# FPGA Acceleration of Automated Ship Detection and CNN-based Sip/Iceberg Discriminator in SAR Imagery

by

Ehsan Mahoor

M.A.Sc., K.N. Toosi University of Technology, 2014
B.Sc., K.N. Toosi University of Technology, 2012

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Sciences

© Ehsan Mahoor 2023
SIMON FRASER UNIVERSITY
Spring 2023

Copyright in this work is held by the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.

# Declaration of Committee

| | |
|---|---|
| **Name:** | Ehsan Mahoor |
| **Degree:** | **Master of Applied Science** |
| **Thesis title:** | **FPGA Acceleration of Automated Ship Detection and CNN-based Sip/Iceberg Discriminator in SAR Imagery** |
| **Committee:** | **Chair:** Michael Adachi<br>Assistant Professor, Engineering Science |
| | **Jie Liang**<br>Supervisor<br>Professor, Engineering Science |
| | **Zhenman Fang**<br>Committee Member<br>Assistant Professor, Engineering Science |
| | **Jiangchuan Liu**<br>Examiner<br>Professor, Computing Science |

# Abstract

The need for automated ship detection methods has become increasingly important with the advancements in Synthetic Aperture Radar (SAR) technology in Maritime Domain Awareness in Canada. In this thesis, we present an automated ship detection algorithm for SAR imagery based on a Trimodal Discrete Model and Nelder-Mead Simplex Algorithm. We explain the theoretical foundation of the algorithm and its optimization techniques to improve its performance. Furthermore, we present the FPGA implementation of this system, which improves its speed and efficiency.

Since ships and icebergs can appear similar in SAR images, we design and train a Convolutional Neural Network (CNN)-based classifier to discriminate between these two objects. To make the CNN model suitable for deployment on small devices, we apply network quantization to shrink its size. Our results demonstrate that the quantized model with 8-bit weights and activation functions has the same accuracy as the floating point one.

Overall, this thesis provides a comprehensive solution for automated ship detection in SAR imagery, including a novel statistical model, FPGA implementation, and deep learning-based classification. Our approach improves the accuracy and efficiency of ship detection and classification in SAR imagery, which has practical implications for maritime surveillance and safety.

**Keywords:** SAR; Deep Learning; Automated Ship Detection; Quantization; FPGA.

# Acknowledgements

I would like to express my gratitude to everyone who has supported me throughout my thesis. I am thankful for their guidance, constructive criticism, and advice during my project work.

I would like to give a special thanks to my supervisor, Dr. Jie Liang, for his invaluable guidance and support throughout my master's program. His expertise and patience have been indispensable to me and have played a crucial role in the success of my thesis. He has always encouraged a spirit of adventure in research and scholarship, challenging me to think outside the box and to pursue innovative approaches to my research questions. His mentorship has been an invaluable asset to my academic growth.

I would also like to thank Dr. Zhenman Fang for his useful comments and remarks during my master's program and defence session. He was the first to enlighten me about my research and provided valuable insights that helped me to refine my research questions and to approach my project from new perspectives.

Furthermore, I am grateful to Dr. Michael Adachi, the chair of my master's defence session, and Dr. Jiangchuan Liu, my examiner, for their time and advice in completing my thesis.

I extend my gratitude to Vantek Innovation Ltd for providing me with the opportunity to conduct my research and for all the resources and support they provided. I would like to give a special thanks to Joe Steyn, who went above and beyond to help me with my work.

I would also like to thank my friends who have supported me in striving towards my goal. In particular, I appreciate Nahid for her love, help, time, and engagement.

Finally, I would like to express my deepest thanks and appreciation to my family, especially my mother, for all of the sacrifices they have made on my behalf. Their unwavering support and encouragement have been a constant source of strength and inspiration throughout my academic journey, and I am deeply grateful for their love and support.

# Table of Contents

**Declaration of Committee** ii

**Abstract** iii

**Acknowledgements** iv

**Table of Contents** v

**List of Tables** vii

**List of Figures** viii

**Acronyms** x

**1 Introduction** 1
- 1.1 SAR in Remote Sensing . . . . . 3
  - 1.1.1 Discovery of Synthetic Aperture Radar . . . . . 3
- 1.2 Application of SAR . . . . . 4
  - 1.2.1 Thesis Organization . . . . . 5

**2 Detection Algorithm In SAR** 6
- 2.1 Maximum Likelihood Quotient Test . . . . . 7
- 2.2 K-Distribution based models for Sea Clutter . . . . . 9
  - 2.2.1 K-Distribution . . . . . 9
  - 2.2.2 Improving K-Distribution . . . . . 9
- 2.3 Trimodal Discrete Texture Model . . . . . 11
  - 2.3.1 Central Moments . . . . . 13
  - 2.3.2 Probability of False Alarm . . . . . 13
  - 2.3.3 Parameter Estimation . . . . . 13
- 2.4 Nelder-Mead Simplex Method . . . . . 15
  - 2.4.1 Nelder–Mead algorithm . . . . . 16

**3 FPGA Implementation of Automated Ship Detection Algorithm** 20
- 3.1 FPGA Resources . . . . . 21

3.2 Input and output . . . . . . . . . . 22
3.3 ASD Architecture . . . . . . . . . . 22
3.4 FPGA Implementation . . . . . . . . . . 24
3.4.1 Magnitude Detection . . . . . . . . . . 24
3.4.2 Window Extraction . . . . . . . . . . 25
3.4.3 ASD Engine . . . . . . . . . . 28
3.4.4 AE-K-FSM . . . . . . . . . . 29
3.4.5 AE-K-STORE . . . . . . . . . . 29
3.4.6 AE-K-STATS . . . . . . . . . . 29
3.4.7 AE-K-MOMS . . . . . . . . . . 31
3.4.8 AE-K-3MD . . . . . . . . . . 33
3.4.9 AE-K-THRSH . . . . . . . . . . 37
3.4.10 AE-K-PIXDET . . . . . . . . . . 38
3.4.11 AE-K-SPAT . . . . . . . . . . 38
3.4.12 Velocity Measurement . . . . . . . . . . 39
3.5 Evaluating Hardware Results . . . . . . . . . . 42
3.5.1 Experimental Platforms . . . . . . . . . . 42
3.5.2 Experimental Results . . . . . . . . . . 42
3.5.3 Resource Utilization . . . . . . . . . . 42

**4 CNN-based Ship/Iceberg Discriminator 45**
4.1 Designing CNN-based Discriminator . . . . . . . . . . 46
4.1.1 Dataset . . . . . . . . . . 46
4.1.2 CNN Model Architecture . . . . . . . . . . 48
4.1.3 Experimental Results . . . . . . . . . . 49
4.2 FPGA Implementation of the CNN Model . . . . . . . . . . 50
4.2.1 CNN Quantization . . . . . . . . . . 51

**5 Conclusion and Future Work 54**
5.1 Future Work . . . . . . . . . . 54

**Bibliography 56**

# List of Tables

Table 3.1 Number of bits allocated to the sum of each power . . . . . . . . . . . 31
Table 3.2 Calculated moments in software and hardware. . . . . . . . . . . . . . . 33
Table 3.3 Initial values of the parameters for Nelder-Mead algorithm . . . . . . 34
Table 3.4 Gamma values and their representation as addition/subtraction of power of 2 integers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
Table 3.5 Software and hardware results of the AE-K-3MD unit. Parameters are randomly initialized and are the same for both software and hardware. 37
Table 3.6 Ship report comparison among software and FPGA desgin. . . . . . . 43
Table 3.7 Resource utilization, latency, and throughput of the hardware design. 44
Table 3.8 The performance comparison among CPU and FPGA. . . . . . . . . . 44

Table 4.1 Performance of the designed model over the train, validation, and test datasets. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
Table 4.2 Accuracy of the quantized model with different quantization bit-width over validation data set. W/A represents bit-widths of weights (W) and activations (A). . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
Table 4.3 DPU Resource usage for quantized CNN ship/Iceberg discriminator on ZCU104 device. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53

# List of Figures


Figure 1.1 Illustration of the Seasat-A SAR satellite. Figure from [1] . . . . . . 2

Figure 2.1 Comparison of the fit between the measured and theoretical cdf in logarithmic scale [2]. . . . . . . . . . . . . . . . . . . . . . . . . . . 10

Figure 2.2 RADARSAT-2 SAR image of the Strait of Georgia, BC, Canada [2]. 11

Figure 2.3 Perfect fit between empirical moments and theoretical values given in equation (14) for the heterogeneous case (red plus/blue curve) and homogeneous case (cyan curve/green plus). . . . . . . . . . . . . . . . 15

Figure 2.4 Nelder–Mead simplices after a reflection and an expansion step. The original simplex is shown with a dashed line [3]. . . . . . . . . . . . 18

Figure 2.5 Nelder–Mead simplices after an outside contraction, an inside contraction, and a shrink. The original simplex is shown with a dashed line [3]. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

Figure 3.1 Automated Ship Detection Unit Architecture . . . . . . . . . . . . 23

Figure 3.2 Applying magnitude detection on a complex image. (a) real part. (b) imaginary part. (c) magnitude. . . . . . . . . . . . . . . . . . . . . . 25

Figure 3.3 The concept of dividing a magnitude detected image into multiple windows. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

Figure 3.4 The state machine for dividing the magnitude detected image into windows. (a) Transferring the first half and storing the second half of the image. (b) Reading the second half windows. . . . . . . . . . 27

Figure 3.5 Finite State Machine Transition Diagram for the AE-K-FSM Module 30

Figure 3.6 Storing the SLC image in hardware . . . . . . . . . . . . . . . . . . 40

Figure 3.7 XY coordinates of the target center in SCL image . . . . . . . . . . 40

Figure 3.8 Memory addresses for image chip extraction . . . . . . . . . . . . . 41

Figure 3.9 An input window to the ASD engine for ship detection containing 4 targets. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

Figure 4.1 Sample images from Kaggle competition [4]. Top row: Ship. Bottom row: Iceberg . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

Figure 4.2 Architecture of the simple CNN model. . . . . . . . . . . . . . . . . 48

Figure 4.3 Architecture of the ResNet-based CNN model. . . . . . . . . . . . . 49

Figure 4.4 A sample image and its augmented data. . . . . . . . . . . . . . . . . . 51

## Acronyms

**API** Application Programming Interface

**ATI** long-track interferometric

**cdf** cumulative density function

**DPU** Deep-learning Processor Unit

**EM** Electromagnetic

**FAR** False Alarm Rate

**FPGA** Field Programmable Gate Array

**FSM** Finiste State Machine

**GPU** Graphics Processing Unit

**HLS** Xilinx Vitis High-Level Synthesis

**KPG** K-plus-Gauss

**KPN** K-plus-noise

**KPR** K-plus-Rayleigh

**MDA** Maritime Domain Awareness

**MLS** minimum least square

**PPN** Pareto-Plus-Noise

**ResNet** Residual Network

**ROM** Read Only Memory

**SAR** Synthetic Aperture Radar

**SCR** Signal-to-Clutter Ratio

**SDK** Software Development Kit

**SLC** Single Look Complex

**SNR** Signal-to-Noise Ratio

**SOAP** SAR On-board Automated Processor

**SB** space-based

**VHDL** Very High Speed Integrated Circuit Hardware Description Language

**3MD** Trimodal Discrete Sea Clutter

# Chapter 1

# Introduction

A Synthetic Aperture Radar, commonly known as SAR, is a radar system that employs microwave energy to generate high resolution images of targets, regardless of the lighting conditions or visibility [5]. SAR works by transmitting microwave pulses and measuring the time it takes for the energy to reflect back to the antenna. The obtained data is then processed to produce an image, enabling the detection and imaging of objects that are otherwise concealed or obscured.

Prior to the development of imaging radar, most high resolution sensors relied on camera systems with detectors sensitive to reflected solar radiation or thermal radiation emitted from the earth's surface. SAR, however, presented a revolutionary approach to earth observation [6]. As an active system that transmits a beam of Electromagnetic (EM) radiation in the microwave region of the spectrum, SAR extends our ability to observe surface properties that were previously undetectable. Unlike passive sensors, SAR provides its own illumination, making it operational during day and night. Moreover, microwaves are unaffected by clouds, fog, or precipitation, enabling all-weather imaging. This makes SAR a versatile instrument for continuously observing dynamic phenomena like ocean currents, sea ice motion, and changing vegetation patterns [7].

Sensor systems operate by intercepting earth radiation using an aperture of some physical dimension. In traditional non-SAR systems, the angular resolution is determined by the ratio of the wavelength of the EM radiation to the aperture size. The spatial resolution of the image is the angular resolution multiplied by the sensor distance from the earth's surface. Therefore, the spatial resolution of the image decreases as the sensor altitude increases unless the physical size of the aperture is increased [1]. In visible and near-infrared wavelengths, high resolution imagery can be obtained at spaceborne altitudes with modest aperture sizes. However, for microwave instruments, where the wavelengths are typically $100,000$ times longer than light, high resolution imagery with a reasonably sized antenna aperture is not feasible. For example, consider the Seasat SAR, which has a 10 m antenna aperture and orbits at an altitude of 800 km (Fig. 1.1). At the radar wavelength of 24 cm,

the real aperture resolution is approximately 20 km. To obtain a 25 m resolution image, an antenna over 8 km long would be required [1].

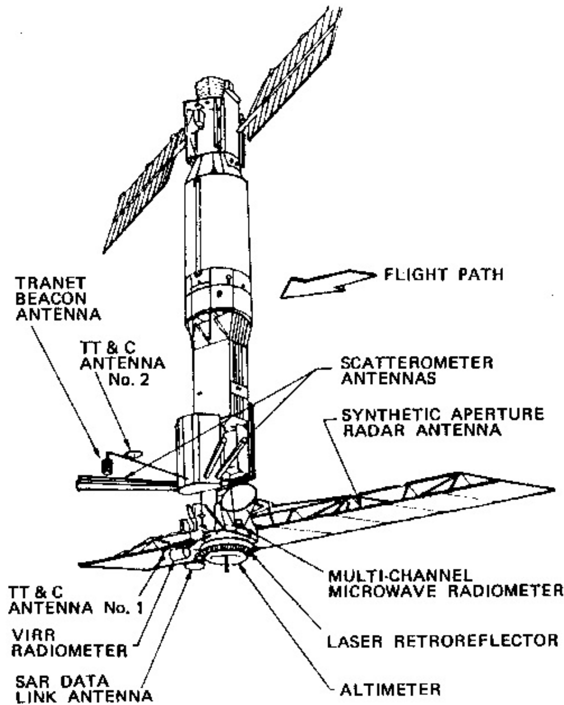



Fig. 1.1. Illustration of the Seasat-A SAR satellite. Figure from [1]

To achieve high resolution without increasing the physical antenna size, SAR technology is utilized [1]. A SAR system retains both phase and magnitude of the backscattered echo signal, enabling the synthesis of an extremely long antenna aperture in the signal processor [1]. The synthetic array is created digitally in a ground computer by compensating for the quadratic phase characteristic associated with the long synthetic aperture's near field imaging. Consequently, the SAR system can achieve a resolution independent of the sensor altitude, making it an invaluable instrument for space observation [8].

However, any remote sensor designed for global coverage at high resolution generates a substantial amount of data. The SAR is no exception, as hundreds of mathematical operations must be performed on each data sample to form an image from the downlinked signal data. For instance, processing a 15-second Seasat image frame consisting of several hundred million data samples into imagery in real-time requires a computer system capable of performing several billion floating point operations per second. As a result, much of the early processing of SAR data was performed optically using laser light sources, Fourier optics, and film. The early digital correlators could only process a small portion of the acquired data and generally approximated the exact matched filter image formation algorithms due to limited computer hardware capabilities. Consequently, the signal processors produced an

image product of degraded quality with a limited dynamic range that could not be reliably calibrated. These limitations of the optically processed imagery and the limited quantity and performance of the digital products served to constrain progress in the scientific application of SAR data during its formative years [1].

## 1.1 SAR in Remote Sensing

SAR is unique in remote sensing due to its ability to image day or night, in any weather condition, and its geometric resolution is not affected by sensor altitude or wavelength. Additionally, the signal data of SAR is specific to the microwave region of the electromagnetic spectrum.

SAR belongs to a class of instruments that include radars, altimeters, scatterometers, and lasers, which transmit a signal and measure the reflected wave. Unlike passive sensors like cameras and radiometers, SAR does not rely on external radiation sources like solar or nuclear radiation. The radar frequency can be selected to minimize its attenuation by atmospheric molecules, making it possible for SAR to image the Earth's surface regardless of the presence of clouds or precipitation.

### 1.1.1 Discovery of Synthetic Aperture Radar

The concept of synthetic aperture radar was first proposed by Carl Wiley of the Stanford Research Institute in the early 1950s [1]. Wiley recognized that a stationary radar antenna could effectively synthesize a larger antenna by processing the received signals over a period of time [1]. This approach would provide the benefits of high resolution and long-range detection in a compact system. Wiley's idea was based on the principle of aperture synthesis, which had already been used in radio astronomy to create high resolution images of celestial objects.

The early SAR systems were airborne and used a side-looking configuration. The radar antenna was mounted on the side of an aircraft and the radar beam was directed at the ground at an oblique angle. The aircraft flew along a straight line, while the radar antenna transmitted pulses of microwave energy towards the ground. The echoes reflected back from the ground were received by the antenna and processed to form an image of the ground surface. The image was built up by combining the echoes received from multiple pulses as the aircraft flew over the target area. The system used the Doppler shift of the echoes to determine the ground range and the side-looking geometry to determine the azimuth [1].

During the early 1950s, engineers discovered that instead of rotating the antenna to scan the target area, it could be fixed to the fuselage of the aircraft, which resulted in longer apertures and improved along-track resolution. Additionally, film was used to record the CRT display of the pulse echoes, and these early versions of side-looking aperture radar (SLAR) systems were primarily employed for military reconnaissance purposes. It

wasn't until the mid 1960s that the first high resolution SLAR images were declassified for scientific use, and their value for applications like geologic mapping, oceanography, and land use studies was recognized immediately [1].

As technology continued to advance, the SLAR systems evolved into SAR systems, which are now widely used for remote sensing applications. Unlike SLAR, which used a fixed antenna and relied on the aircraft's motion to create a synthetic aperture, SAR uses a moving antenna and precise control over the radar signal to synthesize an aperture that is much longer than the physical antenna [1].

SAR systems can operate at a variety of frequencies and polarizations, depending on the specific application. For example, L-band (1-2 GHz) and C-band (4-8 GHz) SAR systems are commonly used for Earth observation and disaster monitoring, while X-band (8-12 GHz) and Ku-band (12-18 GHz) systems are used for military surveillance and target detection [9].

## 1.2 Application of SAR

SAR technology is commonly used in various applications such as military surveillance, earth observation, and environmental monitoring. SAR is widely used in maritime surveillance for a variety of purposes, including detecting and monitoring vessels, monitoring ocean currents, and identifying potential hazards such as icebergs or oil spills. Here are some key ways SAR is used in maritime surveillance [1]:

**Vessel Detection:** SAR can detect vessels even in adverse weather conditions such as fog, rain, and darkness. By analyzing the radar signals reflected by a vessel, SAR can determine the size, speed, and direction of the vessel, as well as its heading and other details. This information is useful for maritime security, search and rescue operations, and monitoring of illegal fishing or other activities.

**Ocean Current Monitoring:** SAR can also be used to monitor ocean currents, which can be important for shipping, environmental monitoring, and search and rescue operations. By analyzing the radar signals reflected by the ocean surface, SAR can detect variations in sea surface roughness caused by ocean currents and provide information about their direction and strength.

**Hazard Detection:** SAR can also detect potential hazards in the ocean such as icebergs, oil spills, and other objects that may pose a threat to navigation or the environment. By analyzing the radar signals reflected by these objects, SAR can provide information about their location, size, and other characteristics, which can help to alert authorities and guide response efforts.

