American University of Sharjah
Department of Computer Science and Engineering

Maritime Vessel Detection Using a Network of Marine Radars via a Simulation

This Project is submitted to the department of Computer Science and Engineering at the College of Engineering in partial fulfillment of the requirements for the degree of

Bachelor of Science Degree in
Computer Science and Engineering

24/11/2024

Table of Contents

Abstract

This project explores the development of a maritime vessel detection system using a network of marine radars simulated through Unity. The objective is to enhance maritime surveillance capabilities in the UAE by addressing limitations in traditional radar systems such as range and coverage constraints and adding an extra data layer for maritime surveillance. By integrating deep learning (DL) with a network of buoy-mounted radars, this project aims to provide un-manned, real-time maritime situational awareness. We propose a cost-effective, scalable solution that utilizes a simulated environment to train and test radars and marine vessels detection, which is a proof of concept that, if implemented in real-life, would improve national security and economic stability in maritime domains. Our trained model, CenterNet, achieved an F1-score of 0.938. Additionally, our network of radars sends the detected ships' locations to the database, which are then retrieved for visualization.

1. Introduction

Maritime security is a crucial aspect for global trade, sustainability and military power. As maritime activities continue to expand, the need for effective military surveillance and situational awareness becomes increasingly pronounced. Traditional maritime surveillance methods often rely on large, detached systems which may not provide a complete situational image of the country's maritime territory. Specifically, traditional radar systems are limited in range, coverage and mobility, leaving gaps in security frameworks. The UAE, given its strategic location along key maritime routes, has recognized the importance of advanced maritime surveillance systems to ensure the safety and security of its waters. The need for larger scale situational awareness over oceanic territory is imperative to ensure national security and economic stability.

1.1 Motivation

The need for our project comes from the current gaps in maritime surveillance. Despite its growing importance in marine security, conducting maritime surveillance is a challenge for authorities because of the vastness of the maritime space. This challenge is mitigated by existing surveillance technologies. First, patrol boats equipped with marine radars, which are used commonly in the UAE [1], are deployed to actively monitor a designated area. Second, satellites precisely capture images of the ocean. Third, some vessels are equipped with an automatic identification system (AIS), which sends the vessel information to an onshore receiver. Finally, aircrafts and drones are used for aerial surveillance, providing coverage of vast maritime areas. Each of these solutions work together to get a complete picture of the maritime territory. However, their main limitations include their poor cost effectiveness, lack of real-time tracking and limited coverage area. Therefore, our aim with this project is to fill these gaps and contribute to the existing maritime surveillance methods. This will be achieved by combining deep learning with a new marine radar system.

1.2 Project Description

Our project aims to develop a marine radar network system that strengthens maritime surveillance. The main requirements of this project is the development of a radar network with seamless data exchange between the radar nodes and the central on land database. Additionally, it involves the implementation of deep learning for real-time marine vessel detection, tracking, and visualization on a map. We will achieve this by building a simulation that implements the radars and vessels.

The main benefits of this project lie in its ability to significantly enhance maritime surveillance capabilities. The project enables authorities to monitor vessels in real-time, and detect suspicious activities, which allows for more proactive measures in marine security. Furthermore, it facilitates efficient management of marine resources, resulting in the strategic allocation of resources to prioritize critical areas.

1.3 Organization of the Report

Section 2 - Problem statement and design objectives: This section outlines the specific issue our project seeks to solve. It includes a precise description of the problem, our proposed solution and our design objectives. Furthermore, it describes any limitations or challenges that might impact the project.

Section 3 - Literature review: In this section we offer a summary of previous research and work related to our project.

Section 4 - System specification: In this section we detail the specifications of the system we intend to develop outlining the functional, nonfunctional, and pseudo requirements.

Section 5 - Cost estimate: In this section we mention the cost of our project.

Section 6 - Technical approach and design alternatives: In this section we discuss the technical approach and solutions, including detailed descriptions of alternative solutions to the problem and decision analysis to justify the chosen design approach. Additionally, we employ different diagrams to showcase our project's design.

Section 7 - Project management plan: In this section, we cover the management aspects of our project, detailing the breakdown of project phases, timelines, and responsibilities of team members through a Gantt chart.

Section 8 - Implementation: In this section we discuss the implementation details of our simulation, deep learning and visualization platform.

Section 9 - Validation, verification, and performance analysis plan: This section covers the strategies for assessing the quality of our final project output and presents a performance analysis plan outlining the evaluation process.

Section 10 - Project global, economic, societal impact: In this section we explore the global, economic, and societal impact of our project.

Section 11 - Standards: In this section we discuss the standards relevant to our technology and procedure.

Section 12 - Conclusion: This section marks the end of our report. We offer a summary of the main points discussed, highlight the achievements, and conclude with thoughts on potential future work or developments related to the project.

References: This section lists all the sources, literature, and references cited throughout the report.

2. Problem Statement and Design Objectives

2.1 Problem Statement and Proposed Solution

Problem Statement:

Maritime surveillance in the UAE relies on transponders installed in registered vessels, coast guard patrols and surveillance aircraft [1]. Although the existing approach is effective in tracking vessels, it has limitations in areas where coast guards are absent. This can cause some vessels to slip past radar detections and access unauthorized zones in the maritime territory. Moreover, the onboard radars used in coast guard patrols have a maximum radius of 10km (up to the horizon) [2]. Any vessel beyond this radius tends to remain undetected by the patrol, leading to potential gaps in security and defense. Additionally, it can be expensive to monitor the entire maritime space of the UAE, as a large number of coast guard patrol ships are needed due to the limited radar range.

Proposed Solution:

Our proposed solution involves building a lattice network of buoy-mounted radars to actively monitor the maritime space of the UAE. These radars, incorporated with a deep learning model, will detect and track vessels, transforming radar plan position indicator (PPI) images into map coordinates. The coordinates will then be received by a centralized onshore site for aggregation and visualization of the UAE's maritime space. Our goal is to develop an unmanned system, resulting in a cost effective approach. By utilizing a network of radars along with deep learning, we aim to enhance accuracy of vessel detection and reduce false positives (FP). Additionally, we plan to simulate the maritime space of the UAE using the Unity Engine.

Due to the scarcity of public radar datasets, we propose developing a simulation using the Unity Engine for synthetic data generation. This program will simulate ships and radar systems, allowing for the creation of diverse scenarios to produce a wide range of training data. Our approach will serve two purposes. First, it will support this project's deep learning solution by providing sufficient training data. Second, it will contribute to the academic community by offering a versatile tool for radar data synthesis and a public synthetic dataset for training existing systems. This initiative aims to address the data shortage in radar research while advancing both our project and the broader field of study.

