classified as negative. This change is important to consider when implementing on hardware such as the Xilinx VCK190 where quantization is necessary to run in real time. Detection capability must be weighed against the system's requirements for reliability and efficient resource use. In the case of the easy dataset

in Table 5.2, we note that the loss in precision for the quantized NN did not significantly impact performance compared to the floating point model. The P_{D} and F1-score are only slightly lower for the quantized model, however, the accuracy dropped by almost a factor 10. The P_{FA} was also lower for the quantized model. However, since the inference time dropped by almost 40 ms for the quantized model, it shows that a quantized model could be suitable for scenarios with relatively high SNR and lower mismatch levels as it achieves similar results as the original FP32 model much faster.

In contrast, the results in Table 5.3 from inference on the hard dataset show that the quantized model's loss in precision heavily impacted detection performance. The FP32 model still managed to achieve a rather high P_D with a relatively low P_{FA}. The quantized model still achieved a very low P_{FA} of $1.5 \cdot 10^{-4}$, but with a P_D of only 70.82 %; only slightly outperforming the Kelly detector. This low P_D suggests that the model becomes more cautious in classifying positive cases after quantization. Due to the fact that the hard dataset includes heavily mismatched signals and lower SNR, the matched and mismatched H_1 clusters are much closer to the H_0 cluster. Thus, the classification decision could depend on extremely small variations in β and $\tilde{t}$. For the NN this would require more finely adjusted weights to improve performance, which is lacking in the quantized model.

Our proposed NN model exhibits a more adaptive, non-linear decision boundary that contours tightly around the H_0 cluster, thereby isolating the combined mismatched and matched H_1 clusters. In more complex scenarios, such as in the hard dataset, where clusters are grouped closer to each other, we can clearly see the benefits of using an adaptive, robust detector. However, testing with other larger, more diverse datasets and comparing the results with other well-known detectors is needed to ascertain that the NN outperforms traditional methods for detection. Before such comprehensive tests have been made, we cannot conclude that the computational load added by an NN is worth the effort.

6.2 Evaluation of RTL Results

The VHDL modules implemented for the CFAR-FP processing chain passed functional verification as described in Section 5.2, with most having a relative error in the range of the resolution of the fixed-point format compared to the floating point results. The greatest computational error was found in the GJ inverse module, which was not surprising given the vast amount of multiplications and divisions being performed by the algorithm.

Timing violations were also observed for the GJ inverse module. This occurred because no steps were taken to ensure synchronicity between the two different 100 MHz and 400 MHz clock domains used in the module, which might introduce metastability issues. In order to correct this, one possible solution is to introduce input and output control signals to the CDIV module which are synchronized to the 100 MHz using D-flip flips. This would ensure that input and output data are synchronized to the system clock. However, adding more registers will delay the signal even more which might remove any benefits added from using two clocks. A more reasonable

design choice would probably be to let the complex division take four clock cycles and account for this delay within the FSM instead.

The main issue with the GJ inverse module was that functional verification failed when a real scatter matrix $\boldsymbol{S}$ generated from the radar data was used. We concluded that the reason behind this could be explained by how the CDIV module operates. If we observe the block diagram for the CDIV in Fig. 4.7, we see that the two multipliers in the middle of the first layer compute the square of both the real and imaginary parts of the divisor. We had neglected the fact that the pivot elements of $\boldsymbol{S}$ are in the scale of 10^3, which means that when division is being performed as part of the GJ algorithm the divisor is also in this scale. As a result, the two multipliers in the CDIV module will produce a product in the scale of 10^6 which is far greater than the representable value of our s13.18 fixed-point format. Thus, these divisions will cause extreme overflow, generating incorrect results. From this we could conclude that our estimation of the required number of bits for the fixed-point representation was too low, and will need to change in order for the design to function when real radar data is used.

6.3 Reflections

At the start of this project, we outlined the expected stages and timeline in a Gantt chart (Appendix A), based on our four goals listed in Section 1.4. This aided us in planning and structuring our work.