Overall, SAR technology is an important tool for maritime surveillance, providing valuable information to help keep ships and crews safe, protect the environment, and ensure the security of our oceans. In this thesis, our focus is on the automated ship detection and

ship/iceberg discrimination, and their real-time implementation. Real-time ship detection in SAR is critical because it allows authorities to quickly respond to potential threats or emergencies. In contrast to traditional ship detection methods that rely on visual or manual searches, SAR can provide up-to-date information about vessel activity in near-real-time. This is particularly important in situations where time is of the essence, such as in maritime security or search and rescue operations. Real-time ship detection in SAR is made possible by advances in SAR technology and data processing. SAR systems can now acquire and process data much more quickly than in the past, allowing for near-real-time detection and tracking of vessels. In addition, advances in machine learning and artificial intelligence have enabled the development of Ship/iceberg discrimination algorithms that can quickly analyze SAR data and classify vessels and icebergs with a high degree of accuracy.

Achieving real-time implementation of ship detection in SAR requires specialized hardware because SAR data processing is computationally intensive and requires a large amount of data bandwidth [10]. The amount of data generated depends on factors such as the size of the area being imaged, the resolution of the image, and the frequency of the radar pulses. For example, a typical SAR system operating in X-band frequency can generate data at rates of up to several hundred megabits per second.

To process this data in real-time, specialized hardware is required that can handle the high-speed data flow and perform the necessary computations quickly and efficiently. This hardware typically includes specialized processors, high-speed data buses, and advanced memory architectures to handle the high-bandwidth data flows and perform the necessary computations. In addition, hardware acceleration techniques such as Graphics Processing Unit (GPU) or Field Programmable Gate Array (FPGA) can be used to offload the processing from the CPU and achieve real-time performance.

Without specialized hardware, processing SAR data in real-time would be very challenging if not impossible. The large amount of data generated by the SAR system requires specialized hardware to handle the high-speed data flow and perform the necessary computations quickly and efficiently. This is why hardware implementation is essential for achieving real-time implementation of ship detection in SAR.

### 1.2.1 Thesis Organization

The remainder of this thesis is divided into four more chapters. Chapter 2 provides a detailed discussion of the automated ship detection algorithm. Chapter 3, provides the details of the FPGA implementation of the ship detection unit and its experimental results. In Chapter 4, the design of the CNN-base ship/iceberg discriminator along with its hardware implementation explained in detail. Finally, Chapter 5 concludes this thesis and suggests potential future work.

# Chapter 2

# Detection Algorithm In SAR

Maritime Domain Awareness (MDA) in Canada, as well as in other coastal countries, has reached an operational stage with the use of space-based SAR for detecting and observing vessels in open oceans [11, 12, 13]. The all-weather, day-and-night capability of SAR, along with its global reach and increasing coverage, has made it the preferred choice for MDA.

With the advancements in SAR technology, SAR sensors can now generate vast amounts of data in a short period, which highlights the need for automated target detection methods. This is particularly crucial in the case of ship surveillance as a majority of the imagery captures the open ocean, where the presence of ships can be challenging to detect. By automating the ship detection process, consistency and predictability can be ensured. Currently, operational ship detection predominantly relies on analyzing single co- or cross-polarized SAR images through a statistical detection step. This step involves the declaration of a ship when a pixel magnitude exceeds a predetermined threshold, which is determined based on a statistical model of the measured SAR magnitude sea background data, commonly known as clutter in radar terms [2].

The term "clutter" is used to refer to any object that may produce undesired radar echoes and potentially disrupt the normal functioning of radar systems . Clutter can be broadly categorized into two types: surface clutter and airborne/volume clutter. Surface clutter encompasses objects like trees, vegetation, man-made structures, ground terrain, and the sea surface (also known as "sea clutter"). In contrast, airborne/volume clutter usually consists of larger objects such as chaff, rain, birds, and insects. While surface clutter can vary significantly depending on the location and terrain, airborne/volume clutter tends to be more predictable in terms of its extent and characteristics. To address the challenges posed by clutter, various techniques are used to suppress or reject unwanted signals and improve the accuracy of radar operations [2].

Clutter echoes are typically random and exhibit characteristics similar to thermal noise due to the random phases and amplitudes of individual clutter components, also known as scatterers. Often, the level of clutter signal can exceed the level of receiver noise. Therefore, detecting targets in a high clutter background depends on the Signal-to-Clutter Ratio (SCR)

rather than the Signal-to-Noise Ratio (SNR). In contrast to white noise, which affects all radar range bins equally, the power of clutter returns can vary within a single range bin. As clutter returns can resemble target echoes, accurately modeling clutter in different areas is crucial for a radar to distinguish target returns from clutter echoes [2].

To improve detection accuracy and reduce False Alarm Rate (FAR), SAR images are divided into rectangular subimages of predetermined size for individual processing. However, these areas cannot be assumed to be homogeneous, especially with higher resolution radars. Therefore, it is critical to accurately model the sea clutter and noise in an adaptive manner to determine the appropriate detection threshold. A new statistical sea clutter model called Trimodal Discrete Sea Clutter (3MD) is introduced in this thesis, which provides more accurate data modeling in challenging environments and thermal-noise limited cases. The 3MD model is notable for its numerical simplicity, allowing for efficient parameter adaptation and enhancing robustness while reducing computational complexity. Before discussing the 3MD model in depth, it is important to understand the most commonly used models that employ a continuous random variable, such as the K-distribution, KK-distribution, and K+N-distribution, for modeling sea clutter [2].

To explain the statistical clutter models, it is necessary to explain the likelihood ratio test evaluates the fitness of two competing statistical models based on the ratio of their likelihoods, one derived by maximizing the likelihood function over the entire parameter space, and another that is constrained in some way. When the constraint, also known as the null hypothesis, is upheld by the observed data, the difference between the two likelihoods should not exceed the sampling error. Therefore, the likelihood-ratio test determines whether the ratio is significantly different from one or if its natural logarithm is significantly different from zero.

## 2.1 Maximum Likelihood Quotient Test

A statistical test is used to determine whether a target $S$ is present in the SAR image by distinguishing between two hypotheses, $(H_0)$ and an alternative hypothesis $(H_1)$.

$$\begin{aligned} &H_0\colon \mathbf{Z} = \Delta.\mathbf{C} + \mathbf{N}, \ \ \text{(interference alone)} \\ &H_1\colon \mathbf{Z} = \alpha\mathbf{S} + \Delta.\mathbf{C} + \mathbf{N}, \ \ \text{(interference plus target)} \end{aligned} \tag{2.1}$$

based on $n$ measurements $(Z_1, \cdots, Z_n)$, concatenated into vector $\mathbf{Z}$, and unknown $\alpha \in \mathbb{C}$ with $|\alpha| \neq 0$. The assumption is that the clutter and noise vectors $\mathbf{C}$ and $\mathbf{N}$ have zero-mean and are complex normal distributed with covariance matrices $\mathbf{R}_C$ and $\mathbf{R}_N \in \mathbb{C}^{n \times n}$, respectively. This assumption is motivated by the central limit theorem, which suggests that the superposition of a large number of independent distributed scatterers in each

resolution cell should result in a normal distribution. The positive scalar texture random variable with unit second moment is denoted by $\Delta$ and is assumed to be independent of the target, clutter, and noise. Under $H_0$, $\mathbf{R} = \mathcal{E}\mathbf{Z}\mathbf{Z}^* = \mathbf{R}_C + \mathbf{R}_N$, where $\mathcal{E}$ denotes the expectation operator. Throughout the thesis, capital letters represent random variables and small letters represent their realizations. Bold letters represent vectors and matrices, matrix transposition is denoted by the superscript $'$, and conjugate complex transpose by $*$.

The optimum test, in the Neyman–Pearson [14] sense, is given by the probability ratio

$$\mathrm{T} = \max_{\alpha} \frac{f_{H_1}(\mathbf{Z}, \alpha)}{f_{H_o}(\mathbf{Z})} = \max_{\alpha} \frac{\int f_{H_1}(\mathbf{Z}|\Delta = \delta, \alpha)\ f_{\Delta}(\delta)\, d\delta}{\int f_{H_0}(\mathbf{Z}|\Delta = \delta)\ f_{\Delta}(\delta)\, d\delta} \lessgtr \eta, \tag{2.2}$$

where a decision of "target" is made when the ratio between the probability density functions exceeds a threshold $\eta$. However, for classical texture models like gamma or inverse gamma distributions, obtaining an analytical derivation of the statistics T or $\log \mathrm{T}$ in equation 2.2 is difficult.

On the other hand, when the measurements Z are Gaussian (i.e., no texture) and the target signal $\mathbf{s}$ is deterministic, the solution of 2.2 after maximization with respect to $\alpha$ becomes

$$\mathrm{T} = \frac{\left|\mathbf{s}^*\mathbf{R}^{-1}\mathbf{Z}\right|^2}{\mathbf{s}^*\mathbf{R}^{-1}\mathbf{s}}. \tag{2.3}$$

Given that the conditional probability density function of $\mathbf{Z}$ for fixed $\delta$ is a complex Gaussian distribution, one can use an approximate solution given by equation 2.4.

$$\mathrm{T}|\Delta = \frac{\left|\mathbf{s}^*\mathbf{R}^{-1}(\mathbf{Z}|\Delta)\right|^2}{\mathbf{s}^*\mathbf{R}^{-1}\mathbf{s}}. \tag{2.4}$$

It is assumed that the target signal $\mathbf{s}$ is only present in the first L cells, represented by the vector $\mathbf{s} = s[\mathbf{1}_L, \mathbf{0}_{n-L}]'$, where $\mathbf{1}_m$ and $\mathbf{0}_m$ are vectors of length $m$ containing all ones or zeros, respectively. Additionally, it is assumed that all cells are mutually statistically independent, resulting in a covariance matrix that is a multiple of the identity matrix $\mathbf{R} = \sigma^2\mathbf{I}_n$ with $\sigma^2 = {\sigma_c}^2 + {\sigma_n}^2$. The test equation 2.4 can then be simplified to the sum of the normalized squared SAR pixel amplitudes given by equation 2.5.

$$\mathrm{T}|\Delta = \frac{1}{L\sigma^2}\sum_{k=1}^{L} |(\mathbf{Z}_k|\Delta)|^2. \tag{2.5}$$

The marginal test probability density function can be computed through integration

$$f_T(t) = \int_0^\infty f_{T|\Delta}(t, \delta)\, f_{\Delta}(\delta)\, d\delta. \tag{2.6}$$

For more information on the statistics of the conditional and marginal tests, refer to [15, 16]. It is worth emphasizing that the test presented in equation 2.5 requires integrating only the L pixels that contain the target and ignoring all other pixels. This is reasonable because adding interference only reduces the signal-to-clutter-plus-noise ratio (SCNR) and makes it more difficult to detect the target. However, in real-world scenarios, the size of the target is not known beforehand and can vary. Therefore, a predetermined number of cells, denoted as n, is often used. Typically, this number is based on the smallest target size of interest.

## 2.2 K-Distribution based models for Sea Clutter

### 2.2.1 K-Distribution

In the field of detection theory, the analysis of the tail of a probability density function (pdf) is crucial, especially when considering the logarithmic form. It is important to note that often in literature, the agreement between a theoretical distribution and the data histogram is evaluated only for the main part of the pdf, which may give an impression of an excellent match. However, for practical requirements, very low false alarm rates (Pfa) in the order of $10^{-9}$ to $10^{-10}$ are often demanded, which can lead to wrong detection thresholds and missed targets. The validity of the K-distribution, for instance, is limited to moderate false alarm rates, as its asymptotic behavior is too weak to reflect empirical data, especially for heterogeneous scenes. Analytically continuing the pdf to regions unrepresented by available data is assumed, but validation of the false alarm rate is still necessary [2].

An example of this is shown in Figure 2.1, which displays the match between the measured cumulative distribution function (cdf) and the K-cdf (red curve) for the SAR data within the white box in Figure 2.2 in logarithmic scale. Figure 2.2 presents a RADARSAT-2 image of a section of the Strait of Georgia off Vancouver captured on November 11, 2008. It is evident that the agreement worsens for larger amplitudes, but the tail is critical for determining the detection threshold. The shape and fit of the K-distribution mainly depend on the value of the texture variable. Without any limitation on computational time, a maximum likelihood estimator with global maximization of the likelihood function using all pixels within the white frame was employed, and the obtained value was $\nu = 4.2642$. This relatively small value confirms the visible heterogeneity of the selected sea surface [2].

### 2.2.2 Improving K-Distribution

To address the limitations of the K-distribution, Watts [17] introduced the K-plus-Gauss (KPG) distribution which incorporates thermal noise. However, calculating the resulting pdf analytically is difficult and numerically computing it is computationally challenging [18]. Recent studies have shown that the KPG does not accurately fit actual SAR data, resulting in the loss of targets [19]. While numerical integration is required for computing the KPG

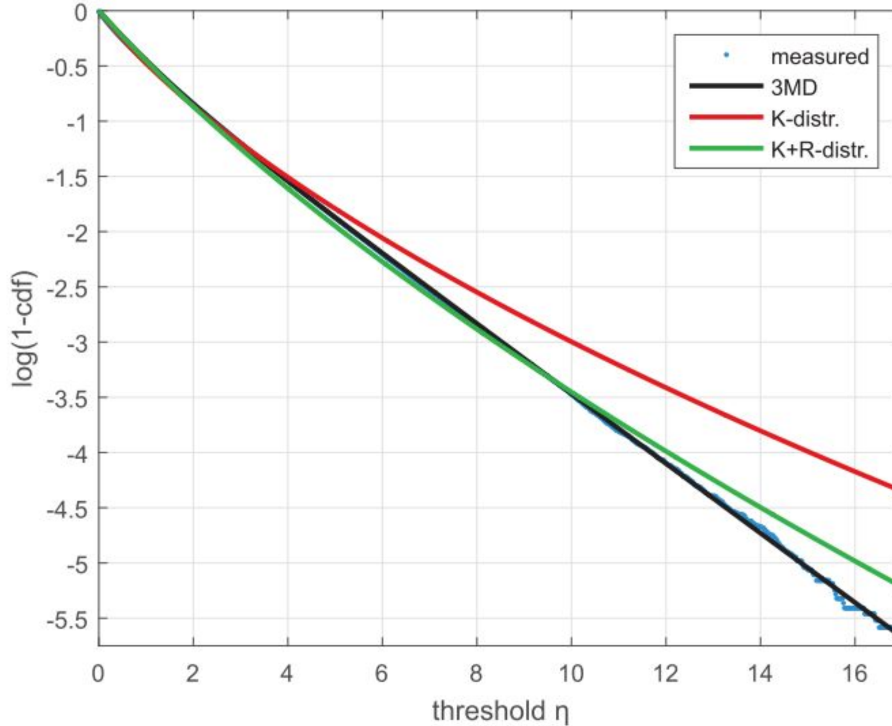



Fig. 2.1. Comparison of the fit between the measured and theoretical cdf in logarithmic scale [2].

pdf, closed-form expressions for whole-number central moments can be used to estimate the unknown texture parameter [20]. An interesting approach to improve KPG parameter estimation have proposed by [21, 22] using the simplex algorithm for multidimensional unconstrained nonlinear minimization. This method also used in this thesis for estimating the parameters of a new model [2] and will be explained in detail.

The K-plus-Rayleigh (KPR) distribution [23] is an extension of the KPG distribution that accounts for an additional statistical white clutter contribution not captured by the assumed thermal noise component. The KPR generally provides better fits to the sea surface clutter pdf than the K-plus-noise (KPN) [24, 25, 26] and Pareto-Plus-Noise (PPN) [27] distributions , and the method of moments (MoM) [28] can be used to estimate the texture parameter. However, numerical integration is required as there is no closed-form expression for the pdf. Fig. 2.1 illustrates the considerable improvement in the tail compared to the classic K-distribution [2].

Another method for improving data fit is the mixture speckle model, which combines multiple clutter pdfs. However, this approach does not add random variables and is dependent on the existence of disjoint types of scatterers with specific individual distributions. The K-K model, which is a combination of two K-pdfs with different parameters, is a proposed candidate for sea clutter models. However, the K-K is more numerically challenging than the single K-distribution, making it computationally intensive and numerically unstable.

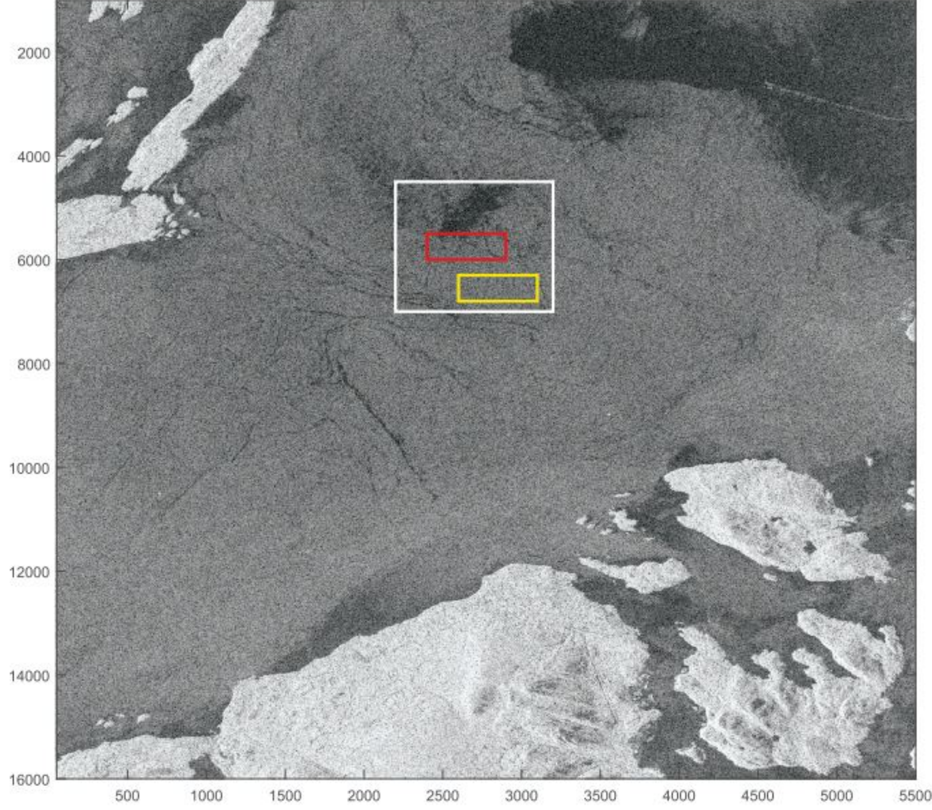



Fig. 2.2. RADARSAT-2 SAR image of the Strait of Georgia, BC, Canada [2].

## 2.3 Trimodal Discrete Texture Model

The existing methods for extending and modifying the compound model all assume that the slowly varying power fluctuations are modeled as a continuous random variable with support between $0 \leq \delta < \infty$, resulting in a small but nonzero probability of infinite texture values. This is not physically feasible and cannot be measured by any radar system. Instead, a discrete texture model can be used with a probability density function represented by equation 2.7:

$$f_{\Delta}(\delta) = \sum_{i=1}^{I} c_i \varsigma(\delta - a_i), \quad \sum_{i=1}^{I} c_i = 1, \; c_i \geq 0, \tag{2.7}$$

here, $\varsigma(.)$ represents the delta function, $\mathbf{a} = [a_1, ...., a_I]$ denotes the discrete texture intensity levels, and $\mathbf{c} = [c_1, ...., c_I]$ represents their corresponding relative weightings [15, 29, 30] . This discrete texture model is physically sound as the sea clutter usually consists of a finite, and often small, number $I$ of distinct types of scatterers [15].