2.2 Design Objectives

The project's design objectives are set to advance the current approaches to maritime vessel detection. They are critical to the project's success and its distinctive contribution to the field.

Simulation of the system: Build a representative simulation that implements a lattice network of radars. The program should accurately simulate vessels on the sea and mimic real world radar detection.

Dataset creation: Build our own simulated dataset of radar PPI images to test the deep learning model developed.

Deep learning model training: Train a deep learning model that is capable of accurately detecting vessels which can be used to track various types and sizes of vessels.

Real-world applicability: While using simulation for the prototype of the system, ensure the system's design and algorithms are viable for implementation with real-world marine radar equipment.

Visualization platform: Build a program that can gather the coordinates of the marine vessels relative to the radars and visualize the location of the vessels on a map.

Scalability: Design the radar network to be easily expandable, allowing for varying territory coverage.

2.3 Limitations

Maritime radar detection in the UAE may have various limitations that need to be taken under consideration as mentioned below.

Costs: Due to the high costs associated with using radars and conducting continuous real-world testing, our project will be built on a simulation that is close to the real world.

Accessing military-grade radar data: Accessing real data from military-grade radar for our project can be significantly difficult as the data may contain confidential information. The UAE based maritime authorities have to abide by security protocols and military law, restricting their ability to release data.

Detection of unmanned aerial vehicles (UAVs): The project will not be able to detect UAVs even though detecting them constitutes an essential aspect of maritime surveillance. UAVs are difficult to spot due to their small size and fast agility. Additionally, UAVs use composite material in their airframe which are poor reflectors of radar signals, making them hard to spot [3].

Detection of submarines: We do not intend to detect submarines in our project because radio signals generated by a radar cannot meaningfully penetrate into the surface of water [2]. Therefore, we restrict the project’s scope to detecting and tracking maritime vessels on the surface of water.

Maritime vessels information: The project will not identify the owners or origin of the vessels since radar technology alone is not able to extract this information. This level of detail would require the implementation of additional hardware and algorithms, which is outside the scope and purpose of the project.

3. Literature Review

In this section, we review the main works related to marine radars and surveillance. Our project's aim is enhancing maritime surveillance by building a network of marine radars and using deep learning for marine vessels detection. To achieve this, we review works related to the current technologies employed, deep learning models, and existing radar simulations. We conclude this section by highlighting the main differences between our proposed solution and the related works.

- **Section 1: Current Technologies**

3.1 Maritime Surveillance and Information Sharing Systems for Better Situational Awareness [4]

This paper focuses on several maritime surveillance methods that are currently employed within the European maritime domain. The functioning mechanism, efficacy and unique advantages of each method is examined in the context of maritime surveillance. Firstly, the principle of high frequency surface wave (HFSW) present in over the horizon (OTH) radars is discussed. The HFSW technology installed in these radars assist in detecting maritime targets beyond the line of sight and make them beneficial to execute surveillance activities. Secondly, the paper discusses satellite-based surveillance such as the CleanSeaNet solution offered by the European maritime safety agency (EMSA). This solution alongside synthetic aperture radar (SAR) imaging provides the authorities a clear visual of the maritime domain irrespective of the day-night cycle or fog conditions. The solution proves to be effective primarily in detecting instantaneous oil spills and vessels. Lastly, maritime patrol and surveillance aircrafts also known as MPA and MSA are examined. The former has the ability to operate for long periods over seas and coastal regions. They are equipped with radars that detect ship movements and forward looking infrared cameras (FLIR). However, MPA has high flight costs and requires several crews for continuous monitoring. MSA is used for emergency services such as search and rescue and for protecting coastal regions.

3.2 Safety–Security Analysis of Maritime Surveillance Systems in Critical Marine Areas [5]

The article discusses maritime surveillance and explores the different surveillance methods adopted. First, it examines coastal radars employed since the 70s, and recent advancements that have been made in radar systems particularly using OTH radars

and HFSW radars. These radars are known for advanced detection capabilities, which resulted in their adoption by the maritime authorities in many countries. Since these radars possess impressive detection capabilities, they can detect large maritime subjects with accuracy even at long distances. However, they are unable to detect small and fast speed vessels. Second, the authors study the sea borne surveillance method, which uses coast guard (CG) vessels. These vessels have built in radars and AIS systems. The CG vessels benefit from the ease of movement around the maritime domain, but they suffer from the inability to maintain continuous monitoring.

3.3 Improving AIS reliability [6]

This paper discusses the reliability of the AIS. The authors gathered a large amount of AIS data and based on their analysis of the data, they concluded that 20% of the AIS data is wrongly transmitted. More specifically, errors in AIS data are for a variety of reasons including errors with sensors, issues resulting from manually inputting data, lack of security, and the crew intentionally falsifying information. The authors highlight the fact that AIS is unreliable and suggest additional training and rigorous control measures as a solution to the problem.

3.4 Applications, Evolution and Challenges of Drones in Maritime Transport [7]

The paper discusses the use of drones in maritime transport, facilitated by advanced technologies such as artificial intelligence (AI) and internet of things (IoT). Furthermore, the authors provide an in-depth study of the details and features of different categories of drones such as fixed-wing drones and fluttering drones used for marine surveillance. Second, the paper explores the application of drones in maritime transport which are used for a number of purposes such as ship inspections, monitoring marine life, and conducting search and rescue operations. Additionally, the drones support well informed decision making by providing up to date information on maritime conditions for the authorities. Lastly, the authors examine the challenges of drones under maritime safety and surveillance. An existing challenge in drones is reliable transmission of data, particularly in adverse weather conditions and remote areas. This may require innovative approaches to enhance overall efficiency and security. Furthermore, there is also the need for introducing robust strategies for data processing and fusion to improve maritime surveillance.

3.5 Space-Based Global Maritime Surveillance. Part I: Satellite Technologies [8]

The paper outlines the evolution of sensor technologies over time by discussing the introduction of maritime radar sensors in the 50s to the development of AIS in the 90s. It emphasizes that relying solely on AIS data and ground based marine radars cannot provide effective coverage of the oceans especially in remote areas. Therefore, the paper argues that space-based sensor technologies such as SAR, multispectral and hyperspectral optical sensors, and global navigation satellite system reflectometry

(GNSS-R) must be used. It provides an in depth overview of these technologies, including their advantages and disadvantages in the context of maritime surveillance.