Initially, our objective was to perform some preprocessing of the experimental data and determine how to integrate it into the CFAR-FP mapping chain in MATLAB. However, the limited knowledge of clutter, targets, and noise in the data introduced challenges we had not anticipated. Available literature relating to the CFAR-FP is sparse, which made it challenging to obtain comprehensive insight in how the mapping chain could be adapted to our experimental data. For example, the original work [1] relied exclusively on simulated ideal Gaussian noise and clutter, making direct application to real-world data difficult – particularly when defining the clutter covariance matrix. Together with our advisors at Saab, we finally decided to simplify the labeling process for the NN by injecting synthetic targets into the dataset and neglecting range bins with high levels of clutter. However, reaching this conclusion proved to be significantly more time-consuming than expected, which delayed the design phase of this project.

To accommodate potential delays or unforeseen issues, the project plan included a time slot labeled *Optimize Performance* in the Gantt chart shown in Appendix A. This slot was dedicated either for resolving unexpected challenges during development or for improving system performance. Due to the delays in system development, this time was utilized to attempt to finish the RTL design and to evaluate the performance of a quantized NN. Furthermore, the lengths of the time slots dedicated to setting up the development environment and hardware turned out to be too ambitious. In order to deploy designs to the Versal VCK190 board, the Vitis development environment has to be used. This high-level synthesis (HLS) tool includes a multitude of possible configurations and tool settings that need to be properly

understood for any specific application. As we did not have any prior experience or knowledge of Vitis, setting up the VCK190 board correctly proved difficult and time-consuming. Because significant time had already been spent on the initial system development in MATLAB and Python, there was insufficient time remaining to fully deploy the complete system onto the VCK190 board and explore the potential for streaming data through the pipeline. As a result, these tasks are identified as future work in Section 7.1.

In hindsight, the project plan could also have benefited from including a time slot for generating CFAR-FP datasets and training the NN with different training data and model configurations. Both of these tasks required significant processing time – especially for larger datasets – and thus became a bottleneck in the project. For more efficient use of time, we should have early defined desired datasets and different NN models to test. Furthermore, the NN testing may have been optimized by performing automated tests on the different models and storing the results, which would have saved significant time when determining the optimal number of hidden layers and nodes in the network.

7

Conclusion

In this work we have explored the possibility of implementing a robust radar detector based on a supervised ANN trained on a CFAR-FP dataset generated from real radar data, with the ultimate goal of implementing the system on Xilinx AI core series FPGA VCK190 for edge signal processing. A fully functional system model has been implemented at system level, which includes a MATLAB program for generating the CFAR-FP with customizable targets inserted at desired range bins as well as an NN model implemented in Python.

The NN model was evaluated with different datasets of various SNR and mismatch levels, and showed promising results. The NN outperformed the traditional Kelly detector in terms of detection probability. Overall, our results prove that while the Kelly detector remains a strong baseline for detection, NN-based detectors offer superior robustness and adaptability, especially when operating in scenarios with low SNR and high levels of signal mismatch. This suggests that ML-based solutions could possibly play a critical role in modern detection systems. We have also showed that quantizing an NN for implementation on resource-constrained platforms, such as the VCK190, speeds up inference time significantly. However, for difficult datasets the prediction accuracy degrades quite drastically.

RTL designs for performing advanced complex valued linear algebraic operations on an FPGA have been implemented. Utilizing said modules could significantly speed up the signal processing chain compared to traditional software implementations, since expensive operations such as the matrix inverse may be computed in parallel with other operations and pipelined in order to limit the amount of blocking tasks. However, many of these operations require significant resources in terms of hardware which might limit some parallel applications. Further testing is required in order to conclude whether or not the extra design complexity compared to high-level programming is worth the effort.

7.1 Future Work

Due to the size of this project, we did not have time to implement every aspect of our proposed design. Therefore, some parts of our described goals in Chapter 1.4 have been left unexplored and left as future work. This chapter describes what is left to be done, as well as possible improvements to our design we discovered towards the end of the project.