The new model's performance is demonstrated in the smaller areas of the red and yellow boxes in Fig. 2.1, which showcase the fit of the theoretical pdf, the cdf, and the Pfa with their empirical counterparts for both very heterogeneous and rather homogeneous sea clutter. Determining the number I of scatterer classes to consider for a sea clutter patch is not

known in advance. One approach to address this is to estimate the order number explicitly from the same data set. However, analyzing various RADARSAT-2 data sets for different modes, including extremely non-homogeneous surfaces, revealed that $I = 3$ was sufficient in all cases. Similar findings have been reported in [15, 31], suggesting that two or three different classes of sea back-scatter appear to be sufficient for SAR image segmentation purposes. Hence, the technique has been named trimodal discrete (3MD) sea clutter model [32].

### Probability and Cumulative Density Function

After combining the $2I$ unknown parameters into the vector $\mathbf{\Theta} = [\mathbf{c}; \mathbf{a}]$, and using the discrete texture, 2.7 yields the continuous test pdf [32]:

$$f_T(t, n, \Theta) = \frac{n^n}{\Gamma(n)} t^{n-1} \sum_{i=1}^{I} \frac{c_i}{b_i^n} \exp\left(-\frac{nt}{b_i}\right), \tag{2.8}$$

which

$$b_i = \rho_c a_i^2 + \rho_n, \tag{2.9}$$

and

$$\rho_c + \rho_n = \frac{\sigma_c^2}{\sigma_c^2 + \sigma_n^2} + \frac{\sigma_n^2}{\sigma_c^2 + \sigma_n^2} = 1. \tag{2.10}$$

The interpretation of 2.8 is the existence of disjunct scatterer types, meaning that each scatterer in the observed scene is a realization from homogeneous clutter random variables resulting in the sum of three Rayleigh pdfs for the image amplitude. For single-look data, with $n = 1$, the pdf simplifies to a sum of weighted exponentials as the following equation:

$$f_T(t, \Theta) = \sum_{i=1}^{I} \frac{c_i}{b\ i} \exp\left(\frac{-t}{b_i}\right)$$

Through integration of 2.8, the cdf reads in closed form:

$$F_T(t, n, \Theta) = 1 - \sum_{i=1}^{I} \frac{c_i}{\Gamma(n)} \Gamma\left(n, \frac{nt}{b_i}\right). \tag{2.11}$$

which simplifies for single-look data to:

$$F_T(t, \Theta) = 1 - \sum_{i=1}^{I} c_i \exp\left(\frac{-t}{b_i}\right) \tag{2.12}$$

### 2.3.1 Central Moments

Moments are mathematical quantities that help us describe the shape, location, and variability of a probability distribution. In mathematical statistics, we use moments to calculate a distribution's mean, variance, and skewness.

The first moment of a distribution is the mean, which is a measure of the central tendency of the distribution. The second moment of a distribution is the variance, which is a measure of the spread or variability of the distribution. The third moment of a distribution is the skewness, which is a measure of the asymmetry of the distribution. A distribution is said to be skewed if it is not symmetrical around its mean.

Higher order moments, such as the fourth moment, can also be used to describe the shape and characteristics of a distribution. To calculate the central moments, we first need to describe the characteristic function as the expectation

$$\Phi_T(\alpha) = \mathcal{E}_T e^{j\alpha T} = \int_o^\infty e^{j\alpha t} f_T(t) dt, \tag{2.13}$$

which, with pdf 2.8, we can calculate all central moments via

$$\Phi_T(\alpha, \Theta) = n^n \sum_{i=1}^{I} \frac{c_i}{(n - j\alpha b_i)^n} \tag{2.14}$$

Alternatively, the $r$th moment for single-look data, $n = 1$, can be calculated through direct integration of the pdf

$$\mathcal{E}\, T(\Theta)^r = \int_0^\infty t^r f_T(t)\, dt = \Gamma(1 + r) \sum_{i=1}^{I} c_i b_i^r. \tag{2.15}$$

### 2.3.2 Probability of False Alarm

The probability of false alarm $\mathbf{P}_{fa}$ as a function of a varying threshold $\eta$ is given as one minus the cdf

$$\mathbf{P}_{fa}(\eta, \Theta)^r = 1 - F_T(\eta, \Theta) = \sum_{i=1}^{I} \frac{c_i}{\Gamma(n)} \Gamma\left(n, \frac{n\eta}{b_i}\right), \tag{2.16}$$

which for single-look data, $n = 1$, reduces to :

$$\mathbf{P}_{fa}(\eta, \Theta) = \sum_{i=1}^{I} c_i \exp\left(\frac{-\eta}{b_i}\right). \tag{2.17}$$

### 2.3.3 Parameter Estimation

Typically, discrete texture models have a higher number of unknown parameters or degrees of freedom compared to most continuous texture sea surface models. These parameters have

a direct physical meaning, and it is demonstrated in this section that their estimation can be efficiently and robustly conducted due to the straightforward nature of the test statistic. Various methods are available to estimate the unknown parameters, including the option of using a minimum least square (MLS) fit of the model pdf to M histogram bins $h_m$ as described in

$$\arg\min_{\Theta} \sum_{m=1}^{M} \left(f_{T_m}(t_m, n, \Theta) - h_m\right)^2, \tag{2.18}$$

or to maximize the log-likelihood function via

$$\arg\max_{\Theta} \sum_{m=1}^{M} \ln f_{T_m}(t_m, \Theta). \tag{2.19}$$

where the image pixels for $m = 1, ..., M$ are the realizations of mutually statistical independent random variables, all possessing the pdf 2.8.

The method of moments (MoM) is the arguably most applied technique in practice. in which a set of $2I$ equations

$$\mathcal{E}T(\Theta)^r = \hat{t_r} \qquad r = 1, \ldots, 2I, \tag{2.20}$$

has to be solved with respect to the $2I$ unknowns in $\Theta$, where the $\hat{t_r} = (\frac{1}{M}\sum_{m=1}^{M} Z_m^r)$ are the sample moments. By assuming the noise floor $\sigma_n^2$ to be a priori known, the MLS approach can be used to solve an over-determined system of equations and improve the robustness and accuracy of the estimated parameters.

$$\arg\min_{\Theta} \sum_{r=0}^{R} \left(\mathcal{E}\, T^r(\Theta) - \hat{t}_r\right)^2 \tag{2.21}$$

The choice of the order $R$, $R \geq 2I$, can be made based on the desired accuracy and acceptable numerical complexity. It should be noted that including some higher order moments can improve the representation of the tail of the probability density function. However, since the optimization must satisfy the constraint that the integral of the pdf equals one, the number of unknowns is reduced to $2I - 1$. For example, when $I = 3$ is chosen, there are five unknowns. To solve the optimization more efficiently, the parameters can be substituted, and the constrained problem can be reformulated into an unconstrained one. One possible substitution is given by

$$\mathbf{c} = \begin{bmatrix} \cos(\varphi)^2 \, \sin(\phi)^2 \\ \sin(\varphi)^2 \, \sin(\phi)^2 \\ \cos(\phi)^2 \end{bmatrix} \tag{2.22}$$

to estimate the model parameters, a multidimensional unconstrained nonlinear optimization based on the Nelder–Meads direct search method [33] is adopted. This approach is explained

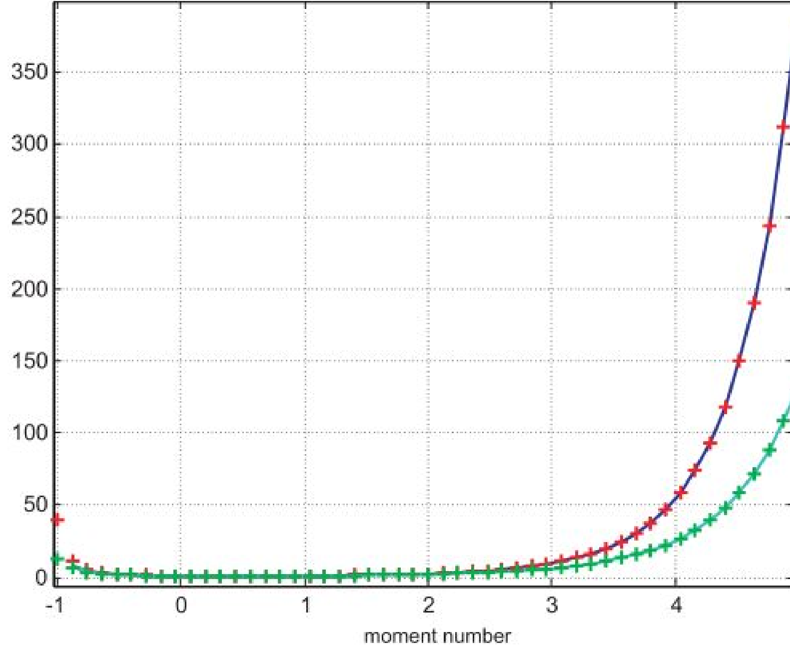



Fig. 2.3. Perfect fit between empirical moments and theoretical values given in equation (14) for the heterogeneous case (red plus/blue curve) and homogeneous case (cyan curve/green plus).

in detail in next section. Adopting this approach for 2.21 using $R = 50$ equidistant values of $r$ in an interval $[-1, 5]$ yields the following results for the two patches (red and yellow) in 2.2

$$\begin{aligned} \hat{\Theta}_{red} &= [0.065, 0.609, 0.326; 1.486, 1.133, 0.470] \\ \hat{\Theta}_{yel} &= [0.081, 0.918, 0.001; 1.158, 0.991, 1.119] \end{aligned} \tag{2.23}$$

The empirical moments and theoretical values given in equation 2.15 are compared in Fig.2.3 for both the heterogeneous case (indicated by the red plus and blue curve) and the homogeneous case (indicated by the cyan curve and green plus). It can be observed that there is a perfect fit between the empirical moments and the theoretical values for both cases, indicating that the parameter values in equation 2.23 provide an accurate representation of the underlying probability distribution. Additionally, based on Fig. 2.1, the match between the measured cumulative density function (cdf) and the 3MD (black curve) for the SAR data confirms the effectiveness of this method over other methods.

## 2.4 Nelder-Mead Simplex Method

The Nelder-Mead simplex algorithm is a popular optimization method for minimizing unconstrained functions in multiple dimensions [33]. It belongs to the class of direct search

algorithms, which do not require explicit computation of derivatives and work well when the objective function is non-smooth, noisy, or has many local minima.

The Nelder-Mead simplex algorithm is often preferred over other optimization methods such as gradient-based methods because it does not require the calculation of derivatives, which can be computationally expensive or even impossible for certain types of functions. The method attempts to minimize a scalar-valued nonlinear function of $n$ real variables using only function values, without any derivative information (explicit or implicit) [3].

One of the advantages of the Nelder-Mead algorithm is its simplicity and ease of implementation. The algorithm only requires the specification of a starting simplex, which can be chosen based on prior knowledge or randomly generated, and a set of tuning parameters that control the size and shape of the simplex. The algorithm operates by iteratively transforming a simplex, which is a geometric object consisting of $n+1$ points in $n-dimensional$ space, with the goal of reducing the objective function value at the vertices of the simplex. At each iteration, the algorithm evaluates the objective function at the vertices of the simplex, and then performs a set of operations to determine how to move the simplex to new vertices with lower function values. The analysis of the method in one dimension is presented in the following section.

### 2.4.1 Nelder–Mead algorithm

The algorithm is proposed as a method for minimizing a real-valued function $f(x)$ for $x \in R^n$. There are four scalar parameters to define a complete Nelder–Mead method: These coefficients are: $reflection(\rho)$, $expansion(\chi)$, $contraction(\gamma)$, and $shrinkage(\sigma)$. According to the original Nelder–Mead paper [33], these parameters should satisfy Eq. 2.24.

$$\rho > 0, \quad \chi > 1, \quad \chi > \rho, \quad 0 < \gamma < 1, \quad and \quad 0 < \sigma < 1 \tag{2.24}$$

The nearly universal choices used in the standard Nelder–Mead algorithm are

$$\rho = 1, \quad \chi = 2, \qquad \gamma = \frac{1}{2}, \quad and \quad \sigma = \frac{1}{2} \tag{2.25}$$

At the beginning of the $k$th iteration, where $k \geq 0$, a non-degenerate simplex named $\Delta_k$ is provided, and it includes $n+1$ vertices. Each of these vertices is a point in $R^n$. The assumption is that the vertices are ordered and labeled as $X_1^{(k)}, \ldots, x_{n+1}^{(k)}$ at the beginning of iteration $k$.

$$f_1^{(k)} \leq f_2^{(k)} \leq \cdots \leq f_{n+1}^{(k)}, \tag{2.26}$$

where $f_i^{(k)}$ denotes $f(x_i^{(k)})$. In the $k$th iteration, a set of $n+1$ vertices is generated, which defines a new simplex for the next iteration, so that $\Delta_{k+1} \neq \Delta_k$. As the objective is to minimize $f$, the vertex $x_1^{(k)}$ is referred to as the "best" point, $x_{n+1}^{(k)}$ is called the "worst"

point, and $x_n^{(k)}$ is referred to as the "next-worst" point. Additionally, $f_{n+1}^{(k)}$ is known as the worst function value, and so on.

One generic iteration is specified by omitting the superscript $k$. Every iteration results in either (1) a single new vertex-*the accepted point*-that replaces $x_{n+1}$ in the next iteration, or (2) when shrink occurs, a set of $n$ new points that, together with $x_1$, form the simplex at the next iteration.

One iteration of the algorithm in discussed in detail as follows:

1. **Order.** Order the $n+1$ vertices to satisfy $f(X_1) \leq f(X_2) \leq \cdots \leq f(X_{n+1})$

2. **Reflect.** The $reflection\ point$ $X_r$ is calculated from the following equation:

$$X_r = \bar{X} + \rho\left(\bar{X} - X_{n+1}\right) = (1+\rho)\,\bar{X} - \rho X_{n+1}, \tag{2.27}$$

   here, $\bar{X} = \sum_{i=1}^{n} \frac{X_i}{n}$ represents the centroid of the $n$ best points (except for $x_{n+1}$). If $f_1 \leq f_r < fn$, the reflected point $X_r$ is accepted and the iteration is terminated. Otherwise, step 3 or step 4 is executed.

3. **Expand.** The expansion point $x_e$ is calculated, if $f_r < f_1$.

$$X_e = \bar{X} + \chi\left(X_r - \bar{X}\right) = \bar{X} + \rho\chi\left(\bar{X} - X_{n+1}\right) = (1+\rho\chi)\,\bar{X} - \rho\chi X_{n+1}, \tag{2.28}$$

   If $f_e < f_r$, the expansions point, $X_e$, is accepted and the iteration terminates; otherwise, (if $f_e \geq f_r$), $X_r$ is accepted and the iteration terminates.

4. **Contract.** If $f_r \geq f_n$, *contraction* between $\bar{X}$ and the better of $X_{n+1}$ and $X_r$ performs:

   a. **Outside.** An *outside contraction* is performed and $X_c$ is calculated, if $f_n \leq f_r < f_{n+1}$ (i.e., $x_r$ is strictly better than $x_{n+1}$).

$$X_c = \bar{X} + \gamma\left(X_r - \bar{X}\right) = \bar{X} + \gamma\rho\left(\bar{X} - X_{n+1}\right) = (1+\rho\gamma)\,\bar{X} - \rho\gamma X_{n+1}, \tag{2.29}$$

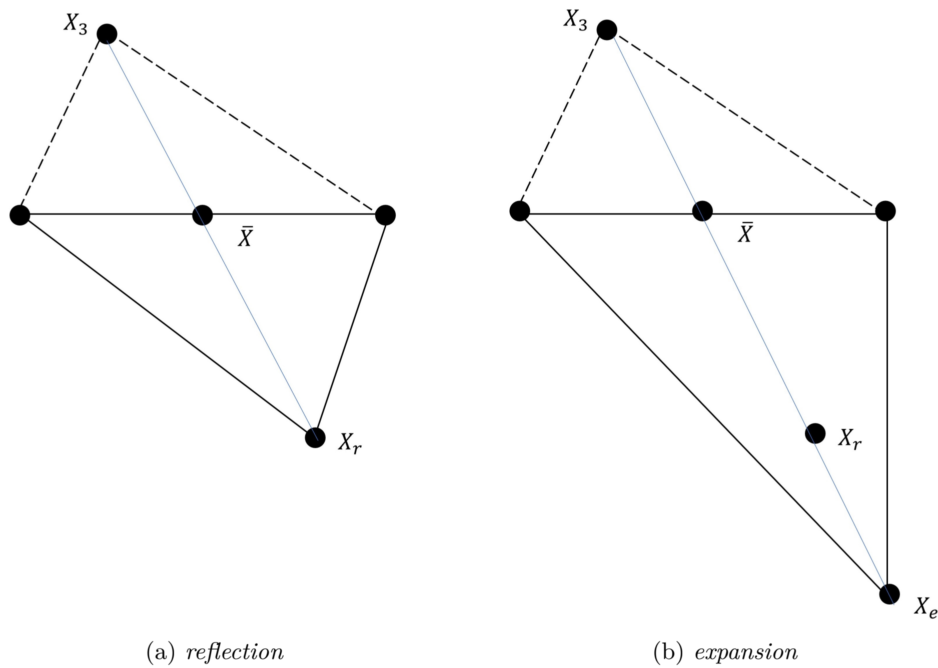
   If $f_c \leq f_r$, $X_c$ is accepted and the iteration terminate; otherwise, step 5 is executed(perform a shrink).

   b. **Inside.** Perform an *inside contraction* and calculate $X_{cc}$, if $f_r \geq f_{n+1}$,

$$X_{cc} = \bar{X} - \gamma\left(\bar{X} - X_{n+1}\right) = (1-\gamma)\,\bar{X} + \gamma X_{n+1}, \tag{2.30}$$

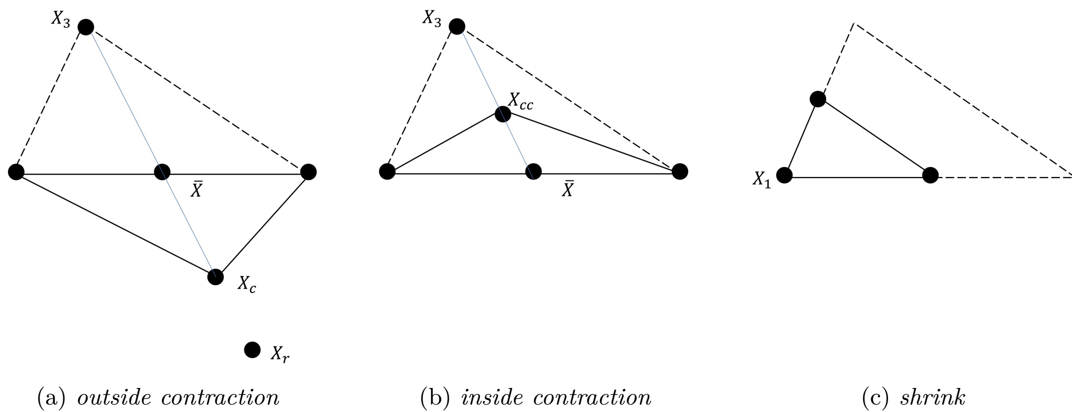
   If $f_{cc} < f_{n+1}$, accept $X_{cc}$ and terminate the iteration; otherwise, step 5 is executed(perform a shrink).

5. **Shrink.** Calculate $n$ new points $V_i = X_1 + \sigma(X_i - X_1)$, $i = 2, ..., n+1$. The (unordered) vertices of the simplex at the next iteration consist of $X_1, V_2, ..., V_{n+1}$.




(a) *reflection* (b) *expansion*

Fig. 2.4. Nelder–Mead simplices after a reflection and an expansion step. The original simplex is shown with a dashed line [3].




(a) *outside contraction* (b) *inside contraction* (c) *shrink*

Fig. 2.5. Nelder–Mead simplices after an outside contraction, an inside contraction, and a shrink. The original simplex is shown with a dashed line [3].