Current Technologies Conclusion

It is evident based on the gathered information from various research papers that a variety of surveillance methods are currently employed in the field of maritime surveillance. These methods include coast guard ships, drones, patrol planes, AIS and satellite based solutions. Radar systems such as OTH and HFSW are important for accurate detection of large marine subjects. However, they are less effective in detecting small and fast speed vessels. Satellite-based solutions such as CleanSeaNet and SAR images are effective in identifying oil spills and vessels irrespective of weather conditions, yet they are not able to provide continuous surveillance of the maritime domain. Furthermore, acknowledging the evolution of sensor technologies from radar sensors to AIS, maritime authorities solely cannot depend on them to get a comprehensive coverage of the oceans, particularly in isolated areas. Thus, space-based sensor technologies such as SAR, optical sensors and GNSS-R are necessary to enhance the current techniques of space-based maritime surveillance. Coast guard vessels equipped with AIS support seamless movement in the maritime environment but they face difficulties in conducting continuous surveillance by being ineffective due to data transmission issues. Drones have also shown great potential in the field of maritime surveillance, but they also face difficulties in providing real time data transmission. Lastly, patrol aircrafts have proven to be effective in operating for long periods over the vast maritime domain, however they require high costs and presence of crew for continuous surveillance.

To conclude, although there are a variety of approaches to carry out maritime surveillance, each has its own advantages and disadvantages. We look forward to contributing to and addressing the limitations of the current technologies used in marine surveillance with our proposed solution. The lattice network of radars will aid in active monitoring of the maritime space of the UAE. Through deep learning integration, we aim to enhance accuracy of vessel detection, resulting in fewer false alarm rates and false positives. Furthermore, our objective is to develop an unmanned cost effective approach that will reduce the need for expensive manned patrols, thus contributing to marine surveillance especially in areas where coast guards are absent. This will minimize the risk of security threats and unauthorized access of vessels in the maritime domain of the UAE. Ultimately, our goal is to contribute to, not substitute, the current technologies with our innovative solution, aimed to improve maritime surveillance and defense across the UAE’s maritime space.

- **Section 2: Simulation of Radar Detection**

3.6 Radar and sea clutter simulation with Unity 3D game engine [9]

This paper examines and implements a radar and sea clutter simulation via the Unity game engine. The implementation utilized ray tracing which emits rays and has a receiver to detect the rays reflecting off an object. These rays are individually tracked and recorded, providing a high fidelity simulation of radar data, mimicking the real world. A limitation in their implementation was the inability to trace over a large surface, due to computational power. Additionally, the implementation does not simulate wave related phenomena such as diffraction and imperfect reflection. However, the paper concludes that ray tracing in Unity is a viable approach in simulating radar.

3.7 Autoferry Gemini: a simulation platform for electromagnetic radiation sensors [10]

This paper outlines a framework built on the Unity game engine for simulating electromagnetic sensors (EM) for autonomous ships. Their approach used ray casting, which casts rays from a camera (sensor in this case) to detect objects. The rays then collide with objects and record their relative position. The framework then mathematically models the behavior of electromagnetic waves to provide a high-fidelity representation of the EM sensors. Their implementation can handle multiple sensors simultaneously, up to the limit that the computational resources allow. However, the authors conclude that using ray casting is a simplistic approach to modeling electromagnetic sensors like radar as it does not consider various wave properties of real-world EM waves.

3.8 Virtual generation of lidar data for autonomous vehicles [11]

This paper discusses a lidar simulation for autonomous vehicles. The approach uses ray casting and unity shaders to model the laser detection. They compared the results of their simulator with real lidar data, mimicking the scene and achieved a very high similarity. However, they did not use any mathematical modeling to calculate the signal received, making the model simplistic. The high similarity with the comparison case may mean that simulating short range lidar does not need accurate modeling of EM waves to give an accurate result.

3.9 Ray tracing for radio propagation modeling [12]

This paper discusses the fundamental concepts behind ray tracing for radio waves. The authors describe the different types of ray tracing algorithms and their suitability for modeling radio propagation, namely the Image method that calculates a ray reflection accurately but is inefficient with large numbers of reflection surfaces. The

shooting and bouncing ray (SBR) method traces every ray to determine if they arrive at a field point and can do so efficiently, particularly useful for radar simulation. The algorithm of interest is the Hybrid Image and SBR method which combines the simplicity of SBR while maintaining the accuracy provided by the image method. The paper also discusses different methods to computational efficiency, most importantly the uses of GPU acceleration for ray tracing.

3.10 Low-Fidelity Radar Implementation for Real-Time Ship Maneuvering Simulator with Unity3D [13]

This paper discusses a method for simulating radar detection using a low-fidelity model in the unity game engine. The method proposed is a low-fidelity method based on a mathematical model unlike methods such as ray tracing or ray casting. The method transforms a camera depth view image into a radar reading which can be plotted on a PPI image. The paper addresses the shortcomings of this method and attempts to combat this through the addition of noise and additional artifacts. Specifically, the paper implemented rain artifacts to simulate different weather conditions, random false echoes and random curved lines imitating interference. With all these additional artifacts, the resulting radar simulation is suitable for environments where performance and speed is necessary.

3.11 A Ray-tracing Algorithm for Signal-level Pulsed Radar Simulation [14]

The authors design and implement a signal-level pulsed radar simulation using ray tracing and the NVIDIA® OptiX™ engine. Their main focus was on addressing some issues present in previous works such as not accounting for phase shifts and refraction effects. Additionally, they compare their algorithm to the radar simulator known as phased array system toolbox (PAST). The authors implemented the ray tracing simulator (RTS) and conducted two simulation experiments. The first experiment had a single radar transmitting 16 pulses to a drone resulting in very similar signals received by the radar in both PAST and RTS. Furthermore, the authors note that PAST makes use of point model approximation, while RTS uses tessellated meshes composed of triangular primitives, which allows it to account for a material's type and volume. In the second experiment, the authors used two cubes located at varying distances from the radar instead of a singular drone. The result was both RTS and PAST having similar signals with minor differences due to noise. Finally, the authors conclude that ray tracing is a viable option for complex simulations with realistic targets.

Simulation of Radar Detection Conclusion

The task of simulating radar has been approached in different ways by researchers. From the literature available we gather 3 main approaches: ray tracing, ray casting and low-fidelity methods. Ray tracing proves to be the closest replication of real world radar, but is computationally expensive. Ray casting is slightly less

computationally demanding but does not capture details in wave properties like diffraction and imperfect reflection. Low-fidelity methods transform basic depth views into radar readings and are suitable for applications that rely on performance and speed. These methods highlight an important trade off between simulation accuracy and performance, with the most representative method being highly demanding and vice versa. For the project's purpose, we plan on using a combination of low-fidelity methods and ray casting, in favor of high performance. After comparing the methods, we found that the performance of ray tracing is unsuitable for the generation of a large dataset using multiple radars. Combining the approaches of both methods allow us to generate realistic radar data without compromising on performance.