7.1.1 NN improvements

The datasets for the NN were generated by sweeping over multiple range bins and injecting target for each CUT. Since there was no filter to remove clutter from the data, we chose to only sweeping over range bins after range bin 500. This allowed us to disregard the clutter located in the first 500 bins, which enabled minimal clutter in the datasets. It would be interesting to implement some postprocessing Doppler filtering to handle clutter, and use all range bins to generate the datasets. This would be needed in order to evaluate the detector's performance when targets are hidden within clutter.

The datasets lacked diversity in terms of weather conditions, environmental settings, or other variations since they were generated at one location during a short time span. They were also comparably small in order to limit the time required for training. Future work could include generating a huge dataset gathered from different location and conditions to get a more comprehensive training set for the model. Furthermore, it would be interesting to generate datasets with known real targets instead of synthetically injecting targets as we did.

An alternative approach for the NN design could also be to incorporate additional input features in the network other than β and $\tilde{t}$, such as metadata describing the weather conditions or environmental context of the collected data, which could allow the network to detect more complex patterns. This may contribute to training a more robust detector capable of generalizing across a wider range of real-world scenarios.

Since the NN detector proposed in this work realizes a robust detector, a more fair evaluation of performance would be to compare it would other well-known robust detectors, such as the AMF or the adaptive ED, instead of the Kelly detector which is selective. Alternatively, the model could be trained in separating the matched H_1 cluster from the mismatched H_1 cluster and H_0 instead, which would produce a selective detector.

7.1.2 Generate CFAR-FP in hardware

Some operations required to perform the CFAR-FP mapping chain were never implemented in VHDL. Firstly, any further devolpment must start with redefining the data format of the modules to avoid the issue of overflow described in Section 6.2. Considering the wide dynamic range of the radar data and the operations performed when computing the matrix inverse, some form of floating point representation might be necessary after all.

The remaining operations that then need to be implemented are the matrix-vector multiplication and the complex absolute value. Since matrix-vector multiplication results in a column vector where each row is the dot product of row i of the matrix and the multiplying vector, it should be possible to implement this seamlessly using our designed dot product module and a simple FSM. As for the complex absolute value, the squared-root operation is needed which may be implemented using for example the coordinate rotation digital computer (CORDIC) algorithm [39].

Once these remaining modules have been implemented, the entire CFAR-FP mapping chain needs to be implemented by interconnecting these modules. In order to assess its performance, future research could include generating clusters in the β-$\tilde{t}$ plane within the FPGA given a radar dataset and compare it to the clusters produced by our MATLAB model. Measuring the computation time of the MATLAB model and hardware implementation respectively would also be of interest in order to determine the trade-off in accuracy versus speed. However, for a fair comparison of speed, a software implementation, such as an embedded C application on a processor, is warranted.

7.1.3 Hardware deployment

Due to limited remaining time and difficulties establishing a stable connection to the hardware board for testing the compiled NN model, deployment of the NN model to the VCK190 board could not be completed. Future work could focus on resolving the connectivity issues, followed by implementing and evaluating the model directly on the hardware platform.

Finally, the deployed NN needs to be connected with the CFAR-FP mapping chain on the VCK190 to implement our entire proposed detector system. Once communication between a host computer and the board has been established, streaming radar data through the system could then be explored in order to examine the system's real-time capabilities.

7.2 Ethical and Ecological Aspects

Since Saab is at its core a defense company and this project will contribute to their technology, it is important that we as engineers consider the ethical and societal aspects of technology used for combat and defense, as well as the environmental aspects of producing and utilizing said technology.

Saab Surveillance specializes in defense systems for detecting incoming threats. The priority of these systems are to warn people of for example a potential missile strike so that civilians and military personnel can seek shelter in time; the main goal is to save lives. These systems could also act as a deterrence for possible aggressors, which might reduce the risk of future conflicts breaking out.