Figures 2.4 and 2.5 depict how reflection, expansion, contraction, and shrinkage affect a two-dimensional simplex (a triangle), when the standard coefficients $\rho = 1$, $\chi = 2$, $\gamma = 0.5$, and $\sigma = 0.5$ are used. It can be observed that, except for a shrink, the newly added vertex is always located on the line that connects the centroid $\bar{X}$ and the vertex $X_{n+1}$. Additionally, it is apparent from the figures that the shape of the simplex undergoes a significant transformation during an expansion or contraction when using the standard coefficients. In this chapter, the detection algorithm that involves the declaration of a ship

(or a target, in general) when a pixel value exceeds a predetermined threshold, which is calculated based on 3MD was introduced. The Nelder-Mead algorithm was also explained in detail to find the optimal necessary parameters for statistically modeling the threshold. In the next chapter, the FPGA implementation of automated ship detection system is explained in detail.

# Chapter 3

# FPGA Implementation of Automated Ship Detection Algorithm

In previous chapter, the cluttering model and the Nelder-Mead simplex for ship detection algorithms were explained. These two methods are the main parts of a ship detection and need to be precisely implemented. Since these algorithms are computationally expensive and time consuming, and the ASD needs to perform real-time, accelerated implementations of these algorithms are necessary. There are different platforms available for accelerated ASD implementation, including CPU, GPU, and FPGA:

1. CPU: Central processing unit, the most common platform, is the primary processor in most computers and is capable of executing complex algorithms. However, CPUs are generally designed for general-purpose computing and may not provide the best performance for specific applications and real-time implementation. Although, they are powerful, they are designed mostly for controlling tasks.

2. GPU: Graphics Processing Units are designed to handle large amounts of parallel computations and are optimized for certain types of algorithms. They are often used for tasks such as machine learning and scientific computing, where massive amounts of data need to be processed quickly. An implementation of the ASD on a typical GPU although can be real-time, it consumes a lot of energy since the it cannot be customized to suit the algorithm. As mentioned in Chapter 1, power consumption is a significant aspect of any RADAR, makes GPUs less useful to this application.

3. FPGA: Field Programmable Gate Arrays are programmable integrated circuits that can be customized for specific tasks. FPGAs can be programmed to perform specific tasks more efficiently than general-purpose CPUs and GPUs, making them a popular choice for accelerating algorithms. FPGAs usually consume less power compared to GPUs for the same throughput. Nowadays FPGAs, are very powerful in terms of

computation but, they suffer from low in-chip memory and off-chip bandwidth. In ASD case, memory consumption is relatively low which makes the FPGAs the best option for real-time implementation of the ASD algorithm.

The rest of this chapter is organized as follows: Section 3.1 explains the various FPGA resources available for ASD design. The input and output of ASD are introduced in section 3.2. In section 3.3, the architecture of the ASD system and a summary of the its algorithm is explained. The implementation detail of the ASD is then discussed in section 3.4, followed by presenting the implementation results in section 3.5.

## 3.1 FPGA Resources

Xilinx FPGA is a type of programmable logic device that can be customized to perform specific digital functions. It consists of configurable logic blocks, memory resources, and programmable interconnects, which can be programmed using a hardware description language like Verilog or VHDL.

In addition to logic resources, Xilinx FPGAs also have built-in high-speed transceivers, memory interfaces, and digital signal processing blocks. These resources can be used to implement high-speed serial communication interfaces like PCIe, Ethernet, and high-performance digital signal processing functions like FFTs, and image processing algorithms. In general, there are 3 main components in the Xilinx FPGAs which are categorizes as follows:

1. **Digital Signal Processing Blocks:** Xilinx FPGAs have specialized Digital Signal Processing (DSP) blocks that can perform complex mathematical operations on digital signals with high precision and throughput. These blocks include adders, multipliers, accumulators, and other digital signal processing functions. These DSP blocks are optimized for high-speed, parallel processing of large data sets, making them ideal for applications such as digital filtering, FFT processing, and other signal processing algorithms.

2. **Memory Resources:** Xilinx FPGAs contain different types of memory resources such as block RAM (BRAM), distributed RAM, and UltraRAM (URAM). BRAMs and URAMs are types of memory resource that can be used for storing large amounts of data, such as image or video frames, and is optimized for high-speed access. Distributed RAM is a type of memory that is distributed throughout the FPGA and can be used for storing small amounts of data, such as lookup tables or coefficients. There are two separate read/write ports that can have different write modes. BRAMs are in 36kb blocks which can be configured in different bit width such as 8K $\times$ 4, 4K $\times$ 9, 2K $\times$ 18 or 1K $\times$ 36. Unlike BRAM, URAM is available in fixed blocks of 4K $\times$ 72 bits.

3. **Configurable Logic Block:** Xilinx FPGAs consist of configurable logic blocks that can be programmed to perform various logical operations, such as AND, OR, XOR, and NOT. These logic blocks which are look-up tables (LUT) and Flip-Flops (FF), can be connected in different configurations to create custom digital circuits, such as adders, counters, and multiplexers. The number of logic blocks in an FPGA determines its processing capabilities, and Xilinx FPGAs can have anywhere from thousands to millions of logic cells, depending on the device family and configuration.

Overall, the combination of DSP blocks, memory resources, and logic cells in Xilinx FPGAs provides a versatile platform for implementing complex digital systems, making them popular in applications such as telecommunications, data centers, aerospace, and defense.

## 3.2 Input and output

Before digging into the details of ASD architecture, it's necessary to introduce how the ASD receives the inputs its inputs. The ASD system receives 3 complex images, namely "*full*", "*aft*," and "*fore*", as inputs. Let's assume $Z_i = X_i + jY_i$ is the complex image produced by a typical ScanSAR mode, with size $1152 \times 512$ pixels. ASD receives real and imaginary part in two different stream channels in a column-major order. Every pixel of Each channel is a 12-bit signed integer sampled at 250 MHz. The ship detection algorithm only utilizes the "*full*" channel, while the remaining two channels are utilized in the velocity measurement module. The ASD's output is a report that provides information such as location, size, head angle, and velocity of the detected targets, which could either be ships or icebergs since both appear similar in SAR images. In general, the detected target of the ASD in this chapter is called ship. However, they can be either a ship or and iceberg since both ship and iceberg can appear similar in SAR images. In the next chapter, the detected targets from ASD will be classified to either ships or icebergs by leveraging a ship/iceberg discriminator.

## 3.3 ASD Architecture

A typical Scan SAR mode designed for a wide coverage mission produces burst images around 5 to 20 km along-track (depends on band or synthetic aperture length) by 30 to 80km cross-track. Such an image contains anywhere between 20 and 240 megapixels, assuming a pixel spacing of 2.5 x 2.5m. A single serial link transfer of the image data to ASD at 250 MSPS (12-bits per sample) requires 750ms to do so. The image data passes through the ship detection algorithm at least four times (mean, moments, threshold, and finally pixel detection). This implies that, it requires $3s$ to transfer data through the algorithm.

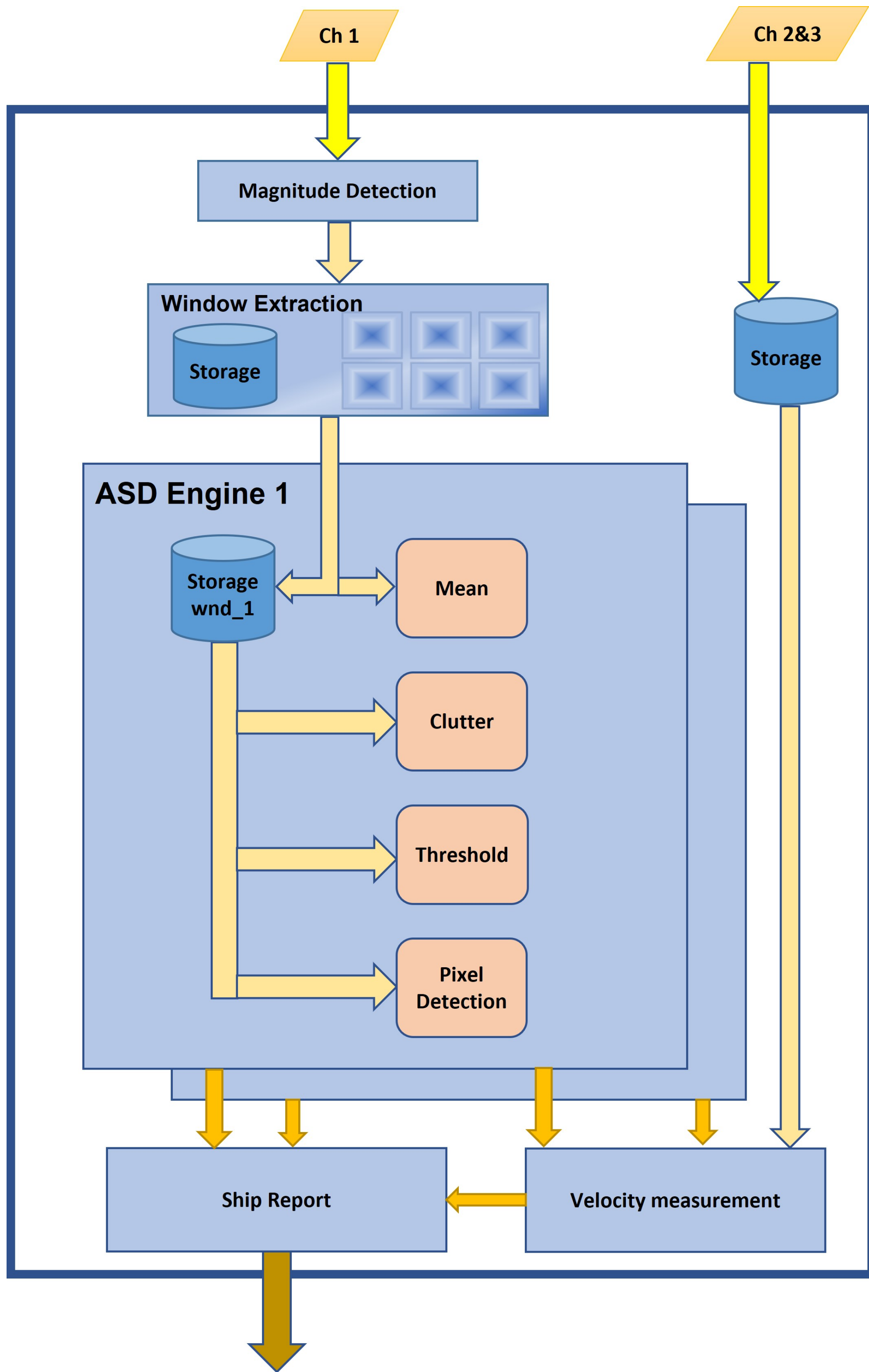



Fig. 3.1. Automated Ship Detection Unit Architecture

However, for the typical Scan SAR mode, there is only $\leq 1s$ burst time available before the next image data processing starts. This makes it clear that one should segment the image into at least three windows to transfer and process the windowed images in parallel. This also means that, three copies of the ship detection engines are necessary.

On the other hand, the mean back-scatter of ocean is a function of incidence angle. Large cross-track extent will therefore have a sloped mean pixel value and any threshold will apply at least locally over small range extent unless the image is first normalized to remove the slope. Smaller windows allow one to ignore this effect.

Additionally, FPGA memory (BRAM (36 kbit) and URAM (288 kbit)) is available in discrete blocks. Each ship detection engine requires temporary storage for the windowed image, and it is therefore a good idea to select the window size such that the memory allocated for temporary storage fits within an integer number of the discrete memory blocks. It's obvious that dividing the input image to more windows and parallelizing the number of ship detection engines leads to an increase in the throughput of the whole system. However, this results in more resource usage. An optimal number of windows and number of parallel engines need to be determined to optimize both the throughput and the resource usage.

The architecture of the designed ASD is illustrated in Fig. 3.1. First, the complex image goes through the magnitude detection and windowing modules. Then, the automated ship detection engine will be executed on the magnitude detected windows, records the detected targets along with their size and locations. When a target is detected, the velocity measurement module extract patches around the target and then measures the velocity of the target. Finally, a ship report generator combines the generated reports from each engine.

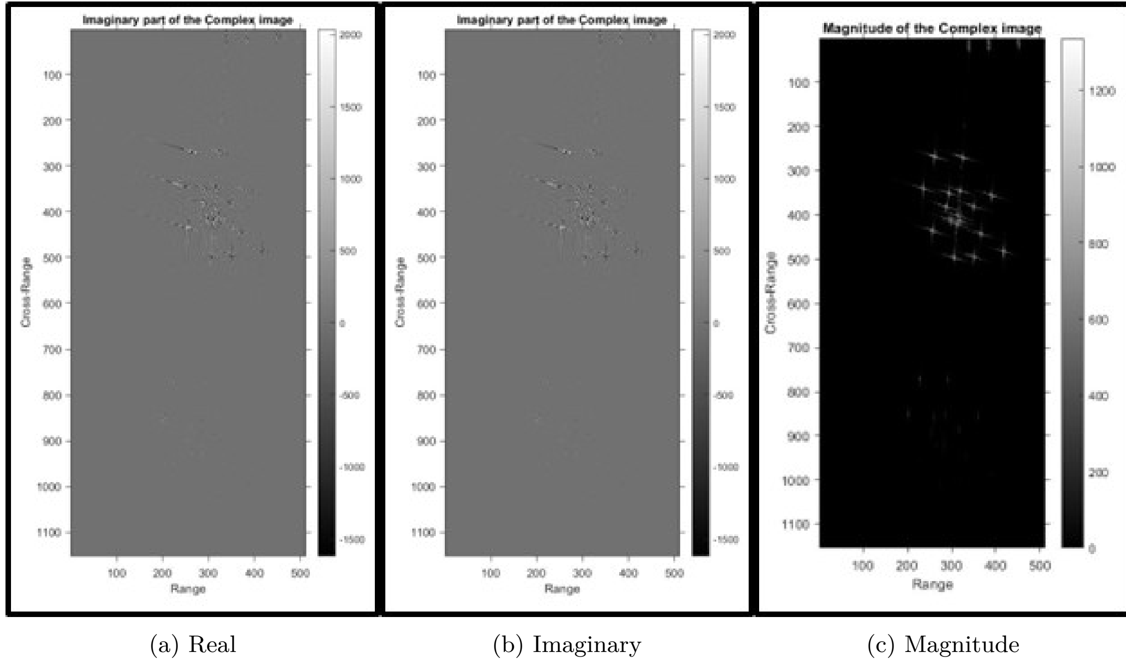
## 3.4 FPGA Implementation

### 3.4.1 Magnitude Detection

The image for ship detection is a complex image (real and imaginary part), but the detection can be performed only on the magnitude of this image. The magnitude detection function is the first function applied to the input complex image, which only calculates magnitude of the incoming pixels without storing them. The magnitude of a complex value is calculated by the following equation:

$$|Z| = \sqrt{X^2 + Y^2} \tag{3.1}$$

To minimize the resource usage, we can omit the square root and calculate the squared magnitude. Fig. 3.2 illustrates real and imaginary parts of the input images along with its magnitude. It is clear that targets in the magnitude image is much clearer.




(a) Real (b) Imaginary (c) Magnitude

Fig. 3.2. Applying magnitude detection on a complex image. (a) real part. (b) imaginary part. (c) magnitude.

**Bit allocation:** Since the input image to magnitude detection module is a 12-bit signed integer, range from $-2048$ to 2047, the squared of each signal needs 24-bit signed integer for correct calculation. In practice, we can allocate one less bit for the squared signal since it's very unlikely that a pixel value gets a value of -2048. The benefit of allocating one less bit for the signal is far greater than accepting the risk of a miscalculation. Additionally, since the result of the squared operation is always positive, the sign value is always zero. We can change the signal attribute to a 22-bit unsigned integer to further decrease the resource usage. Finally, the output the addition is truncated to 12-bit unsigned integer because lower bits have less impact on the detection.

### 3.4.2 Window Extraction

As mentioned in section 3.3, it is necessary to divide the input image to several windows to feed ASD engines to be able to reach a real-time detection. The total number of windows and number of engines depend on several things such as available HW resources, burst time of input images, processing time in each engine, and size of each window. For example, if the window size is too short, most of windows will not contain any target and thus is not worth to be processed for the detection.

Extracted windows will be stored in FPGA URAMs which are in blocks of $4k \times 72$bits, and since the input signal is $12 - bit$, we can store 6 samples in one address line of URAM to optimize the memory usage. Additionally, since each engine has its own memory to store

the whole window data, the windowing module only passes windows for the first numbers equal to the number of engines. Moreover, since the input signal comes in a column-major order, if we divide the image into 2 or more windows column-wise, the engine data will be ready sooner. Width of the input image is 512 and therefore, dividing the image into two columns is a good choice to have a proper window size.

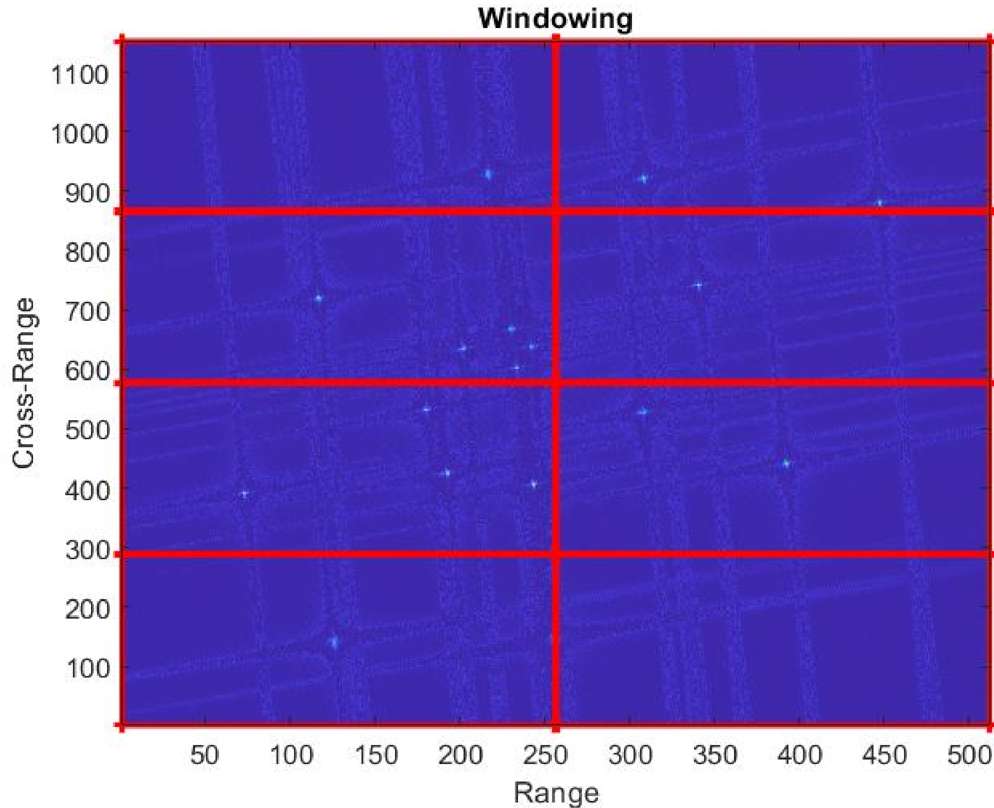


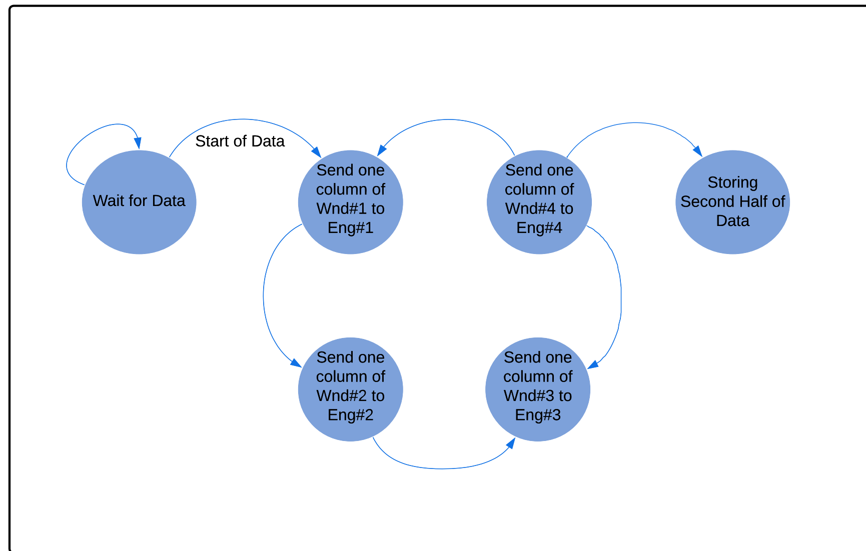

Fig. 3.3. The concept of dividing a magnitude detected image into multiple windows.