- **Section 3: Deep Learning for Radar Target Detection**

3.12 Marine Target Detection Based on Improved Faster R-CNN for Navigation Radar PPI Images [15]

The objective of the study is to address the challenges associated with traditional target detection methods in marine radars images, particularly in the context of complex marine environments and low signal-to-noise ratios (SNRs). The paper revolves around the development of an enhanced region convolutional neural network (Faster R-CNN) algorithm designed specifically for detecting marine targets in navigation radar PPI images. They enhanced the algorithm by utilizing VGG16 and ResNet101 as shared convolutional networks, replacing ReLU with ELUs, incorporating region of interest (ROI) pooling, and optimizing the training parameters. The methodology employed includes the construction of a specialized navigation radar dataset, preprocessing of radar echo data, generation of PPI images, and training of the optimized Faster R-CNN model. Key findings confirm the enhanced detection accuracy and reliability of the proposed method compared to conventional approaches as shown in Table 3.1. The paper did not mention the drawbacks of their proposed model and did not make their generated dataset public.

Method	AP of marine targets	AP of ground clutter	mAP
Faster R-CNN	0.899	0.911	0.905
Proposed	**0.93**	**0.954**	**0.942**

Table 3.1: Average precision comparison between the authors' model and the traditional Faster R-CNN [15]

3.13 Maritime Target Detection Based on Radar Graph Data and GCN [16]

The study aims to enhance maritime target detection by proposing a novel method based on radar signal graph data and graph convolutional network (GCN). Utilizing a

synthetic dataset with simulated target signals and Gaussian background, the approach employs GCN for feature extraction and neighborhood aggregation to improve detection performance. The paper highlights a limitation with conventional target detection methods based on CNNs, where each signal sample is processed independently, thus failing to utilize the complete dataset's potential. Despite radar signal data not inherently having a graph structure, it often contains valuable information from adjacent times and locations, making a graph-based approach effective. The proposed model's performance evaluation metrics such as detection probability (Pd) at false alarm rates (Pfa) of 10^{-3} and 10^{-4}, along with ROC curves under different SNRs, demonstrate the method's superiority over traditional CNN-based approaches (specifically LeNet.), particularly at SNRs above 5 dB. Their results are shown in Table 3.2 (interestingly, their Pd is increasing when decreasing Pfa and they did not mention anything about that so most probably it is a typo). In addition, they did not make their simulated dataset available publicly.

SNR (dB)	Model	-5	-4	-2	0	2	4
Pd (Pfa = 10^{-3})	**GCN**	**0.182**	**0.447**	**0.503**	**0.551**	**0.690**	**0.768**
	CNN	0.107	0.327	0.336	0.360	0.576	0.622
Pd (Pfa = 10^{-4})	**GCN**	**0.301**	**0.559**	**0.644**	**0.785**	**0.746**	**0.806**
	CNN	0.223	0.421	0.401	0.571	0.598	0.694

Table 3.2: Detection performances comparison at different SNRs [16]

3.14 Multi-Dimensional Automatic Detection of Scanning Radar Images of Marine Targets Based on Radar-PPInet [17]

The paper introduces Radar-PPInet, an improved you only look once (YOLO) network designed for detecting marine targets in radar PPI images. It aims to overcome the limitations of traditional radar detection methods that struggle in complex sea clutter environments. The model incorporates advanced features such as CSPDarknet53, spatial pyramid pooling (SPP), path aggregation network (PANet), power non-maximum suppression (P-NMS), and multi-frame fusion, enhancing detection capabilities. The dataset comprises X-band marine radar PPI images from coastal and open sea environments. Performance metrics reveal that Radar-PPInet outperforms traditional methods like cell averaging constant false alarm rate (CA-CFAR) and Faster R-CNN, with improvements in detection probability by 15% and 10% respectively under specific false alarm rates as shown in Table 3.3. Additionally, the computational load is addressed, demonstrating that Radar-PPInet requires fewer parameters and computations compared to Faster R-CNN. The authors

stated that the dataset they used is public; however, we are unable to access it, possibly due to export control restrictions.

	CA-CFAR	Faster R-CNN	**Radar-PPInet**
Pd (Pfa = 10^{-3})	0.756	0.784	**0.896**
Pd (Pfa = 10^{-4})	0.584	0.647	**0.823**

Table 3.3: Pd comparison with different Pfa (coastal environment).

3.15 Sea Clutter Suppression for Radar PPI Images Based on SCS-GAN [18]

The study introduces a new technique called the sea clutter suppression generative adversarial networks (SCS-GAN) to effectively reduce sea clutter in radar PPI images. This helps improve target detection by cleaning up the radar data before analysis. Their methodology involves constructing multiple datasets, including simulated and real sea clutter under varying sea states, to train and test the SCS-GAN model. By utilizing residual networks and attention modules within the GAN framework, the SCS-GAN demonstrates improved clutter suppression capabilities over traditional algorithms. Key findings indicate that the SCS-GAN outperforms existing denoising and clutter suppression methods in terms of clutter removal speed, generalization ability, and preservation of marine target integrity. Results show that SCS-GAN results in larger clutter suppression ratio (CSR) and lower computational power (measured by the time taken for each image to declutter). The dataset used in this paper is private.

3.16 Maritime RTD in Sea clutter based on CNN with Dual Perspective Attention [19]

The authors aim to improve maritime radar target detection (RTD) between sea clutter, challenging the efficacy of traditional algorithms such as CFAR. Their method first encodes the radar echo in high-dimensional space and then extracts the correlation features from the global and local perspectives through the attention mechanism. Data for this study were gathered using an X-band pulse-compression radar deployed on the coast of Hainan, China, creating a diverse dataset that includes various maritime scenarios. Performance evaluations show that the proposed method outperforms traditional and some recent deep learning-based methods, achieving a Pd of 0.9359 with a Pf of 10^{-3}. Future work will focus on incorporating more diverse offshore target data to enhance the model's robustness and applicability in real-world conditions. The authors mentioned that they will make the dataset public for research but we could not find it.

3.17 Marine target detection based on Marine-Faster R-CNN for navigation radar PPI images [20]

The paper aims to enhance the detection accuracy and reduce false alarms in marine environments by proposing an optimized Faster R-CNN model, named Marine-Faster R-CNN. This model incorporates a new backbone network, feature fusion network (FFNet), adjustments in anchor size, and the implementation of power non-maximum suppression (P-NMS) to improve detection of small-scale targets and handle multiple detections effectively. The dataset, created using Japan Radio Co., Ltd. (JRC) navigation radar under varying conditions, is private and tailored for assessing the model's performance, which demonstrates high precision (0.9902) and recall (0.9365) and a low false alarm rate (0.0098). The findings indicate that Marine-Faster R-CNN outperforms standard Faster R-CNN and traditional CFAR methods, as shown in Table 3.4, especially in complex sea clutter scenarios, although its complexity may impact real-time application efficiency.