As for the ecological aspect, our design will be implemented on an FPGA so further improvements can be implemented on existing units without the need of manufacturing new ASICs after each update. Implementing the design on a large scale might involve numerous FPGAs. The manufacturing process of these boards has a negative impact on the environment, both from the extraction of the raw materials needed and from the pollution related to industrial production processes. However, Saab is committed to the UN goal of net-zero carbon emissions by 2050 and steps are being taken to limit pollution related to manufacturing [40].

Lastly, it is worth noting that Saab strives to follow the UN Sustainability Development Goals [41], prioritizing nine of these goals: peace, justice, and strong

institutions, climate action, and responsible consumption and production to name a few [42].

Bibliography

[1] A. Coluccia, A. Fascista, and G. Ricci, "CFAR Feature Plane: A Novel Framework for the Analysis and Design of Radar Detectors," *IEEE Transactions on Signal Processing*, vol. 68, pp. 3903–3916, 2020, DOI: https://doi.org/10.1109/TSP.2020.3000952.

[2] AMD. (2023) Versal AI Core Series: Adaptive SoCs and FPGAs. Accessed: 2024-12-19. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal/ai-core-series.html#product-table

[3] Z. Liu, Z. Zhang, Y. Zhang, and Z. Li, "An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications," *Sensors*, vol. 21, no. 13, 2021, DOI: https://doi.org/10.3390/s21134412.

[4] X. Zhao, Y. Zhang, L. Song, M. Xu, and Y. Zhang, Eds., *Edge AI: Convergence of Edge Computing and Artificial Intelligence*. Springer, 2020, DOI: https://doi.org/10.1007/978-981-15-6186-3.

[5] S. Kalapothas, G. Flamis, and P. Kitsos, "Efficient Edge-AI Application Deployment for FPGAs," *Information*, vol. 13, p. 279, May 2022, DOI: https://doi.org/10.3390/info13060279.

[6] A. Coluccia, *Adaptive Radar Detection: Model-based, Data-driven, and Hybrid Approaches*. 685 Canton Street, Norwood, MA, USA: Artech House, 2023.

[7] S. Moussa, A. M. Abdel Razik, A. O. Dahmane, and H. Hamam, "FPGA implementation of floating-point complex matrix inversion based on GAUSS-JORDAN elimination," in *2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2013, pp. 1–4, DOI: https://doi.org/10.1109/CCECE.2013.6567785.

[8] R. L. Castro, B. F. Abreu, J. J. P. C. Rodrigues, P. Lima, M. Silva, J. Ferreira, and A. Pinto, "Fight Fire with Fire: Detecting Forest Fires with Embedded Machine Learning Models Dealing with Audio and Images on Low Power IoT Devices," *Sensors*, vol. 23, no. 2, 2023, DOI: https://doi.org/10.3390/s23020783.

[9] J. Doe, J. Smith, and C. Lee, "Sea Mine Detection Framework Using YOLO, SSD, and EfficientDet Deep Learning Models," *Sensors*, vol. 22, no. 23, 2022, DOI: https://doi.org/10.3390/s22239536.

[10] G. Holst and C. Krull, "Classification of Radar Targets Using Neural Networks on Systems-on-Chip," Master's thesis, Chalmers University of Technology and University of Gothenburg, 2021, accessed: 2024-12-19. [Online]. Available: https://hdl.handle.net/20.500.12380/303775

[11] AMD. (2025) AMD Versal™ AI Core Series VCK190 Evaluation Kit. Accessed: 2025-16-04. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/vck190.html

[12] J. L. Eaves and E. K. Reedy, *Principles of Modern Radar*, 1st ed. New York, NY, USA: Springer, 1987.

[13] R. J. Sullivan, *Radar Foundations for Imaging and Advanced Concepts*. Rayleigh, NC, USA: Scitech Publishing Inc., 2004.

[14] J. Ward, "Space-time adaptive processing for airborne radar," in *1995 International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 5, Detroit, MI, USA, 1995, pp. 2809–2812, DOI: https://doi.org/10.1109/ICASSP.1995.479429.