Based on a further resource usage estimation of engines, we can have 3 to 6 copies of engines to work in parallel. We have fixed the number of engines to 4, so we can divide each input image into 4 rows of windows to optimize the resource usage. With the current setting, an input image is divided into 8 windows (4 rows and 2 columns). The first half of the image is passed through windowing module without being stored, but the second half of the image will be stored in URAMs. The Window module performs the following functions:

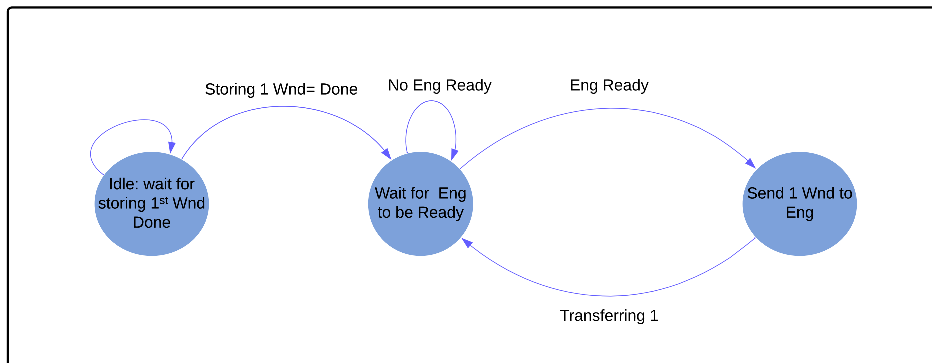
- The module will subdivide a burst image into windows of configurable size.
- The number of windows is currently fixed to 8.
- Transfer the windows to the ASD-ENGS for ship detection and control the data flow depending on the availability of engines. There will be less engines (4) than windows

(8), so a round robin routing scheme is implemented where engines are monitored for availability.

- The first 4 windows will be automatically transfer to the next module without being stored. In the beginning of receiving data, we assume that all 4 engines are ready to receive data.
- The other 4 windows are stored in this module and ready to be transferred to the next available engines.




(a) First half




(b) Second half

Fig. 3.4. The state machine for dividing the magnitude detected image into windows. (a) Transferring the first half and storing the second half of the image. (b) Reading the second half windows.

An illustration of the concept of windowing is shown in Fig. 3.3. The state machine for handling the transferring, storing, and reading the windows is illustrated in Fig. 3.4. The controller is implemented using VHDL and the exported to SysGen as a black box, and the storage part, is implemented with a Dual port URAM SysGen component.

### 3.4.3 ASD Engine

The Automated Ship Detection Engines (ASD-ENG) operates on windows of magnitude detected images. The ASD ship detection engine is an autonomous unit capable of ingesting and temporarily storing the windowed image data, for use in the calculation of the window mean, moments, pixel detection threshold and then to apply the threshold to perform pixel detection on the same window image. It also includes a clustering algorithm for grouping detected pixels into objects (they are referred to as ships, with no regard to the philosophical implication of doing so). Some key properties of the objects, such as spatial size and orientation measured for inclusion in the ship report. The location of the center of the objects are used to extract the *aft*- and *fore* single aperture image chips for velocity measurement. The output of the ship detection engine is a stream of data defining the ship including the SLC image chip as well as the power detected fine resolution image. The Automated Ship Detection Engines include the modules that are summarized as follows:

1. AE-K-FSM: Local FSM for the ASD engines. Manages the enabling, configuring, loading, processing, unloading and resetting of the individual detection engines.

2. AE-K-STORE: This module receives and stores the windowed image in dual port RAM. It transfers the image and image valid signal to the computational modules for calculating window statistics, moments and finds the optimal coefficients describing the trimodal distribution model and finally passes the image through the pixel detection algorithm after the threshold has been calculated.

3. AE-K-STATS: Calculates the statistics (mean, minimum and maximum values) of the winnowed image.

4. AE-K-MOMS: Calculates the statistical moments of the clutter in the image after removing the perceived objects.

5. AE-K-3MD: Implements the 3MD algorithm to find the optimal parameters for threshold estimation.

6. AE-K-THRSH: Determine the threshold for a CFAR PFA configurable parameter.

7. AE-K-PIXDET: Applies the threshold to the windowed image to detect pixels exceeding the threshold.

8. AE-K-SPAT: Clusters the detected pixels into objects/ships and measures spatial attributes, such as its centroid, length, width, and orientation. Combines these parameters into a set of which is passed to create the ship report.

Implementation of the above-mentioned modules will be explained in detail in the following sections.

### 3.4.4 AE-K-FSM

The module consists of a state machine, which provides a hardware implementation of the state controller allowing the engine to move through sequential steps of the algorithm to calculate the mean (statistics), moments and finally the detection threshold, all based on the values of the pixels of the windowed image. This state machine controls the reading of the stored data and delivery to the sequential algorithmic step. The image data needs to be stored and then read from local memory each time a data dependent algorithmic step is performed. The first time the data passes into the module on its way to be stored, the hardware unit AE-K-STATS calculates the mean of the pixel values below the zeta threshold. The unit implements temporary storage of the windowed image allowing reuse several times. The state transition diagram is illustrated in Fig. 3.5.

### 3.4.5 AE-K-STORE

Each engine needs its own memory to store and read the window data many times since the algorithm requires several passes of the magnitude detected data. Each window has $256 \times 288$ 12-bit pixels which is equal to 884,736 bits. To store this data, we can either use BRAMs or URAMs, but not the LUT-based memory. We used URAM memories to store the whole input image, therefore it's better to allocate BRAM to store each engine data. FPGA BRAM is configurable as one $1k \times 36$bit, or two independent $1k \times 18$bit. With 12bit word length, using $1k \times 36$bit configuration optimizes the memory usage by concatenating 3 samples and storing them in one address line. Additionally, since the reading from and writing to this memory will not occur simultaneously, we can use a single-port BRAM to optimize the resource and power usage. The state machine for this module is implemented using VHDL and imported as a black box into the SysGen model. There are only 3 states in the state machine, one for waiting to receive new window data, one for storing the data, and one for reading the stored data. The stored data will be read multiple times to feed data to the next modules.

### 3.4.6 AE-K-STATS

In this module, the mean of pixels with values greater than zero is calculated. To do this, the window data needs to pass through it, to accumulate the pixels with values greater than zeros and count the total number of these pixels. Then, the mean is obtained by dividing

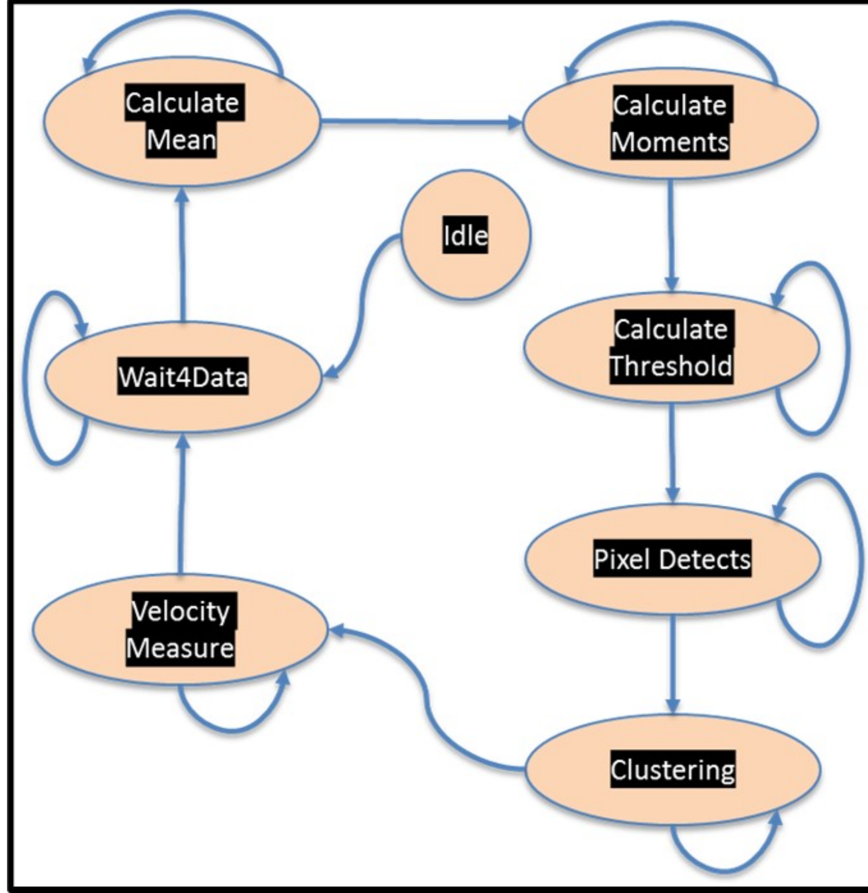



Fig. 3.5. Finite State Machine Transition Diagram for the AE-K-FSM Module

the accumulated value by the number of pixels. Since division operations are time and resource consuming, and the exact value of the mean does not affect further calculation in the algorithm, we can approximate the implementation of the division. The simplest way to approximate the division in FPGA is to implement it as *right-shift* operation. Instead of dividing the sum by total number of nonzero values, we shift right the sum by $floor(log2)$ of this number. The result is the approximated mean signal value for the windowed image.

**Bit allocation:** In this module, the sum of pixels with nonzero values is calculated. Since there are $288 \times 256 = 73728$ pixels of 12bit unsigned integer, the accumulator needs $12 + Ceil(log_2(288 \times 256)$ bits to correctly calculate the sum. The variable which counts the total number of nonzero pixels needs $Ceil(log_2(288 \times 256)) = 17$ bits. Finally, the output mean value will be 12 bit unsigned integer.

### 3.4.7 AE-K-MOMS

This module calculates the moments set of each window. A 6th order moment set is desired. Moments equations are illustrated in the following:

$$Moments(i) = \frac{N^{i-2} \times \sum \left(A^{i-1}\right)}{\left(\sum A\right)^{i-1}} \quad i = 1,\, 2, \cdots, 6 \tag{3.2}$$

where $A$ is the pixel values less than a constant zeta-threshold $(\zeta \times \mu)$, and N is the total number of these pixels. It is obvious that based on this equation, the first two moments are equal to 1. Therefore, there is no need to calculate the first two moments in hardware.

This equation is relatively complex and we need to accelerate it precisely. We use fixed-point representation with different bit-width and some approximation of the equation to speed up its implementation. To implement this equation, we have divided this module to three sequential units. The first unit is to calculate the numerator of Eq. 3.2, and streams the window data pixel values less than the zeta-threshold (excluding large values derived from the image statistics) through a set of multipliers (generating powers of 1 to 5) of the signal values to a set of signal accumulators. On completion of the window, the accumulated signal values are transferred to the next unit, which calculates the powers of the number of pixels below the zeta-threshold and the powers of the sum of the pixel values in the window. Finally, in the last unit, the moments will be calculated by implementing the multipliers and division. The implementation details of the above-mentioned units is explained as follows:

1. The first unit in AE-K-MOMS which calculates $\sum \left(A^{i-1}\right)$ and $N$, receives the window data, window mean, and a constant $\zeta$ as the inputs. The input pixels is first compared to $\zeta \times \mu$, and in case they are less than this value, goes through a tree of multipliers to calculate the power. We use a multiplier tree to reduce multiplier usage, but for a higher latency. The increase in latency is trivial compared to the whole calculation time, thus make the tree multiplier beneficial. **Bit allocation:** Window data are 12-bit unsigned integer and to allocate bits for each power, we just need to multiply the power to this value to determine the maximum bit-width. Since this unit only calculates the power for the pixels with values less than a threshold, we can reduce the bit-width assigned to each calculate power. To determine the maximum bit-width, we have investigate the mean of window with values less than the threshold to estimate the bit-width. Based on the calculation, the mean value is not greater than 512 for 24 windows. Therefore, we can only allocate $9 \times power$ to calculate the powers. The accumulator needs 17 extra bits to accurately calculate the sum. Table 3.1 shows the bit-width for each power accumulation.

Table 3.1. Number of bits allocated to the sum of each power

| Power | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Number of bits | 26 | 35 | 44 | 53 | 62 |

2. The second unit is responsible to calculate the $N^{i-2}$ and $(\sum A)^{i-1}$. It receives two inputs, $N$ and $\sum A$ and calculates 4 different powers of these two inputs. For the implementation, one multiplier trees such as the one in unit 1 is used per each input to reduce the resource usage.

**Problem:** There are 2 inputs to this unit, $N$ and $\sum A$, which are 17 and 26 bits unsigned integer, respectively. We calculate the first 4 powers of the $N$ and the first 5 powers of $\sum A$. To accurately calculate these values, we should allocate $4 \times 17 = 68$ and $5 \times 26 = 130$ bits to store the resulting values. These number of bits are too large for FPGAs to do the following calculations. Therefore, we need to limit the number of bits allocated to each values. One way to do this is to use Floating Point representation. We can use single or double floating point to calculate the powers, but eventually, we need to use fixed-point representation since floating point operations are very costly in FPGAs and the resulting powers will be used in other equations which consist of fixed-point numbers. Therefore, it is not efficient to use floating point representation in this equation.

**Solution:** To overcome this issue, we used approximate calculations. First, we scale down the input values, to get a fraction part. Then, we use fixed-point representation with fixed number of bits for integer and fraction parts for all the calculated powers. Although this method is very similar to floating point representation, it does not need to be implemented by using floating point operations, thus will improve both resource usage and throughput. However, the scaling factor and the number of bits allocated for integer and fraction parts should be carefully chosen so that the results would not change significantly. The scaling factor for $N$ and $\sum A$ is set to 15 and 20. To calculate the powers, we use $fi(0, 32, 14)$ representation for all the values, which means a $32-bit$ unsigned value with 14 bits fraction part. 3. In the last unit, the last division and multiplication are implemented. We first implement the division to deal with relatively smaller numbers in the multiplication part. We have first implemented $\frac{\sum(A^{i-1})}{(\sum A)^{i-1}}$, and then multiplied the results by $N^{i-2}$.

**Bit allocation:** In the previous part, the $(\sum A)^{i-1}$ values are scaled down by a factor of $2^{20}$, making them relatively small values. In opposite, $\sum (A^{i-1})$ values are relatively large values, makes the division very hard to implement with high precision. Hence, we have scaled down these values with a factor of $2^{20}$ to make the numerator relatively small compared to the denominator. To implement the division, we used Xilinx Divide IP with High_Radix algorithm and 14 bits output fractional width for all the division results. Therefore, the outputs has 14 extra bits compared to the numerators.

The outputs of the divisions are then multiplied by $N^{i-2}$. Since the scaling factors for $(\sum A)^{i-1}$ and $\sum (A^{i-1})$ are not equal, we need to scale up the results of multiplications to adjust the results. Eq. 3.3 calculates the factor for scaling up the final results to the correct scales. The scale factor for each moment is different since they are related to the moments

power.

$$Result_j = \frac{\left(\frac{N}{2^{15}}\right)^{j-1} \times \frac{\sum(A^j)}{2^{20}}}{\left(\frac{\sum A}{2^{20}}\right)^{j}} = \frac{Moment(j) \times 2^{20j}}{2^{15j} \times 2^{5}} = \frac{Moment(j) \times 2^{5j}}{2^{5}}, for\, j = 2, 3, 4, 5 \tag{3.3}$$

The implementation of this module contains some approximation to fit the design to the FPGA. We need to evaluate the implementation in terms of speed and accuracy to compare it with a software design. Table 3.2 illustrates the calculated moments of a given window from both software and FPGA. The FPGA results are from SysGen simulation with fixed-point representation and the reference software results are based on Double-Precision Floating Point. The first two moments are discarded from this table since they are not calculated in hardware. The post implementation timing analysis of the this unit guarantees that this module can run with the predefined clock frequency (250 MHz). Also, this unit only needs 60 cycles to calculate the moments after receiving the whole window pixels.

Table 3.2. Calculated moments in software and hardware.

| Parameter | Reference Result | SysGen Result | Error(%) |
|---|---|---|---|
| Moment 3 | 1.6460 | 1.6464 | 0.02 |
| Moment 4 | 5.3662 | 5.3661 | ~0 |
| Moment 5 | 31.2613 | 31.1063 | 0.49 |
| Moment 6 | 236.8814 | 235.4915 | 0.58 |

### 3.4.8 AE-K-3MD

This module is the main part of the engine which calculate the 5 parameters of the trimodal cluttering algorithm to further calculate the threshold. To find the optimal values, we have implemented the Nelder-Mead algorithm as follows:

1. Based on the discussion in section 2.3.3, we need to find the optimal values for 5 parameters ($\Theta = [\varphi, \phi, a_1, a_2, a_3]$). This means $n = 5$ in the Simplex algorithm, which leads to $n + 1 = 6$ vertices.

2. The cost function in Nelder-Mead algorithm is based on Eq. 2.21, which calculates the sum of square of difference between the measured moments and the theoretical moments. The measured moments are calculated in AE-K-MOMS and theoretical moments are calculated by Eq. 2.15.

3. There are 2 conditions that terminates the algorithm: (a) maximum number of iteration which is determined by user, and (b) the maximum absolute difference for both function values and simplex points with the best vertex are less than a tolerance value.

Table 3.3. Initial values of the parameters for Nelder-Mead algorithm

| Parameter Names | $\varphi$ | $\phi$ | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|---|---|
| Simplex 1 | $\frac{\pi}{4}$ | $\frac{\pi}{4}$ | 1 | 1 | 1 |
| Simplex 2 | $\frac{\pi}{4} \times 1.05$ | $\frac{\pi}{4}$ | 1 | 1 | 1 |
| Simplex 3 | $\frac{\pi}{4}$ | $\frac{\pi}{4} \times 1.05$ | 1 | 1 | 1 |
| Simplex 4 | $\frac{\pi}{4}$ | $\frac{\pi}{4}$ | $1 \times 1.05$ | 1 | 1 |
| Simplex 5 | $\frac{\pi}{4}$ | $\frac{\pi}{4}$ | 1 | $1 \times 1.05$ | 1 |
| Simplex 6 | $\frac{\pi}{4}$ | $\frac{\pi}{4}$ | 1 | 1 | $1 \times 1.05$ |

The algorithm takes quite a long time since it may needs numerous iterations to converge. Thus, we need many resources so that the implementation is done in parallel to reach the fastest possible throughput. Additionally, the calculations in this algorithm need to be precise to acquire the optimum values, since the final detection depends on the exact values of the parameters, which makes the implementation more resource hungry. On the other hand, we cannot allocate too much resource only for this module, because there will be more copies of this module in other engines. We need to optimize the resource usage for this module such that the throughput is maximized, the calculation is accurate, and the resource usage is minimized.

The algorithm is implemented in 3 phases, 1. Initialization, 2. Main loop, and 3. Termination decision. These 3 phases are discussed in details in the following:

**1. Initialization Phase**: In the first part of the algorithm, we initialize the parameters, vertices, function values, and sort them. We aim to initialize the five parameters in a way that allows the algorithm to converge as quickly as possible. To achieve this, we tested various random cases with test images and selected the values that resulted in the minimum convergence iteration. These optimal values are presented in Table 3.3.

In the initialization phase, the next step is to calculate the cost function for each simplex. One approach is to implement a single copy of the function and use it for all the inputs in a pipeline fashion. Alternatively, we could implement a copy of the function for each simplex. Since the calculation is performed only once in the initialization phase, using a single copy of the function can save resources. We can then feed one simplex per cycle to the implemented function to calculate its cost function.