	CA-CFAR	Dual-parameter CFAR	**Marine-Faster R-CNN**
Pd (Pfa = 10^{-2})	0.6251	0.6407	**0.9371**
Pd (Pfa = 10^{-3})	0.3924	0.4288	**0.9052**

Table 3.4: Pd comparison with different Pfa.

3.18 Marine Targets Detection for Scanning Radar Images Based on Radar-YOLONet [21]

The paper explores the development of Radar-YOLONet, a deep learning model adapted from YOLO v4, designed to enhance marine target detection in radar PPI images under complex environmental conditions like oceans, land, and islands, for real-time processing. Utilizing a private dataset comprising radar PPI images from X-band solid-state navigation radars, categorized into targets, coastal land, and sea clutter, with precise annotations from the AIS, the model incorporates components such as CSPDarknet53, spatial pyramid pooling (SPP), path aggregation network (PANet), and power non-maximum suppression (P-NMS). Performance evaluation revealed superior metrics compared to traditional methods as shown in Table 3.5.

	2D CA-CFAR	Faster R-CNN	**Radar-YOLONet**
Pd (Pfa = 10^{-3})	0.756	0.784	**0.833**
Pd (Pfa = 10^{-4})	0.584	0.647	**0.765**

Table 3.5: Pd comparison with different Pfa (Average results of multiple images).

3.19 Radar Maritime Target Detection via Spatial–Temporal Feature Attention GCN [22]

The research focuses on enhancing radar maritime target detection through a new method based on the spatial-temporal feature attention GCN (STFA-GCN), aimed at identifying targets within complex sea clutter environments using radar signals. Utilizing real measured and simulated radar data, the study proposes a novel form of graph data representation to demonstrate spatial-temporal features of radar signals and reduce the calculation burden on clutter signal via non-Euclidean data. The proposed STFA-GCN method extracts spatial-temporal features of multiframe scanning radar signals, distinguishing between targets and high-amplitude clutter. Additionally, a graph data structure is introduced to effectively convey the spatial-temporal feature of radar signals, supporting feature extraction in graph neural networks. The STFA-GCN method achieves superior detection performance compared to the three-frame accumulation CA-CFAR as shown in Table 3.6. The dataset used in this paper is private.

	CA-CFAR	**STFA-GCN**
Pd	0.839	**0.917**
Pf	1.65×10^{-4}	1.26×10^{-4}

Table 3.6: Pd comparison with different Pfa.

Deep Learning Conclusion

In reviewing the literature, it becomes evident that traditional methods fall short in effectively addressing the complications of maritime RTD among challenges like sea clutter and noise. Jiang et al. [23] discuss the limitations of conventional signal processing techniques such as CFAR and its variants. These methods, while aiming to maintain a constant false alarm rate by dynamically adjusting radar detection thresholds, often lack generality and are computationally expensive. Moreover, they rely on known statistical models for targets and environments, which are frequently unavailable in complex scenarios. As a result, the efficacy of traditional statistical methods diminishes in such environments, making the development of data-driven approaches inevitable.

The potential of deep learning in addressing these challenges is promising. Neural networks, with their inherent learning capabilities, offer a viable solution. Indeed, RTD can be framed as a pattern recognition problem, aligning well with the capabilities of artificial neural networks (ANNs) [23]. Particularly, deep neural networks (DNNs), with their ability to learn powerful generalizations and abstractions

through multiple layers, outperform traditional machine learning methods. Notably, while shallow neural networks require manual feature extraction to alleviate computational burden, DNNs streamline this process, enhancing efficiency and accuracy in RTD tasks.

One of the difficulties in applying deep learning-based methods to RTD is the lack of publicly available labeled data. Jiang et al. [23] concluded that most research teams construct and label radar datasets by simulation methods for deep RTD, besides SAR images, as the actual radar data was hard to obtain. Many image types, such as SAR, inverse SAR (ISAR), PPI, range-doppler, etc., are used in marine surveillance after conversion from raw data collected from marine radars. Throughout the literature, we found that researchers work on different datasets with different images types. As a result, finding a public dataset specifically for radar PPI images (that we intend to use) is almost impossible. Actually, we could not access any of the datasets that were discussed in the literature. That is why we are planning to simulate our own dataset.

After simulating the dataset, our next step involves implementing well-established deep learning models commonly utilized in the literature, including CNNs, GCNs, and YOLO architectures for marine vessel detection. Given the variability in datasets and collection conditions across studies, we prioritize empirical validation over relying solely on literature-reported results. Consequently, we will select the model with the highest accuracy based on its performance on our simulated dataset, ensuring robustness and applicability in real-world scenarios.

4. System Specification

4.1 Functional Requirements

Simulation System

FR1. The user shall be able to input data into the simulation system.
FR1.1. The user shall be able to input radar locations into the simulation system.
FR1.2. The user shall be able to input the radars' respective operating ranges for ship detection into the simulation system.
FR1.3. The user shall be able to generate a scenario with ship positions, paths and sizes..
FR1.4. The user shall be able to modify the weather condition.
FR1.5. The user shall be able to modify the wave condition.
FR2. The user shall be able to generate multiple new simulation scenarios at once.
FR3. The user shall be able to load a simulation scenario.
FR4. The user shall be able to start the simulation.
FR5. The user shall be able to pause the simulation.
FR6. The user shall be able to end a scenario or the simulation.
FR7. The user shall be able to modify the speed at which the simulation runs.
FR8. The user shall be able to view the log of all the simulation events.
FR9. The user shall be able to run all generated scenarios and save a dataset of PPI images.

Radar Onboard Software

FR10. The radar onboard software shall be able to receive information from its corresponding radar.
FR10.1 The radar onboard software shall be able to receive PPI data from the simulation system.
FR10.2 The radar onboard software shall be able to receive the radar's location from the simulation system.
FR11. The radar onboard software shall store the vessel locations generated by the deep learning model in the database.

Visualization Platform

FR12. The user shall be able to input data regarding map parameters such as scale and orientation in the visualization platform.
FR13. The user shall be able to refresh the map produced by the visualization platform to update the vessel locations.

4.2 Nonfunctional Requirements

NFR1. The simulation software shall be deterministic, ensuring that the same inputs produce the exact same outputs.