[15] M. A. Richards, "Introduction to Beamforming and Space-Time Adaptive Processing," in *Fundamentals of Radar Signal Processing*, 2nd ed. New York: McGraw-Hill Education, 2014, ch. 9.

[16] ——, "Detection Fundamentals," in *Fundamentals of Radar Signal Processing*, 2nd ed. New York: McGraw-Hill Education, 2014, ch. 6.

[17] M. I. Skolnik, "Automatic Detection, Tracking, and Sensor Integration," in *Radar Handbook*, 3rd ed. New York: McGraw-Hill, 2008, ch. 7.

[18] I. S. Reed, J. D. Mallett, and L. E. Brennan, "Rapid Convergence Rate in Adaptive Arrays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-10, no. 6, pp. 853–863, 1974, DOI: https://doi.org/10.1109/TAES.1974.307893.

[19] E. J. Kelly, "An Adaptive Detection Algorithm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 22, no. 2, pp. 115–127, 1986, DOI: https://doi.org/10.1109/TAES.1986.310745.

[20] F. C. Robey, D. R. Fuhrmann, E. J. Kelly, and R. Nitzberg, "A CFAR Adaptive Matched Filter Detector," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 28, no. 3, pp. 208–218, 1992, accessed: 2025-01-29.

[21] A. Farina, F. Gini, and M. Greco, "DOA estimation by exploiting the amplitude modulation induced by antenna scanning," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 38, no. 4, pp. 1276–1286, 2002, DOI: https://doi.org/10.1109/TAES.2002.1145749.

[22] S. Bose and A. Steinhardt, "A maximal invariant framework for adaptive detection with structured and unstructured covariance matrices," *IEEE Transactions on Signal Processing*, vol. 43, no. 9, pp. 2164–2175, 1995, DOI: https://doi.org/10.1109/78.414779.

[23] T. Jo, *Machine Learning Foundations: Supervised, Unsupervised, and Advanced Learning*, 1st ed. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer, 2021, DOI: https://doi.org/10.1007/978-3-030-65900-4.

[24] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Pearson Education, 2016.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org

[26] S. L. Brunton and J. N. Kutz, *Data-driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge: Cambridge University Press, 2019.

[27] A. Larner, *The 2x2 Matrix: Contingency, Confusion and the Metrics of Binary Classification*. Gewerbestrasse 11, 6330 Cham, Switzerland: Springer, 2021, DOI: https://doi.org/10.1007/978-3-030-74920-0.

[28] Digilent Inc., *Nexys A7 Reference Manual*, Digilent, 2022, accessed: 2025-05-21. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

[29] AMD, "VCK190 Evaluation Board User Guide," 2024, accessed: 2025-04-08. [Online]. Available: https://docs.amd.com/r/en-US/ug1366-vck190-eval-bd

[30] ——, "VCK190 Evaluation Board Overview," 2024, accessed: 2025-04-08. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/vck190.html

[31] ——, "VCK190 Evaluation Board: Board Component Descriptions," 2024, accessed: 2025-04-08. [Online]. Available: https://docs.amd.com/r/en-US/ug1366-vck190-eval-bd/Board-Component-Descriptions?tocId=XGeSW2sF0hZ9D2I9fGYfYg

[32] Xilinx Inc. (2019) Xilinx First 7nm Device: Versal AI Core (VC1902). PDF slides. Accessed: 2025-04-17. [Online]. Available: https://ieeexplore.ieee.org/document/8875639

[33] Xilinx. (2023) Vitis AI Quick Start Guide for VCK190. Accessed: 2025-04-17. [Online]. Available: https://xilinx.github.io/Vitis-AI/3.0/html/docs/quickstart/vck190.html

[34] W. Dally, C. Harting, and T. Aamodt, "Fixed- and floating point numbers," in *Digital Design Using VHDL: A Systems Approach*, 1st ed. Cambridge, U.K.: Cambridge Univ. Press, 2016, ch. 11.