To implement the cost function, the first step is to calculate Eq. 2.22. It is done by using two Cordic 6.0 Xilinx IP Cores. The implementation supports pipeline with minimum latency and maximum precision. The output has 18 bits, which will be fed to some multipliers to calculate $c_1$, $c_2$, and $c_3$. The next part in calculating the cost function which is in parallel to the first part is to calculate Eq. 2.9. This equation can be revised for hardware implementation to be more hardware friendly and save some more resources. We use Eq. 3.4 to calculate $\hat{b_1}$, $\hat{b_2}$, and $\hat{b_3}$, to substitute the square operators with absolute one and

constant multiplication with shift operation. Since $\rho_c = 0.75 = 0.5 + 0.25$ is constant, we use two right-shift operation to do the multiplication.

$$b_i = \rho_c \, |a_i| + \rho_n \tag{3.4}$$

$$\tilde{b_i} = |a_i| \gg 2 + |a_i| \gg 1 + \rho_n, \; for \; i = 1, 2, 3 \tag{3.5}$$

where $\gg n$ is n-bit right-shift operation. Next, to calculate $\left(\tilde{b_i}\right)^j, for\, j = 2, 3, 4, 5$, we simply use one multiplier tree for each $\tilde{b_i}$, to minimize the latency. Then, we use some multipliers and addition to implement the summation in the Eq. 2.15 . For the next calculation, we have to multiply the results to some constant values. These values are shown in Table 3.4. The multiplication in this part also can be substituted by shift operation. Table 3.4 also shows these values being consist of addition/subtraction of power of 2 integers and the simplified shift operation to be replaced for the multiplications. Take $\times 120$ as an example, we can rewrite it as $\times 128 - \times 8$ which can be implemented by two left-shift operation and one addition, resulting in saving some DSPs.

Table 3.4. Gamma values and their representation as addition/subtraction of power of 2 integers

| Gamma values | 1 | 2 | 6 | 24 | 120 |
|---|---|---|---|---|---|
| Addition/subtraction of power of 2 | 1 | 2 | $4 + 2$ | $16 + 8$ | $128 - 8$ |
| Shift Operation | - | $\gg 1$ | $\gg 2 + \gg 1$ | $\gg 4 + \gg 3$ | $\gg 7 - \gg 3$ |

After calculating all the cost functions, they need to be sorted. For the sort algorithm we have used the Merge Sort algorithm, to be as fast as possible. It only takes 6 cycles to sort the 6 function and vertices values. After sorting the cost functions, we sort the 6 vertices, and the initialization section is finished.

**2. Main Loop:** The next part in this module is implementing the main loop of the algorithm. Each iteration receives 6 sorted functions along with their vertices, then calculates 4 new vertices, including reflection, expansion, outside contraction and inside contraction in parallel. Function values of these vertices are calculated in parallel using a function calculator module per each vertex. To improve the algorithm speed, we will calculate the shrinkage points and their function values in parallel. As mentioned in Section 2.4.1, the result of each iteration is either (1) a single new vertex—the accepted point—which replaces $X_{n+1}$ in the set of vertices for the next iteration, or (2) if a shrink is performed, a set of n new points that, together with $x1$, form the simplex at the next iteration. To make this decision faster and with less resources, we first compare $X_r$ with $X_5$ to see if the decision is a *reflection/expansion* or a *contraction/shrink*. This way the decision unit will consume less comparisons. If a *reflection/expansion* occurs, we need to compare the $X_r$, $X_e$, and $X_1$, and replace the accepted point with $X_{n+1}$. In case a *contraction/shrink* happens, we first

check if it is an outside contraction or inside contraction by comparing $(X_r, X_6)$ and then comparing ($X_c$, $X_r$) for outside contraction and ($X_{cc}$, $X_6$) for inside contraction. In case a shrink occurs, we replace one of the shrinkage points with $X_{n+1}$, and change a shrink flag as true. If the shrink flag is true, another unit will replace ($X_2$, $X_3$, $X_4$, $X_5$) vertices by already calculated shrink points ($V2$, $V3$, $V4$, $V5$). Then, the 6 new vertices and their function values will be sorted either for the next iteration or for checking the algorithm termination. The sort function is the same as used in the initialization section.

**3. Termination decision:** As mentioned earlier, there are two conditions which make the algorithm to terminate. The first one is the iteration exceeds a maximum iteration and the second one is if the maximum absolute difference for both function values and simplex points with the best vertex are less than a tolerance value (the improvement in the optimal point and its function is less than a tolerance). In the decision unit, we first check the iteration number and if it is not exceeding the maximum, we trigger the next iteration. This way, the next iteration starts without waiting for the result of the second condition. If the iteration number is greater than the maximum iteration, the algorithm terminates, and the optimal point is being selected. Otherwise, we calculate the maximum absolute difference. There are 6 vertices, and each one has 5 parameters. Therefore, there will be 25 subtraction and comparison the find the maximum. We calculate the absolute difference and the maximum of each parameters in parallel, but for different vertices in one cycle. This way, we parallelize only 5 subtractions and comparisons and in 5 cycles we find the maximum absolute difference for each parameters. Then, we compare these 5 values to find the final maximum absolute difference values. If the termination condition is true, we send a stop signal to the previous module to stop the iteration and a done signal which indicates that the optimal vertex was found. When the algorithm terminates, the best $\varphi$ and $\psi$ parameters will pass through another module and the $c_1, c_2$, and $c_3$ will be calculated.

### Hardware vs. Software Results

In this section, we evaluate hardware implementation of the trimodal cluttering algorithm by assessing the latency of a single iteration and comparing its output accuracy against software results. For the software implementation, we use double floating point precision.

Each iteration of the Nelder-Mead simplex algorithm takes 63 cycles, which with a 4 ns clock frequency, will result in 252 ns in total. A MATLAB code of the algorithm takes $6\,ms$ for 100 iterations. Although MATLAB in not optimized for parallel software coding, and the software implementation could be much faster, the difference in the time and energy consumption is huge.

To check the accuracy of the hardware implementation, we provide the value of the optimal simplex and its function value after the first and the $100^{th}$ iteration and compare them with the MATLAB results. Based on the results illustrated in 3.5, the difference of the hardware and MATLAB results are less than 0.5%, which is in an acceptable range.

Table 3.5. Software and hardware results of the AE-K-3MD unit. Parameters are randomly initialized and are the same for both software and hardware.

| Parameter | Iteration 1 | | Iteration 100 | | |
|---|---|---|---|---|---|
| | Reference | SysGen | Reference | SysGen | Error (%) |
| $\varphi$ | -0.0756 | -0.0756 | 0.3558 | 0.3572 | 0.39 |
| $\phi$ | 1.8592 | 1.8591 | 1.354 | 1.3573 | 0.24 |
| $a_1$ | 0.2844 | 0.2844 | 1.0417 | 1.0398 | 0.18 |
| $a_2$ | 1.0442 | 1.0441 | 0.9116 | 0.9122 | 0.06 |
| $a_3$ | -1.2461 | -1.2460 | -0.9695 | -0.9708 | 0.13 |
| F | 2277.113 | 2277.640 | 45.327 | 45.512 | 0.4 |

### 3.4.9 AE-K-THRSH

After finding the optimal minimum parameter, we need to find the threshold such that $P_{fa}$ calculated by Eq. 2.17 is less than a constant. Therefore, the function for Nelder-Mead algorithm is calculated by Eq. 3.6.

$$error = |P_{fa} - P_{fa,min}|\,, \tag{3.6}$$

where $P_{fa,min}$ is the minimum false alarm ration. To find the optimal threshold we use the Nelder-Mead algorithm with only 2 vertices since the threshold is a scalar value ( $\eta \in R^1$). The algorithm starts with two random values for the threshold and is to find the optimal one with the minimum cdf value. In the calculation of the $P_{fa}$, we need to calculate $e^{\left(\frac{-\eta}{b_i}\right)}$

To do this calculation, we follow the following steps:

1. Calculating the $b_i = \rho_c \times a_i^2 + \rho_n$
2. Calculating the division, $\frac{-1}{b_i}$, by using a divide Xilinx IP core
3. Calculating the multiplication, $\eta \times \frac{-1}{b_i}$ by using a multiplier
4. Calculating the exponential part, $e^{\frac{-\eta}{b_i}}$, by using the Exponential Xilinx IP core. This IP core receive a floating point input and the result is also in floating point. Before feeding the input data to this IP, we type-cast the fixed-point input to a single floating point, and then we return the type of the result to a fixed-point number. This way, we only implement the *exp* function by using floating-point representation and therefore minimize the resource usage. Although, the Xilinx IP core for exponential function is not the fastest possible solution, its results are accurate. On the other hand, the latency of this unit does not affect much on the latency of the whole system, since the iterative algorithm usually converges in a few iterations. Therefore, we simply use the Xilinx IP core due to its high accuracy and also ease of its implementation.

The implementation of this module is very similar to AE-K-3MD, thus we do not provide more details.

**Bit allocation:** The output of this module is the threshold which will be used in the next module to detect the pixels by comparing the pixel values by the threshold. Therefore, the output, should have the same bit-width of the pixel values, unsigned 12-bit integers. But, the output of this module is calculated by the Nelder-Mead algorithm which needs enough bits for intermediate calculation in order to accurately calculate the vertices and their functions. we use $fi(0, 18, 10)$ to represent the vertices. 10 bits for the fraction to accurately update them, and 8 bits for the integer parts since based on the statistic, the threshold is usually less than 256.

### 3.4.10 AE-K-PIXDET

This module operates by comparing pixel values to a threshold computed by multiplying $\mu$ and $\eta$, which are outputs of the AE-K-STAT and SE-K-THRSH modules, respectively. If a pixel value is greater than this threshold, it is detected as a target. However, this module can only detect individual pixels and is not capable of grouping them into targets. The next module in the pipeline receives these detected pixels across the entire window and clusters them together as distinct objects.

### 3.4.11 AE-K-SPAT

The pixels that are detected are sent through a pipeline to a clustering module that groups together pixels that are close to each other, forming objects or ships. This module then measures various features of the clusters such as their location, size, and orientation, and combines them into a list of ship parameters. It is crucial that the clustering algorithm is implemented accurately as it can deal with multiple objects within a window. Additionally, it needs to be fully pipelined as the pixels within the window are streamed and come in one pixel per cycle.

The clustering module first computes the height and width location of each pixel. Incoming detected pixels are either added to an existing cluster or form a new cluster. A pixel is considered part of an existing cluster if it falls within the cluster's border. The border is defined as the location of the previous cluster member, plus or minus 2 pixels, and is updated with each new member. Therefore, new pixels are checked against all cluster borders and join an existing cluster if they fall within its border. The module then updates the border locations accordingly.

This module extract all the clusters inside a window, but only can detect 8 clusters at the same time due to both hardware constraint and theoretical optimization, since it is usually very unlikely for a window to have more than 8 objects. After finding the $9th$ cluster, the information of the existing and completely detected clusters will be sent to the next module in order to make space for the new clusters.

We then use the minimum and maximum location of each cluster in each direction to calculate the center and width of the cluster, which are know as $CentX, CentY, \Delta X$, and $\Delta Y$, respectively. Then, by using Eq. 3.7, we calculate the orientation of the cluster. Also, since the center location of the cluster is related to the window size, we further adjust them to the image size. This modification only contains a constant addition based on the location of the window in the whole image. After calculating the cluster orientation, the cluster report containing the center, width and orientation will be stored in a memory. When the clustering is done on a window, all the extracted clusters which are stored in the memory will be sent to another memory to join the extracted clusters from other engines. When all windows of an image are processed, the parameters of the extracted clusters are used to find the object's velocity. This is done in the next module.

$$Head_Angle = \frac{180}{\pi} \times \arctan\left(\frac{\Delta Y}{\Delta X}\right) \tag{3.7}$$

### 3.4.12 Velocity Measurement

After processing all windows of an image, it is time to estimate the radial velocity of each extracted target by using Eq. 3.8. For radial velocity estimation, both the *fore* and *aft* SLC images and the position of the target in those images are required. The goal is to measure the long-track interferometric (ATI) phase (i.e., the phase difference between the SLC images from two apertures related to a single point target) of the peak of the point target using the *fore* and *aft* SLC images (Eq. 3.9). However, the moving point target appears in several pixels in these SLC images, and we have the position of the center of the target, which is not necessarily the peak position. Furthermore, due to channel imbalance, the peak positions in the *fore* and *aft* SLC images may not be the same. Therefore, we need to find the peak positions in these images separately and then measure the ATI phase.

$$Radial\,Velocity \simeq \frac{\lambda v_e}{4\pi d}.\phi_{ATI} \tag{3.8}$$

$$\phi_{ATI} = \arctan\left(\frac{Im(fore_{max} \times aft^*_{max})}{Re(fore_{max} \times aft^*_{max})}\right) \tag{3.9}$$

To do this, we need to store these two images inside the FPGA. The real and imaginary parts of SLC images are stored in the memory as shown in Figure 3.6. We concatenate 4 elements of the image (Real and imaginary of *fore* and *aft*) at the same position, and store them in one address of the memory. We use dual port URAM based memory with 72 bits input width. Each element of the image is a 12 bit signed integer value.

On the other hand, we get the position of the target center in XY coordinates as shown in Fig. 3.7. To find the pixel with maximum power, we first extract a small image chip around the target center from the entire SLC image. This image chip is shown by a (2W+1) by (2W+1) window with blue color in Figure 3.7. Since the real and imaginary parts of

Fig. 3.6. Storing the SLC image in hardware

both *fore* and *aft* SCL images are vectorized first and then stored in the memory (as shown in Fig. 3.6), we need to find the memory addresses corresponding to the image chip in Fig. 3.7.

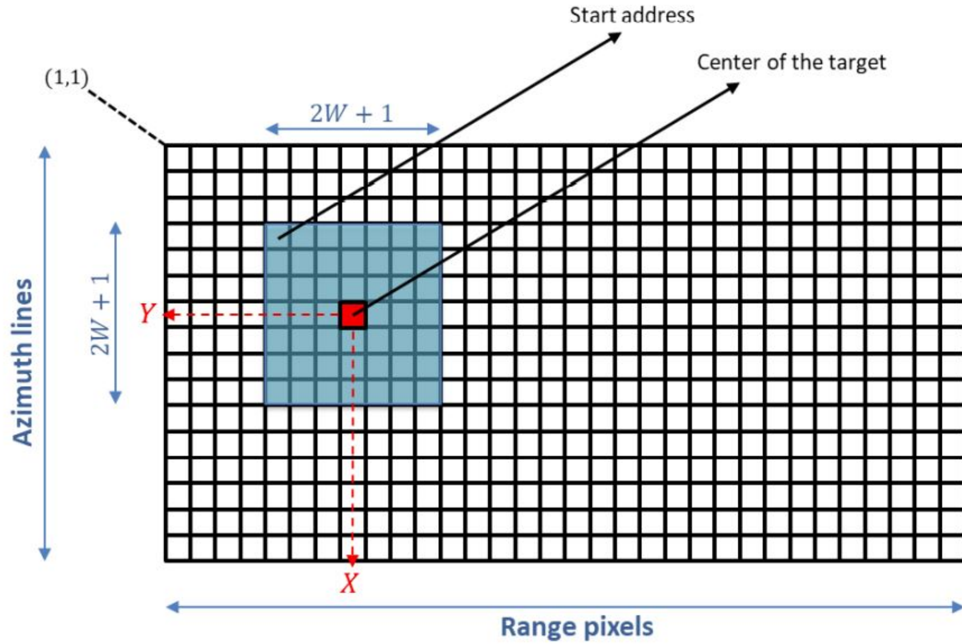



Fig. 3.7. XY coordinates of the target center in SCL image

Memory addresses for image chip extraction are calculated as shown in Figure 3.8. From Figure 3.6 and Figure 3.8, it can be simply seen that the memory address of the top left corner of the image chip is given by:

$$i_{start1} = M\left(X - W -\right) + Y - W, \tag{3.10}$$

where $M$ is number of rows (azimuth lines) in SLC image and $(X, Y)$ refers to the coordinates of the target center. The memory address of the bottom left corner of the image chip is given by:

$$i_{end1} = i_{start1} + 2Y - 1. \tag{3.11}$$

We repeat this process until we extract the bottom right corner of the image chip from the memory.

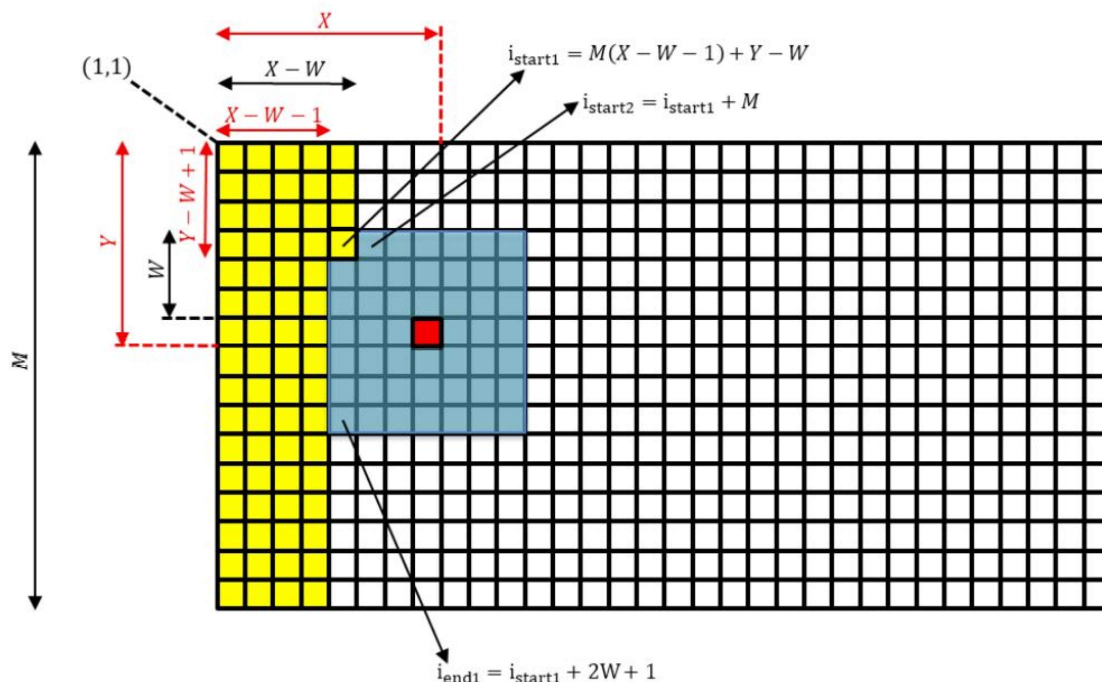



Fig. 3.8. Memory addresses for image chip extraction

After extracting the image chip around the target, we need to find the target peaks in the *fore* and *aft* images and measures the ATI phase followed by the radial velocity. To find the peak, we need to stream all the pixels in the image chip and calculate their absolute values or squared absolute values. To measure the ATI phase we use the real and imaginary parts. Therefore, we should register the pixel values which has the peak absolute value. To implement the ATI phase detection, we first implement the complex multiplier using a Xilinx Complex Multiplier Ip. Then, to find the angle of the result, we use a CORDIC IP in the translation mode instead of the ArcTan mode, since the translation mode is not only faster but also consumes fewer resources. Finally, by multiplying the angle by a constant, we measure the velocity of the extracted target. This velocity is then combined with the other parameters of the extracted target.

So far, we only extract targets which can be either ships or icebergs. However, the system needs to discriminate between these to objects. In next chapter, the ship/iceberg

discrimination algorithm which is a CNN classifier and its hardware implementation is explained in detail.