NFR2. The simulation system shall run simultaneously with the radar onboard systems.
NFR3. The simulation system shall simulate vessel movement and radar detection simultaneously.
NFR4. The simulation system shall run under various virtual scenarios with weather conditions such as different rain variations without errors.
NFR5. The simulation system shall validate the user input to avoid damaging configurations.
NFR6. The proposed system shall allow the expansion of the radar network, enabling the addition of more radar nodes to the existing system architecture.
NFR7. The radar onboard software and visualization platform shall continue to operate in the event of a single node failure.
NFR8. The radar onboard software shall detect marine vessels with F1-score above 0.9 under various weather conditions.
NFR10. The visualization platform shall visualize the latest vessel locations.
NFR11. The visualization platform shall be accessible by the user via a web application.
NFR12. The visualization platform shall automatically read the vessel information from the database every 10 seconds.

4.3 Use Case Diagram

Our use case diagram shows the interaction between the user and each specific subsystem. As shown in Fig. 4.1, the user interacts with the system in various ways while the simulation system is not running such as inputting data or starting the simulation. When the simulation system is running, the user is only able to pause or speed it up. Once complete, the simulation system sends the generated PPI images and radar locations to the radar onboard software for processing as shown in Fig. 4.2. Additionally, the database stores the classified vessel locations. Finally, Fig. 4.3 depicts the user interaction with the visualization platform, which will receive the information from the database and show it to the user. The user will be able to modify some parameters such as the scale and orientation of the map.

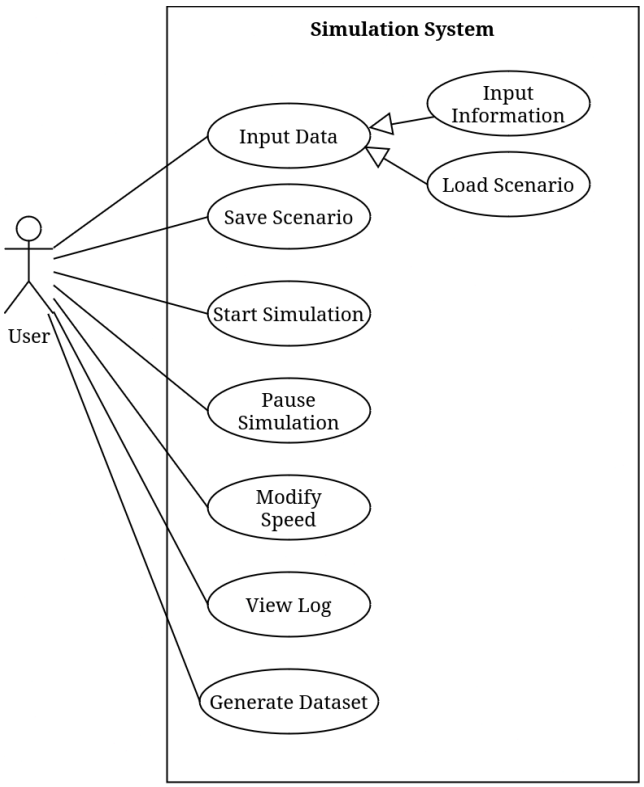

Fig. 4.1: Use case diagram of the simulation system.

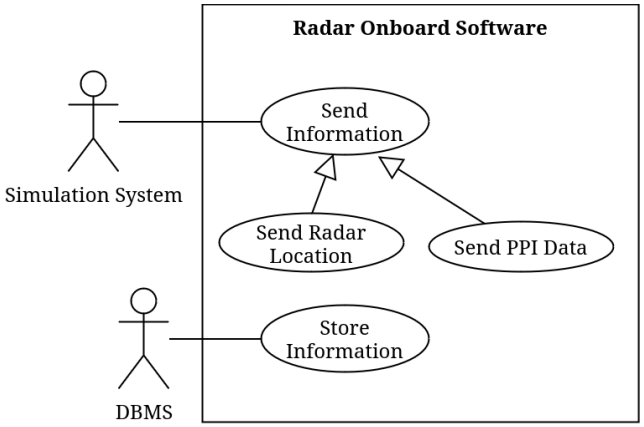

Fig. 4.2: Use case diagram of the radar onboard software.

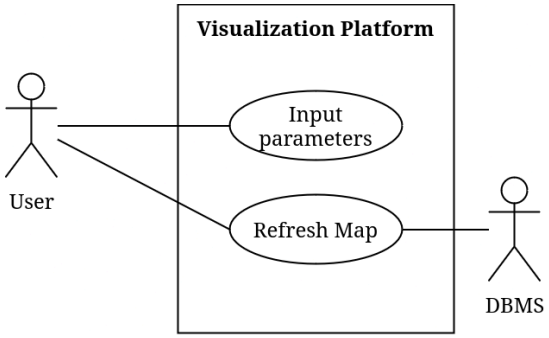

Fig. 4.3: Use case diagram of the visualization platform.

5. Cost Estimate

For most of the project there are no costs associated as this is a simulation based project. More specifically, the software we use will be free of charge. Additionally, we will use our own computers and AUS lab computers to develop and test the simulation, and generate a large dataset.

6. Technical Approach and Design Alternatives

6.1 Problem Statement and Proposed Solution

Although there are multiple methods used in maritime surveillance, patrol boats remain as a common choice due to their effectiveness in tracking vessels. However, this approach has its limitations when the patrol boats are absent in an area. To mitigate this problem and reduce costs, we propose building a lattice network of buoy-mounted radars. These radars will use DL to transform their PPI images into map coordinates of detected vessels, allowing us to visualize vessel movement in the maritime space of the UAE. Due to the cost of radars, we propose to build a simulation of the system, allowing us to test the system and generate datasets to train the DL model.

6.2 Design of the Solution

6.2.1 Block Diagram

This block diagram shows the various subsystems of our project. As shown in Fig. 6.1, The user begins the simulation by inputting the simulation settings, radar settings, simulation scenario, and radar locations. The simulation system creates the needed radar and vessels according to the settings and scenario provided and stores the radar locations in the database. The simulated radar will detect the simulated vessels and export its output as a PPI image, which is communicated to the onboard software and stored in the database.. The radar onboard software detects the vessels in the PPI images, converts the vessel locations from the relative position of the radar to the absolute position on a map and stores the vessel locations in the database. These vessel locations are then sent to the visualization platform for plotting onto a map, where the user can input specific map parameters such as scale and orientation.