[35] P. Larsson-Edefors and E. Börjeson, "Implementation evaluation of fixed-point multipliers for complex numbers," in *IEEE 32nd Symposium on Computer Arithmetic (ARITH)*, 2025, pp. 81–84, DOI: https://doi.org/10.1109/CCECE.2013.6567785.

[36] AMD Inc., *Multiplier v12.0 LogiCORE IP Product Guide (PG108)*, AMD, 2015, accessed: 2025-06-23. [Online]. Available: https://docs.amd.com/v/u/en-US/pg108-mult-gen

[37] ——, *Divider Generator v5.1 LogiCORE IP Product Guide (DS530)*, AMD, 2021, accessed: 2025-06-23. [Online]. Available: https://docs.amd.com/v/u/en-US/pg151-div-gen

[38] ——, *Clocking Wizard v6.0 LogiCORE IP Product Guide (PG065)*, AMD, 2011, accessed: 2025-06-23. [Online]. Available: https://docs.amd.com/r/en-US/pg065-clk-wiz/Clocking-Wizard-v6.0-LogiCORE-IP-Product-Guide

[39] ——, *CORDIC v6.0 LogiCORE IP Product Guide (PG105)*, AMD, 2021, accessed: 2025-06-23. [Online]. Available: https://docs.amd.com/v/u/en-US/pg105-cordic

[40] Saab AB. (2022) Saab the First Major Defence Company to Have Science-Based Emission Reduction Targets Approved. Accessed: 2025-05-15. [Online]. Available: https://www.saab.com/newsroom/press-releases/2022/saab-the-first-major-defence-company-to-have-science-based-emission-reduction-targets-approved.

[41] United Nations. (2015) Sustainable Development Goals. Accessed: 2025-01-22. [Online]. Available: https://sdgs.un.org/goals

[42] Saab AB. (n.d.) Our Approach to Sustainability. Accessed: 2025-01-22. [Online]. Available: https://www.saab.com/sustainability/our-approach

A

Gantt Chart

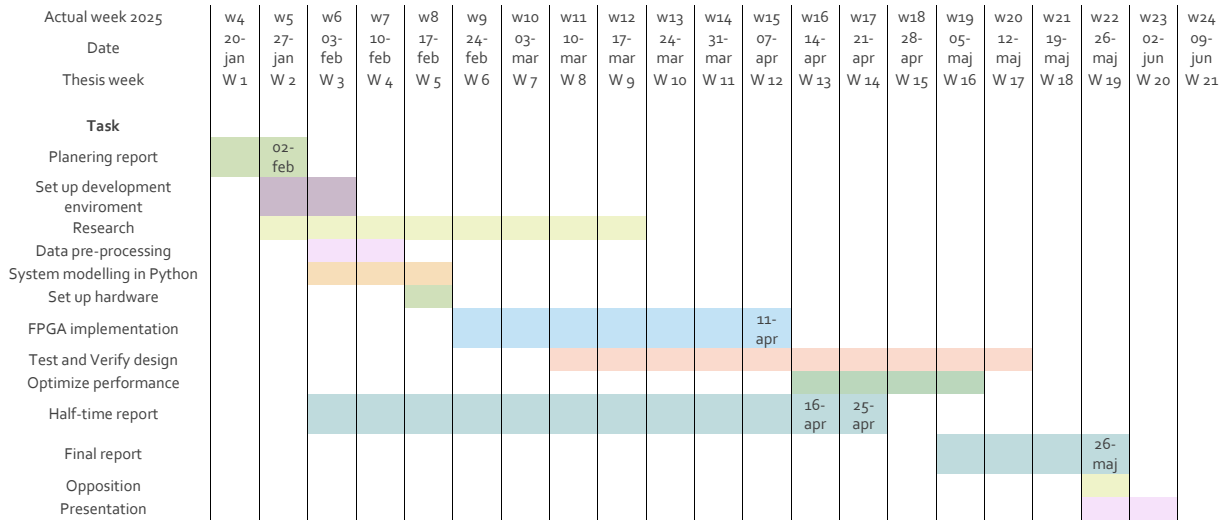


Figure A.1: Gantt chart of the project planning.