## 3.5 Evaluating Hardware Results

### 3.5.1 Experimental Platforms

The FPGA parts we are using for ASD implementation is a Xilinx Virtex Ultrascale+ VCU13P part, comprising 94.5Mb BRAM, 360Mb URAM, 12,288 DSPs and 1,728K logic cells (LUT). We use Xilinx Model Composer and System Generator 2020.2 to synthesize and simulate the hardware design on the selected FPGA. For the reference simulation, we use MATLAB R2020a and Intel Core i7-9750H CPU to evaluate the accuracy of the hardware design. The purpose of the reference simulation is mostly to assess the accuracy of the FPGA implementation, not its throughput or latency.

### 3.5.2 Experimental Results

To evaluate the implemented design, we compare the FPGA implementation with a software design for CPU. The simulation of the FPGA design takes a very long time, therefore, we only implement 1 engine for the ASD system and process only 1 window of the image in order to evaluate the latency and accuracy of the system. However, in the final design, we use 4 copies of the engine. To do a fair comparison, we use the same values for all the configuration parameters. For example, the maximum iteration of Nelder-Mead algorithm used in both AE-K-3MD and AE-K-THRSH units are set to 100 and 10, respectively. A single complex image is fed to the ASD system , and one window is selected for the ship detection. Fig. 3.9 illustrates the magnitude of the selected window with 4 visible targets inside this window.

Table 3.6 presents the locations of the detected targets from both software and hardware simulation. Although in the hardware design we used fixed-point representation and applied many approximation, the hardware design is able to detect all the targets. We have tested the detection accuracy by simulating more windows and reached a 100% accuracy for the detection. This accuracy is not related to the accuracy of the targets information, but means that the system can detect all the targets in the image, same functionality as the reference simulation. By comparing the location, size and head angle of the detected targets from hardware against software, we obtain a very high accuracy for calculating the location and size of target. Also, the accuracy of the head angle is in an acceptable range.

### 3.5.3 Resource Utilization

The resource utilization on FPGA including BRAM, URAM, DSP, and LUT is reported in Table 3.7. The latency (ms) is also reported. To calculate the latency of the hardware, we first assume the Nelder-Mead algorithm only terminates when it reaches the maximum

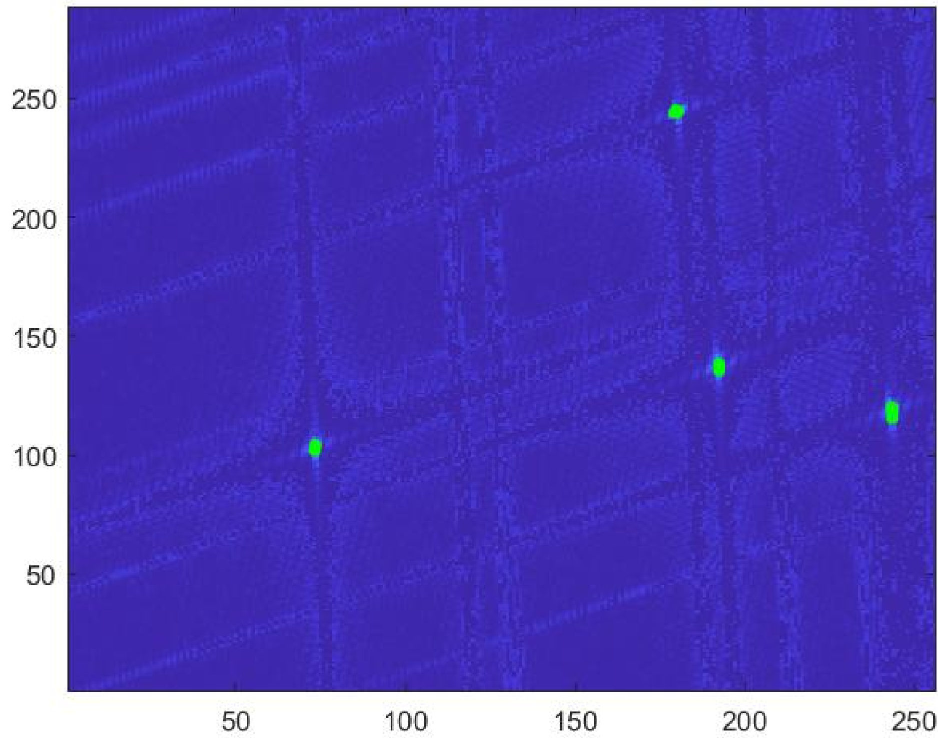



Fig. 3.9. An input window to the ASD engine for ship detection containing 4 targets.

Table 3.6. Ship report comparison among software and FPGA desgin.

| **Ship ID** | **SW** | | | **FPGA** | | |
|---|---|---|---|---|---|---|
| | Center | Size | Head Angle | Center | Size | Head Angle |
| Ship 1 | (73.5, 103) | (5, 2) | -5 | (73, 103) | (5, 2) | -10 |
| Ship 2 | (180, 244.5) | (4, 3) | -20 | (180, 243) | (4, 2) | -9 |
| Ship 3 | (192.5, 137) | (5, 2) | 0 | (192, 136) | (5, 3) | 5 |
| Ship 4 | (243.5, 118) | (7, 2) | 0 | (244, 118) | (8, 2) | 2 |

iteration. Then, we sum the latency of each module in the engine to calculate the whole latency of the engines. Since we have8 windows and 4 engines, we double the latency of the engine and add it to the latency of the other units. Finally, since the clock period is 4 ns, we calculate the latency as time. The resource utilization shows that the ASD system does not need much memory and high bandwidth since it has a relatively complex and iterative algorithm, not a memory hungry one. Also, it is worth mentioning that the low resource utilization is because we want to limit the resource utilization of this system to less than 25% since there will be other units such as SAR image formation algorithm in the same

device. Additionally, the designed implementation is quite customizable and we simply can increase the number of engines to improve both the throughput and resource utilization. We also compare the performance(throughput) among CPU, and FPGA using same input and configurations, as shown in Table 3.8. Our FPGA design achieves 27.07× improvement in throughput compared to CPU.

Table 3.7. Resource utilization, latency, and throughput of the hardware design.

| Device | LUT (k) | BRAM | URAM | DSP | Latency(ms) | Throughput (FPS) |
|---|---|---|---|---|---|---|
| XCVU13P | 296 (17.2)% | 45.5(1.7%) | 32(2.5%) | 2086(17%) | 2.4 | 414 |

Table 3.8. The performance comparison among CPU and FPGA.

| **Device** | **CPU** | **FPGA** |
|---|---|---|
| Manufacturer | Intel | Xilinx |
| Part | Intel(R) Core(TM) i7-9750H | XCVU13P |
| Clock | 2.6 GHz | 250 MHz |
| System Memory | 16 GB | 455 Mb |
| Latency (ms) | ∼80 | 2.4 |
| Throughput (FPS) | ∼12.5 | 416 |

# Chapter 4

# CNN-based Ship/Iceberg Discriminator

As mentioned in the previous chapter, the detected target could be either a ship or an iceberg because both ships and icebergs can appear similar in SAR images. However, misidentifying an iceberg as a ship, or vice versa, can have serious consequences.

In the case of ship detection, misidentifying an iceberg as a ship could lead to a false alarm, which may result in unnecessary search and rescue operations or cause unnecessary alarm to maritime authorities. On the other hand, failing to detect a ship in SAR images could lead to serious security issues or even accidents at sea, such as collisions or illegal activities. In the case of iceberg detection, misidentifying a ship as an iceberg could lead to dangerous navigation for ships, especially in areas with high traffic or where icebergs pose a significant threat to maritime operations. Failing to detect an iceberg in SAR images could also have severe consequences, such as shipwrecks, loss of lives, and environmental damage.

Therefore, accurate ship/iceberg discrimination in SAR is crucial for effective maritime surveillance and decision-making. SAR imagery is widely used in various applications such as search and rescue, oil spill response, maritime security, and environmental monitoring, and accurate ship/iceberg discrimination is critical for the success of these applications.

Discriminating between icebergs and ships in SAR imagery is a challenging task, but there are some methods that can be used to differentiate between icebergs and ships efficiently. These methods can be broadly categorized into two categories: traditional methods and machine learning methods.

**Traditional methods:** Traditional methods for ship/iceberg discrimination in SAR images include techniques such as polarimetric SAR, texture analysis, and contextual information [34, 13, 35]. These methods rely on mathematical models and signal processing techniques to extract features from the SAR images and discriminate between icebergs and ships based on these features. Traditional methods can be effective when there is a clear distinction between the scattering properties of icebergs and ships, but they may not be able to capture complex features and patterns in the SAR images.

**Deep learning methods:** Deep learning algorithms, such as convolutional neural networks (CNNs), have shown remarkable success in various computer vision tasks, including object detection and classification [36]. In SAR imagery, deep learning methods can be used to automatically learn features from the images and discriminate between icebergs and ships based on these features. One of the challenges in using deep learning for iceberg/ship discrimination is the limited availability of labeled data. However, data augmentation techniques, such as flipping, rotating, and scaling the SAR images, can be used to generate additional training data and improve the performance of the deep learning model.

Both traditional and deep learning methods have their strengths and weaknesses and can be used in combination to improve the accuracy of iceberg/ship discrimination in SAR imagery. Traditional methods can be useful for extracting specific features and contextual information, while deep learning methods can learn complex patterns and relationships between features in the SAR images.

Nowadays, there are some large datasets available for ship/iceberg detection with labeled images. One of the most widely used datasets for ship detection in SAR imagery is the Canadian RADARSAT-2 ship detection dataset, which consists of more than 30,000 SAR images labeled with ship annotations. The dataset includes images with different weather conditions and sea states and has been used in many research studies for ship detection and classification. Also, researchers have developed techniques for data augmentation and transfer learning, which can leverage existing labeled data to improve the performance of ship/iceberg discrimination models in SAR imagery.

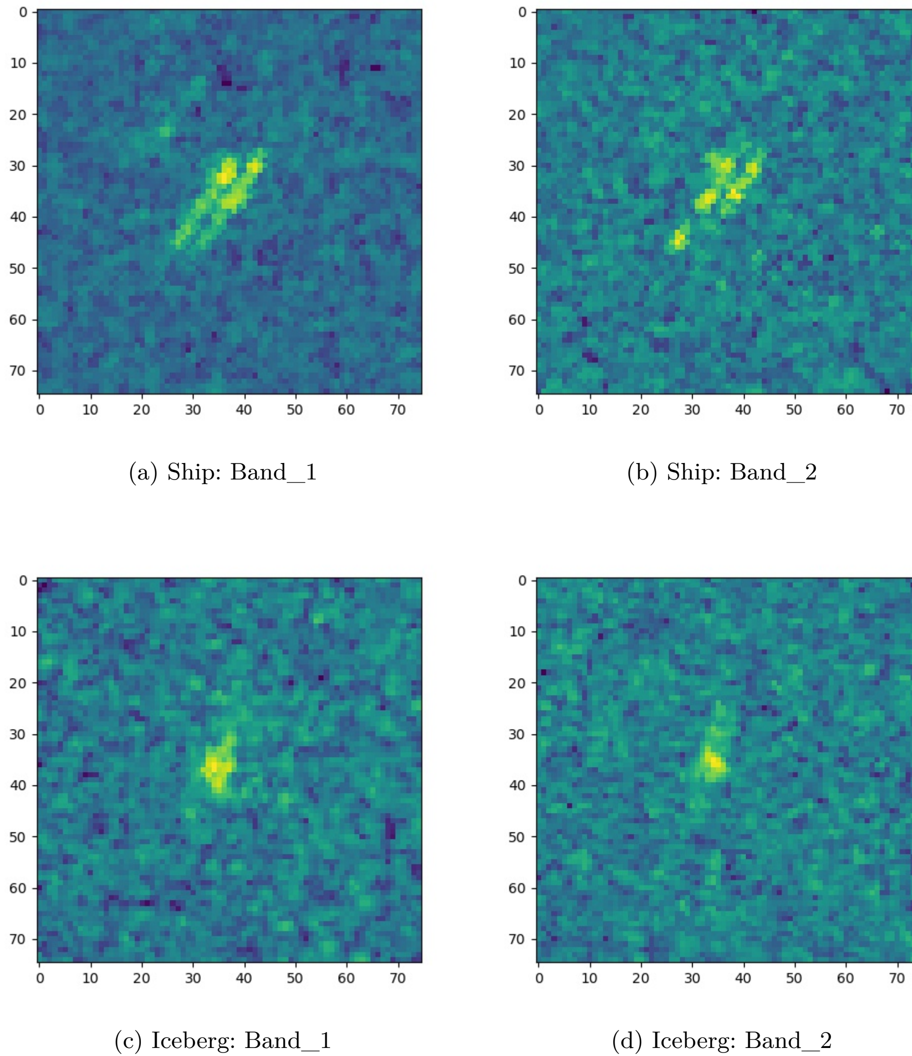
In this chapter, first, a CNN based ship/iceberg discriminator model suitable for hardware implementation is designed by training the network using a publicly available dataset containing images labeled as ships and iceberg. Then, we perform a design of CNN inference engine to run on Xilinx Zynq/UltraScale+ platforms. To make the inference model ready for FPGA implementation, a weight quantization is also performed. Finally, the performance of the hardware ready model will be evaluated against the original model.

## 4.1 Designing CNN-based Discriminator

### 4.1.1 Dataset

To design the model, we first need to examine the dataset. The dataset we use for training and testing is a publicly available dataset form [4]. This Kaggle competition is sponsored by Statoil and C-CORE companies. The dataset has 1604 complex images as input which each input contains only one object, a ship or an iceberg and the labels are provided by human experts and geographic knowledge on the target. All inputs are $75 \times 75$ complex images. The data of each image is flattened and stored as a list. Values are float numbers with unit being dB. Note that these values are not the normal non-negative integers in image files since they have physical meanings. Band_1 and Band_2 are signals characterized by radar

back-scatter produced from different polarizations at a particular incidence angle. The polarizations correspond to HH (transmit/receive horizontally) and HV (transmit horizontally and receive vertically). Along with the pixel data for each complex image, there is also a field named *inc_angle*, that is the incidence angle of which the image was taken. Note that there is missing data marked as "na", and those images with "na" incidence angles are all in the training data to prevent leakage. Fig. 4.1 shows both bands of a ship and an iceberg in the dataset. Top row shows a ship, and bottom row shows an iceberg.




(a) Ship: Band_1

(b) Ship: Band_2

(c) Iceberg: Band_1

(d) Iceberg: Band_2

Fig. 4.1. Sample images from Kaggle competition [4]. Top row: Ship. Bottom row: Iceberg

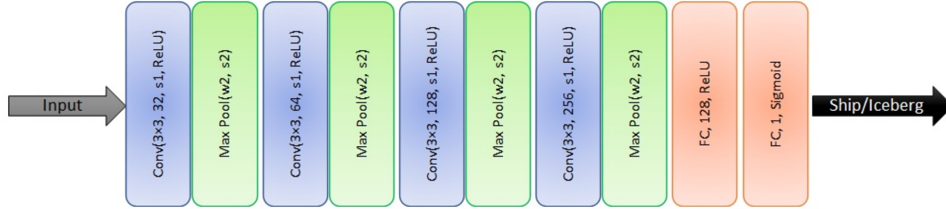



Fig. 4.2. Architecture of the simple CNN model.

### 4.1.2 CNN Model Architecture

Convolutional Neural Networks are specifically designed for image recognition and classification tasks. They are capable of automatically learning relevant features from the input images through a series of convolutional and pooling layers, followed by one or more fully connected layers that perform the classification task.

The ship/iceberg discriminator is a simple binary classifier and the training images relatively contains simple textures. The dataset also does not have many images, only a few hundred images which will be split to training and validation sets. On the other hand, we will implement the model on an FPGA to accelerate it and make the whole system a real-time one. Considering all the aforementioned points, the designed network should not be complex, only contains a few convolutional and pooling layers, followed by a few fully connected layer(s). Fully connected layers have many weights and therefore consume many memories to store the weights while, in general, they don't add much to the accuracy and performance. Fig. 4.2 illustrates the first designed model. We use ReLU for all the activation functions for its effectiveness and implementation simplicity. The last layer has only one node with Sigmoid activation function to discriminate between ships and icebergs. $Conv(3\times3, c, s1, ReLU)$ is a $3\times3$convolutional layer with $c$, the number of filters. $s1$ denotes the stride step. $Max\,Pool(w2, s2)$ is a maxpooling layer with window size $2 \times 2$ and stride $2 \times 2$. ($FC$, $n$, $ActFunc$) is a fully connected layer with $n$ number of nodes, and $ActFunc$ as activation function.

In addition to the previous model, we design a more complex and deeper model to possibly improve the model's performance. However, since we want to implement this model along with the system designed in Chapter 3, we cannot design a very complex model. On the other hand, since the training dataset does not have many images, a very complex model probably does not improve the accuracy considering the complexity it adds. Therefore, we only use residual blocks to design our CNN model.

Residual Network (ResNet) is a deep learning architecture that has achieved state-of-the-art results in image classification tasks. It was first introduced in [37]. The ResNet architecture is based on the idea of residual learning, which addresses the problem of vanishing gradients in very deep neural networks. The basic building block of ResNet is the

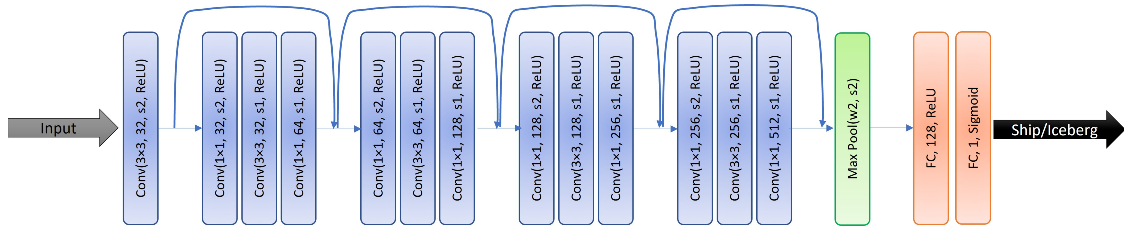



Fig. 4.3. Architecture of the ResNet-based CNN model.

Table 4.1. Performance of the designed model over the train, validation, and test datasets.

| Method | Train Acc. | Validation Acc. | Test Score | # of params | Size (MB) |
|---|---|---|---|---|---|
| CNN | 86.46 | 84.31 | 0.2922 | 305,985 | 1.16 |
| CNN/Data Aug. | 91.22 | 90.55 | 0.1866 | 305,985 | 1.16 |
| ResNet | 93.34 | 89.48 | 0.2420 | 1,484,449 | 4.24 |

residual block, which is composed of several convolutional layers followed by a shortcut connection that bypasses these layers. Fig. 4.3 illustrates the ResNet-based model. It consists of 4 residual blocks followed by a global max pooling and 2 fully connected layers, same as the previous model. Each residual block consists of three convolutional layers and a shortcut connection. The shortcut connection has a $1 \times 1$ convolution filter and bypasses the convolutional layers and adds the input of the block directly to the output of the block. All strides values are 1 except for the the first convolutional layer in each residual block which equals to 2.

### 4.1.3 Experimental Results

**Implementation and Training Setup**

The model was trained using ADAM optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The learning rate was initially set to 0.001. The training was done for 300 epochs with an early stopping criteria. PyTorch [38] was used to implement the model on a GeForce RTX 2060 GPU. The proposed model is optimized using a Binary Cross-Entropy loss function [39].

The dataset is divided to training and validation sets by the ration of 65 : 35, which results in 1043 and 561 images in training and validation set, respectively. Table 4.1 presents the model performance over the training and validation datasets. The last column represents the test score for the test dataset which provided by the Kaggle competition and calculated by the log loss between the predicted values and the ground truth. The models performance is better for the lower score.