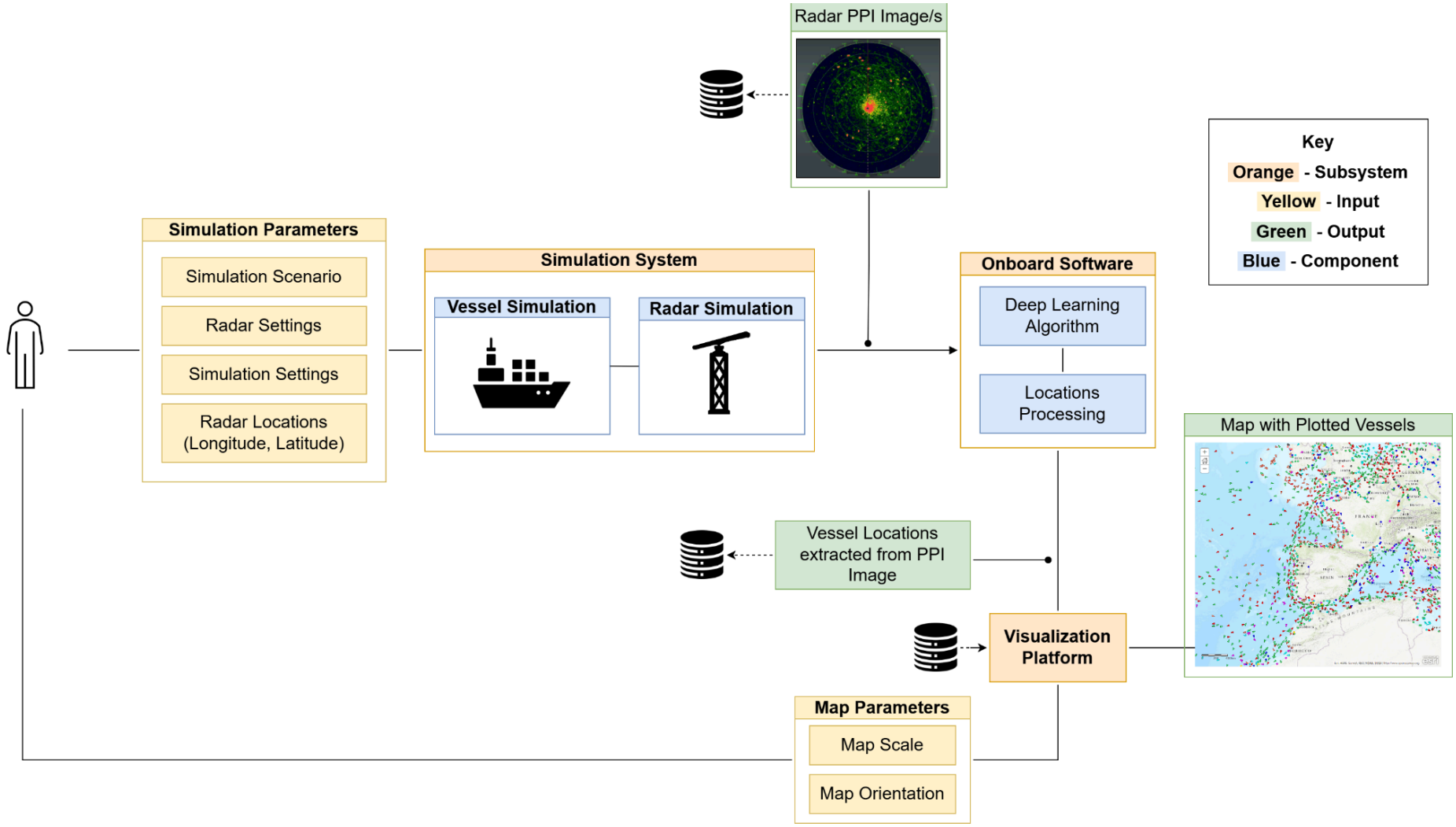

Fig. 6.1: Block diagram of the proposed system.

6.2.2 Workflow Diagram

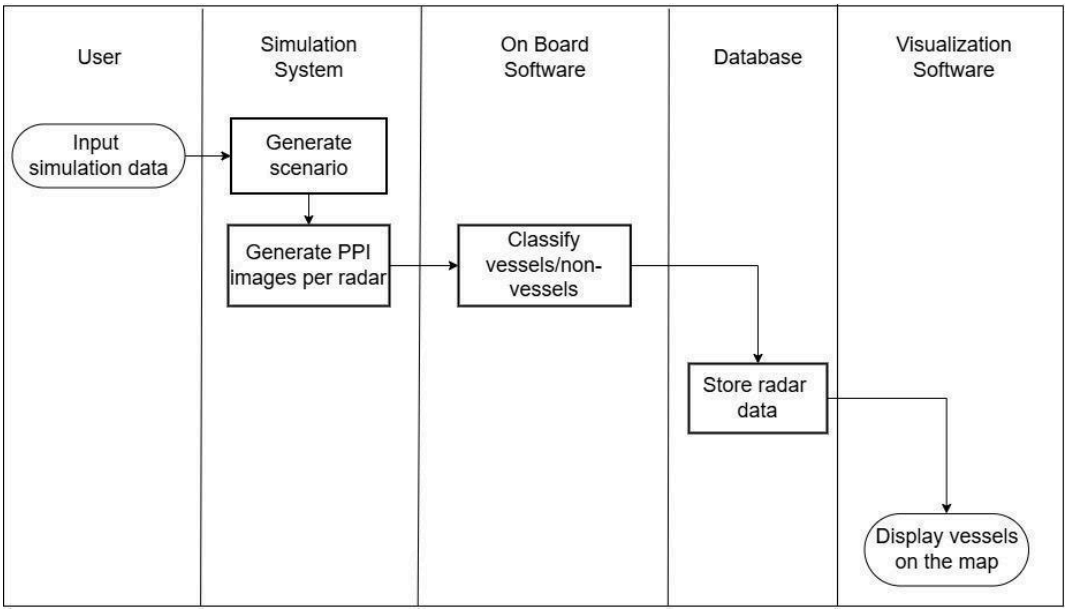

Fig. 6.2: Workflow diagram of the proposed system.

The workflow diagram provides an overview of the subsystems exchange of data. As shown in Fig. 6.2, our system begins with the user inputting simulation related information such as the number of ships and radar parameters. Once this data is inputted, the simulation runs the scenario and generates PPI images. This information is used as the input of the deep learning model on the on board software. The DL model classifies the vessels and sends the data for storage in the database. Finally, the visualization platform displays the vessel locations on the map.

6.2.3 Flow-Charts and/or State Diagrams

The flow-chart diagram in Fig. 6.3 illustrates the sequence of user-system interaction in our project. The process begins when the user starts the simulation system and proceeds to set up the simulation parameters, including radar locations, operating ranges, and simulated ship details. After validating the inputs the user runs the simulation system. It first sends the simulated radars' locations to the database and then the simulated radars detect marine vessels within their operating ranges, and generate PPI images. The PPI images are streamed to the radar onboard software for vessel/non-vessel classification through a deep learning model to allow an enhanced unmanned processing of the PPI images. After achieving that, the radar onboard system generates vessels' locations and sends them to the database. Finally, the user runs the visualization platform and sets up the parameters such as the map's scale and orientation. After validating the inputs, the visualization platform retrieves the vessels' locations and displays them on a map that shows the marine traffic in relation to the maritime context.