### Data Augmentation

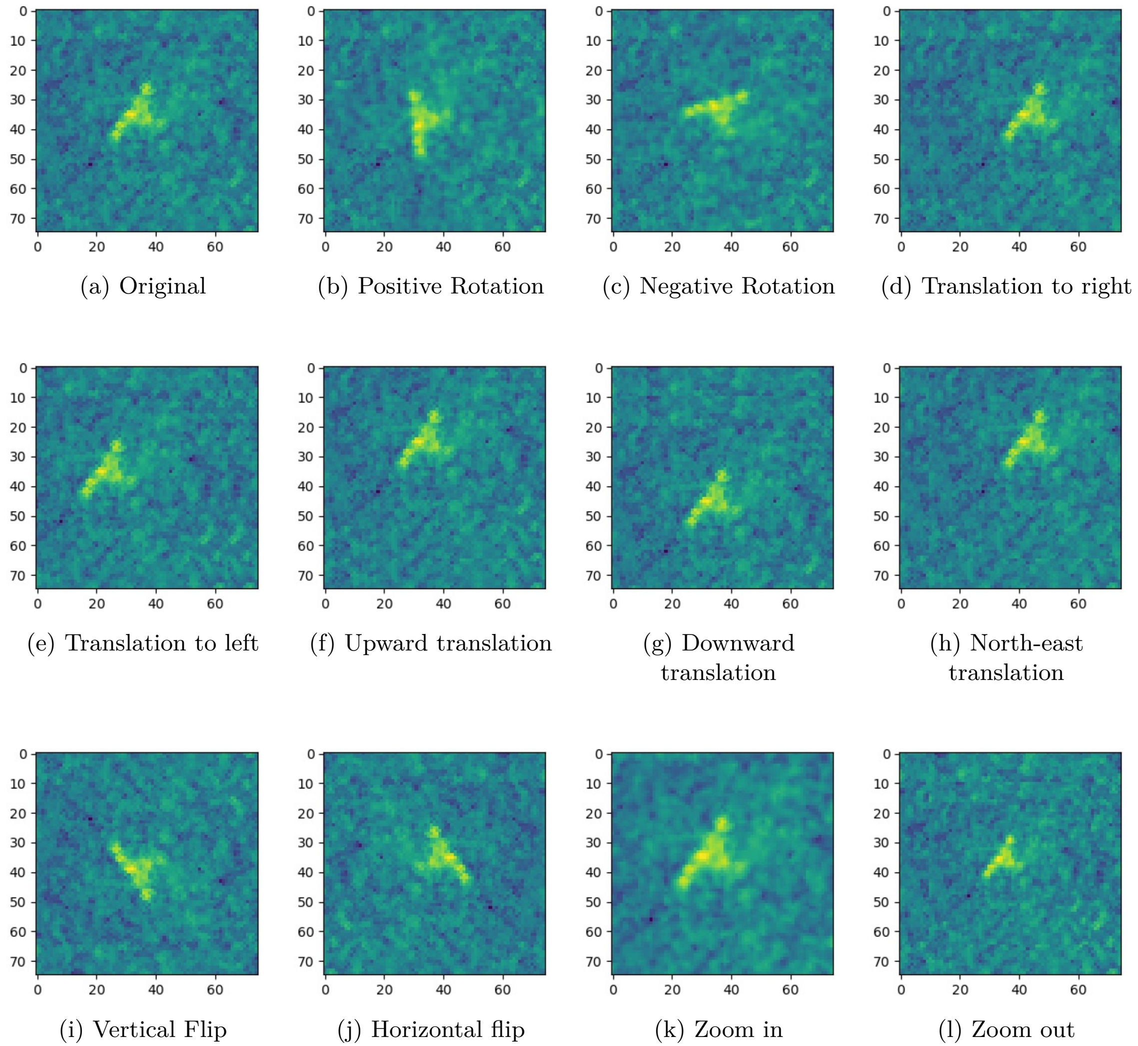
Based on the results of Table 4.1, the accuracy of the designed models is less than 85% which is not high enough for a practical ship/iceberg discriminator. Therefore, we need to improve the accuracy of the model. We can add more convolutional layers to the model to make the model more complex. But, since we are dealing with a binary classification and the input images mostly contains sea clutters, a more complex model will probably lead to overfitting. This claim has been tested by using residual blocks in the model architecture. Second row of Table 4.1 shows the accuracy of the model with residual blocks, which has almost the same accuracy as the base model. Therefore, since we are going to implement the model on an FPGA to get real-time results, it is better to use a simpler and shallower model to save both the memory and the computational resources.

To improve the accuracy of the model, we use data augmentation which is a technique used to artificially increase the size of the training dataset by applying different transformations to the existing data. These transformations can include rotations, translations, flips, and zooms. In the case of CNNs, data augmentation is particularly useful because CNNs are designed to be translation-invariant and can learn to recognize patterns regardless of their location in the input image. By applying these transformations, the model can learn to be more robust to changes in the input data and can perform better on the test data.

In total, we add 14 augmented data for each image in the training dataset, which makes the training set to have $14,602$ images. Fig. 4.4 shows a sample image from the training set along with its 9 augmented data. After increasing the training set, we train both model with the same training setup. Row 3 shows the accuracy result of the model. The ResNet-based model is more complex and in general, ResNet models usually have higher performance compared to the simple CNN model in classification task. However, in our case, the training set does not have enough unique samples, which results in almost the same accuracy for both models. Enriching the training set with more distinct images may increase the ResNet-based model accuracy, even higher than the simple CNN model.

## 4.2 FPGA Implementation of the CNN Model

The CNN model will be implemented on FPGA to join the automated ship detection module explained in the previous chapter. Extracted patches from ASD which contains either a ship or an iceberg will be fed as inputs to the CNN-based ship/iceberg discriminator to further accurately classify targets as ships or icebergs. To implement the CNN model on FPGA, the floating-point parameters and operations should be converted to fixed-point, both to minimize memory requirements as well as to increase the throughput of the computations. After quantization, the model will be accelerated and ready for FPGA implementation. In the following sections, the quantization and the acceleration methods are explained in details.




(a) Original (b) Positive Rotation (c) Negative Rotation (d) Translation to right

(e) Translation to left (f) Upward translation (g) Downward translation (h) North-east translation

(i) Vertical Flip (j) Horizontal flip (k) Zoom in (l) Zoom out

Fig. 4.4. A sample image and its augmented data.

### 4.2.1 CNN Quantization

There is a need to speed up CNN models to make them suitable for deployment on small devices suitable for online applications. There are various methods to make the CNN models suitable for FPGA implementation, including network quantization and network pruning [40]. Quantization refers to the process of converting the high precision values of weights and/or activations into low precision or even binary values, the network size will decrease significantly without further modification to the network architecture. With the current hardware support for low-precision computation, network quantization has brought attention to this procedure. Representing a network with low-precision weights and activations reduces memory for storing intermediate activations in the inference phase, resulting in a reduction in memory traffic and, thus, energy consumption [41].

Quantizing a CNN can be challenging, but it ultimately depends on the specific implementation and the level of precision required for the task at hand. However, quantization can also result in a loss of accuracy if not done carefully. One way to fill in the performance gap is to apply fine-tuning after network quantization [42]. These methods are known as quantization-aware fine-tuning/training, and usually results in lower performance degradation of the quantized network compared to when a post-training quantization is applied to network, which directly quantizes a network without any need for fine-tuning/retraining with the cost of the performance gap. The gap can be drastically high, especially when quantizing to very low-precision, so it may not be tolerated, especially in ship/iceberg discrimination. On the other hand, the former method can be computationally expensive and time-consuming, and more importantly needs the training dataset which in some real-world scenarios, the training dataset is no longer accessible due to either some security concern or the existence of the dataset itself. Fortunately, in our case, the training set is available and the training does not need to be online, meaning the time spent on the fine-tuning/training is trivial, which leads us to use the quantization-aware training method.

There are different methods for quantizing a CNN, such as uniform or non-uniform quantization, which can affect the trade-off between accuracy and efficiency. In this thesis, we use a uniform quantization method similar to [43], which quantize the weights and activations to integers. To perform this quantization, we use quantizer tool from Vitis AI. Vitis AI is a Software Development Kit (SDK) developed by Xilinx that can help with the quantization of CNN models. Vitis AI provides a set of tools and Application Programming Interface (API)s that can be used to optimize and deploy machine learning models on Xilinx FPGA and SoC devices. It also has the Deep-learning Processor Unit (DPU) which is a programmable engine optimized for deep neural networks. It is a group of parameterizable IP cores pre-implemented on the hardware with no place and route required. It is designed to accelerate the computing workloads of deep learning inference algorithms widely adopted in various computer vision applications, such as image/video classification, semantic segmentation, and object detection/tracking. The DPU is released with the Vitis AI specialized instruction set, thus facilitating the efficient implementation of deep learning networks [44].

To quantize the model using Vitis AI, we need a pretrained model and a labeled training set. The accuracy and loss of the pretrained model is the baseline for the quantization. The training setup is the same as the floating-point training. Table 4.2 presents the accuracy of the quantized models over the validation set for different bit width of weights and activations. Comparing the accuracy of the quantized model with different bit widths and considering the FPGA multiplier's input size, we choose 8 bits for quantizing both weights and activations to have the highest possible accuracy. Quantizing the models' weights will shrink the model size by the same factor the weights are quantized. Thus, the model size

Table 4.2. Accuracy of the quantized model with different quantization bit-width over validation data set. W/A represents bit-widths of weights (W) and activations (A).

| Model \ (W/A) | 32/32 | 8/8 | 4/8 | 4/4 | 2/8 |
|---|---|---|---|---|---|
| Simple CNN | 90.55 | 89.84 | 89.66 | 72.01 | 47.41 |
| ResNet-based CNN | 89.48 | 89.66 | 88.23 | 83.42 | 67.37 |

Table 4.3. DPU Resource usage for quantized CNN ship/Iceberg discriminator on ZCU104 device.

| Resource | LUT | BRAM | URAM | DSP |
|---|---|---|---|---|
| ZCU104 | 230400 | 312 | 96 | 1728 |
| DPUCZDX8G | 46,223 (21%) | 84 (23%) | 46 (47%) | 690(40%) |

is shrink by factors of ×4, ×8, and ×16, compared to the floating point model while the weights are quantized to 8, 4, and 2 bits, respectively.

The quantized model will further be compiled to be used in a DPU for FPGA implementation. The Vitis AI compiler receives a frozen quantized graph as as input, and maps the network model to a highly optimized DPU instruction sequence. This DPU can further integrated into a Vitis application project along with a MPSoC to build the overall system. Table 4.3 presents the resource usage of the compiled DPU for DPUCZDX8G which is the DPU designed for the Zynq® UltraScale+™ MPSoC. It is a configurable computation engine optimized for convolutional neural networks. The degree of parallelism utilized in the engine is a design parameter and can be selected according to the target device and application. The architecture is set to B4096, the highest possible parallelism, which means the peak performance of the DPU is 4096 operations/cycle.

# Chapter 5

# Conclusion and Future Work

In this thesis, an automated ship detection algorithm for SAR imagery in maritime surveillance along with its hardware implementation was studied and discussed.

In Chapter 2, we studied that an automated ship detection algorithm is based on comparing the pixel values with a predetermined threshold, which is determined based on a statistical model (also known as 3MD) of the measured SAR data. The 3MD automatically determines and updates the threshold based on magnitude of sea clutter. Furthermore, a multidimensional unconstrained minimization algorithm known as Nelder-Mead simplex method, for parameter estimation of the clutter model was studied and discussed in detail.

In Chapter 3, first, FPGA was proposed as the desired hardware for accelerating the ship detection model. Then, the FPGA implementation of the model was discussed in detail. Finally, it was discussed that the output of the ASD, the detected target, could either be a ship or an iceberg which further needs to be discriminated.

Finally, a CNN-based ship/iceberg discriminator model was proposed in Chapter 4. Although CNN model has shown remarkable success in various computer vision tasks, including object detection and classification, there is a need to accelerate them to make them suitable for deployment on small devices and online applications. Quantizing wights and activation was then introduced as a method to shrink the model size and make them suitable for FPGA implementation. To implement the quantized model on FPGA, Vitis AI was used to compile the model to a DPU. The peak performance of the DPU is 4096 operations/cycle.

## 5.1 Future Work

To accelerate the CNN model designed in Chapter 4, we only applied one quantization method to quantize and thus shrink the model. Although we used linear quantization-aware training method which usually has the minimum performance degradation compared to post-training quantization, we can apply different method to quantize the model to lower bit width while keeping the model's performance. In addition, we can apply weight pruning

to prune weights with zero values results in less size to save the hardware memory. There are various methods for weight pruning which needs to be performed on the model.

For FPGA implementation, we used Xilinx programmable engine, DPU, to compile the quantized model for FPGA implementation. Although these DPUs are highly optimized for deep neural networks, one can further accelerate the CNN model more efficiently and implement it by using Xilinx Vitis High-Level Synthesis (HLS).

# Bibliography


[1] J. C. Curlander and R. N. McDonough, *Synthetic Aperture Radar: Systems and Signal Processing.* Wiley, 1991.

[2] C. H. Gierull and I. Sikaneta, "A compound-plus-noise model for improved vessel detection in non-gaussian sar imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 3, pp. 1444–1453, 2018.

[3] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the nelder-mead simplex method in low dimensions," *SIAM Journal on optimization*, vol. 9, no. 1, pp. 112–147, 1998.

[4] "Statoil/c-core iceberg classifier challenge," https://www.kaggle.com/competitions/statoil-iceberg-classifier-challenge, accessed: 2022-09-30.

[5] I. G. Cumming and F. H. chee Wong, *Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation.* Artech House, 2005.

[6] T. M. Lillesand, R. W. Kiefer, and J. W. Chipman, *Remote sensing and image interpretation*, 7th ed. Wiley, 2015.

[7] C. Elachi, T. Bicknell, R. L. Jordan, and C. Wu, "Spaceborne synthetic-aperture imaging radars: Applications, techniques, and technology," *Proceedings of the IEEE*, vol. 70, no. 10, pp. 1174–1209, 1982.

[8] F. Liu and Y. Zhang, *Synthetic Aperture Radar (SAR) Image Formation and Signal Processing.* CRC Press, 2018.

[9] G. Krieger, A. Moreira, H. Fiedler, I. Hajnsek, M. Werner, M. Younis *et al.*, "Tandem-x: A satellite formation for high-resolution sar interferometry," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium (IGARSS'07).* IEEE, 2007, pp. 1917–1920.

[10] M. Xu, L. Chen, H. Shi, Z. Yang, J. Li, and T. Long, "Fpga-based implementation of ship detection for satellite on-board processing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 9733–9745, 2022.

[11] P. W. Vachon, C. Kabatoff, and R. Quinn, "Operational ship detection in canada using radarsat," in *2014 IEEE Geoscience and Remote Sensing Symposium*, 2014, pp. 998–1001.

[12] N. Kourti, I. Shepherd, G. Schwartz, and P. Pavlakis, "Integrating spaceborne sar imagery into operational systems for fisheries monitoring," *Canadian Journal of Remote Sensing*, vol. 27, no. 4, pp. 291–305, 2001.

[13] J. Deepakumara, P. Bobby, P. Mcguire, and D. Power, "Simulating scn and mssr modes of radarsat-2 for ship and iceberg discrimination," in *2013 IEEE Radar Conference (RadarCon13)*, 2013, pp. 1–6.

[14] J. Neyman and E. Pearson, "On the problem of the most efficient tests of statistical," *Philosophical Transactions of the Royal Society of London.*, pp. 289–337, 1933.

[15] M. Liao, C. Wang, Y. Wang, and L. Jiang, "Using sar images to detect ships from sea clutter," *IEEE Geoscience and Remote Sensing Letters*, vol. 5, no. 2, pp. 194–198, 2008.

[16] C. H. Gierull, "Numerical recipes to determine the performance of multi-channel gmti radars," *Defence R&D Canada: Ottawa, ON, Canada*, p. 28, 2011.

[17] S. Watts, "Radar detection prediction in k-distributed sea clutter and thermal noise," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-23, no. 1, pp. 40–45, 1987.

[18] D. Stein, "Computing k-clutter-plus-gauss distributions," in *Radar 97 (Conf. Publ. No. 449)*, 1997, pp. 194–198.

[19] D. J. Crisp, "A ship detection system for radarsat-2 dual-pol multi-look imagery implemented in the adss," in *2013 International Conference on Radar*, 2013, pp. 318–323.

[20] K. Ward, R. Tough, S. Watts *et al.*, "Sea clutter: Scattering, the k distribution and radar performance," IET, Tech. Rep., 2013.

[21] A. Mezache, M. Sahed, and T. Laroussi, "K-distribution parameters estimation based on the nelder-mead algorithm in presence of thermal noise," in *2009 International Conference on Advances in Computational Tools for Engineering Applications.* IEEE, 2009, pp. 553–558.

[22] A. Mezache, F. Soltani, M. Sahed, and I. Chalabi, "A model for non rayleigh sea clutter amplitudes using compound inverse gaussian distribution," in *2013 IEEE radar conference (RadarCon13).* IEEE, 2013, pp. 1–5.

[23] ——, "Model for non-rayleigh clutter amplitudes using compound inverse gaussian distribution: an experimental analysis," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 1, pp. 142–153, 2015.

[24] M. Farshchian and F. L. Posner, "The pareto distribution for low grazing angle and high resolution x-band sea clutter," in *2010 IEEE Radar Conference*, 2010, pp. 789–793.

[25] G. V. Weinberg, "Assessing pareto fit to high-resolution high-grazin-gangle sea clutter," *Electron. Lett*, vol. 47, no. 8, pp. 516–517, 2011.

[26] L. Rosenberg and S. Bocquet, "The pareto distribution for high grazing angle sea-clutter," in *2013 IEEE International Geoscience and Remote Sensing Symposium - IGARSS*, 2013, pp. 4209–4212.

[27] A. Fiche, S. Angelliaume, L. Rosenberg, and A. Khenchaf, "Analysis of x-band sar sea-clutter distributions at different grazing angles," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 8, pp. 4650–4660, 2015.

[28] K. O. Bowman and L. R. Shenton, "Estimator: Method of moments," in *Encyclopedia of statistical sciences.* Wiley, 1998, pp. 2092–2098.

[29] C. H. Gierull and I. C. Sikaneta, "Processing synthetic aperture radar images for ship detection," Jan. 19 2017, uS Patent App. 14/801,976.

[30] C. H. Gierull, "On the statistics of coherence estimators for textured clutter plus noise," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 5, pp. 679–683, 2017.

[31] P. Lombardo and M. Sciotti, "Segmentation-based technique for ship detection in sar images," *IEE Proceedings-Radar, Sonar and Navigation*, vol. 148, no. 3, pp. 147–159, 2001.

[32] C. H. Gierull and I. C. Sikaneta, "Improved sar vessel detection based on discrete texture," in *Proceedings of EUSAR 2016: 11th European Conference on Synthetic Aperture Radar*, 2016, pp. 1–4.

[33] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[34] C. Howell, J. Mills, D. Power, J. Youden, K. Dodge, C. Randell, S. Churchill, and D. Flett, "A multivariate approach to iceberg and ship classification in hh/hv asar data," in *2006 IEEE International Symposium on Geoscience and Remote Sensing.* IEEE, 2006, pp. 3583–3586.

[35] M. Denbina, M. J. Collins, and G. Atteia, "On the detection and discrimination of ships and icebergs using simulated dual-polarized radarsat constellation data," *Canadian Journal of Remote Sensing*, vol. 41, no. 5, pp. 363–379, 2015.

[36] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision.* Springer, 2016, pp. 525–542.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *Conference on Neural Information Processing Systems*, Long Beach, CA, USA, 4-9 December 2017, pp. 1–4.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Conference on Neural Information Processing Systems.* Vancouver, BC, CANADA: Curran Associates, Inc., 8-15 December 2019, pp. 8024–8035.

[40] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[41] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: energy-efficient acceleration of transformers using fpga," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 175–180.

[42] Q. Jin, L. Yang, and Z. Liao, "Towards efficient training for neural network quantization," *arXiv preprint arXiv:1912.10207*, 2019.

[43] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[44] "Vitis ai user guide (UG1414)," .