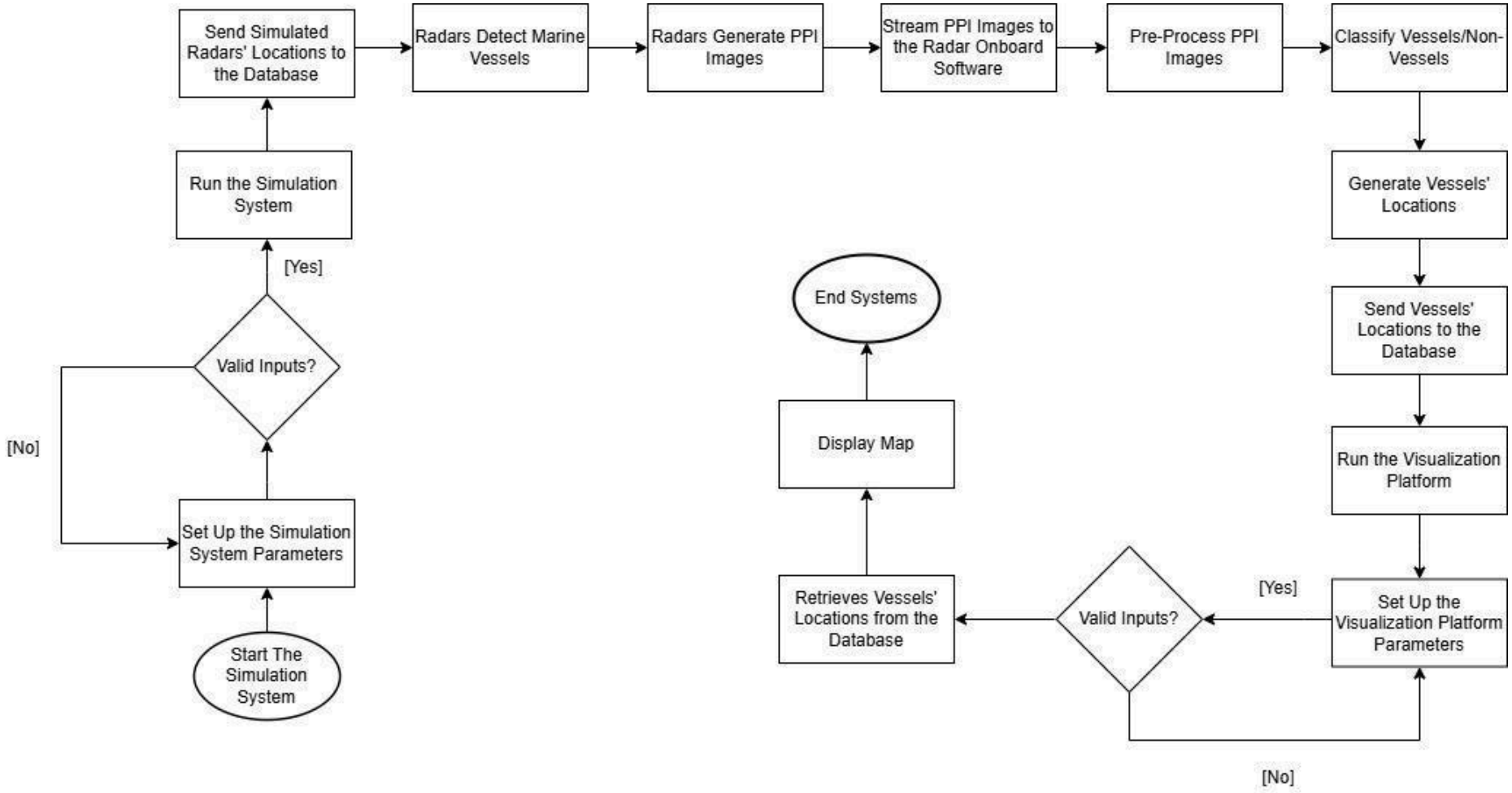

Fig. 6.3: Flow chart of the proposed system.

6.2.4 Database Diagram

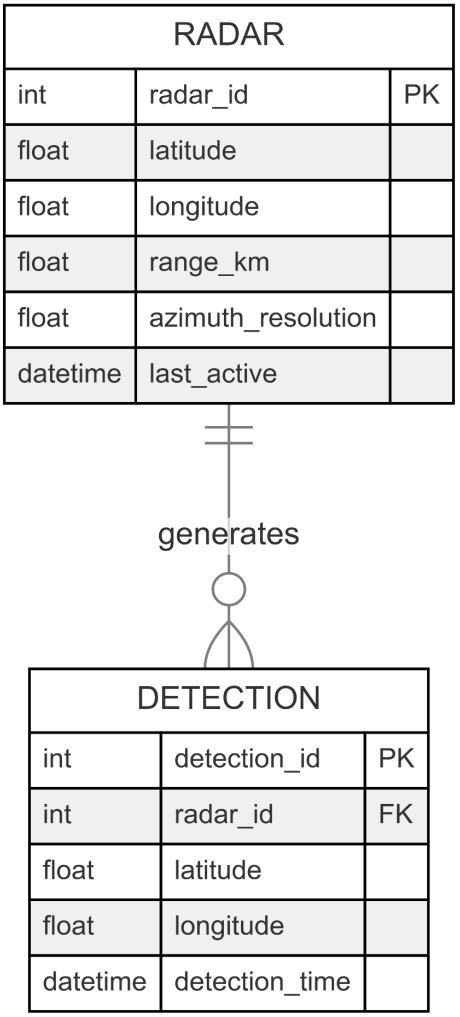

Fig. 6.4. EER Diagram.

As shown in Fig. 6.4, the database consists of a `‘radar’` entity, with each instance being uniquely identified by a `'radar_id'`. Associated with each `Radar` instance are longitude and latitude attributes, indicating its geographic location. The `radar` entity has a one-to-many relationship with a `‘detection’` entity, where each `radar` instance can generate multiple `detection` instances. The `detection` entity includes the latitude and longitude, indicating the location of the ship. Each `detection` instance is also tagged with a `detection_time`, providing the exact time the data was captured. The `'radar_id'` acts as both a primary key within the `radar` entity and a foreign key in the `detection` entity, linking each piece of data back to the specific radar that generated it.

6.2.5 Class Diagram

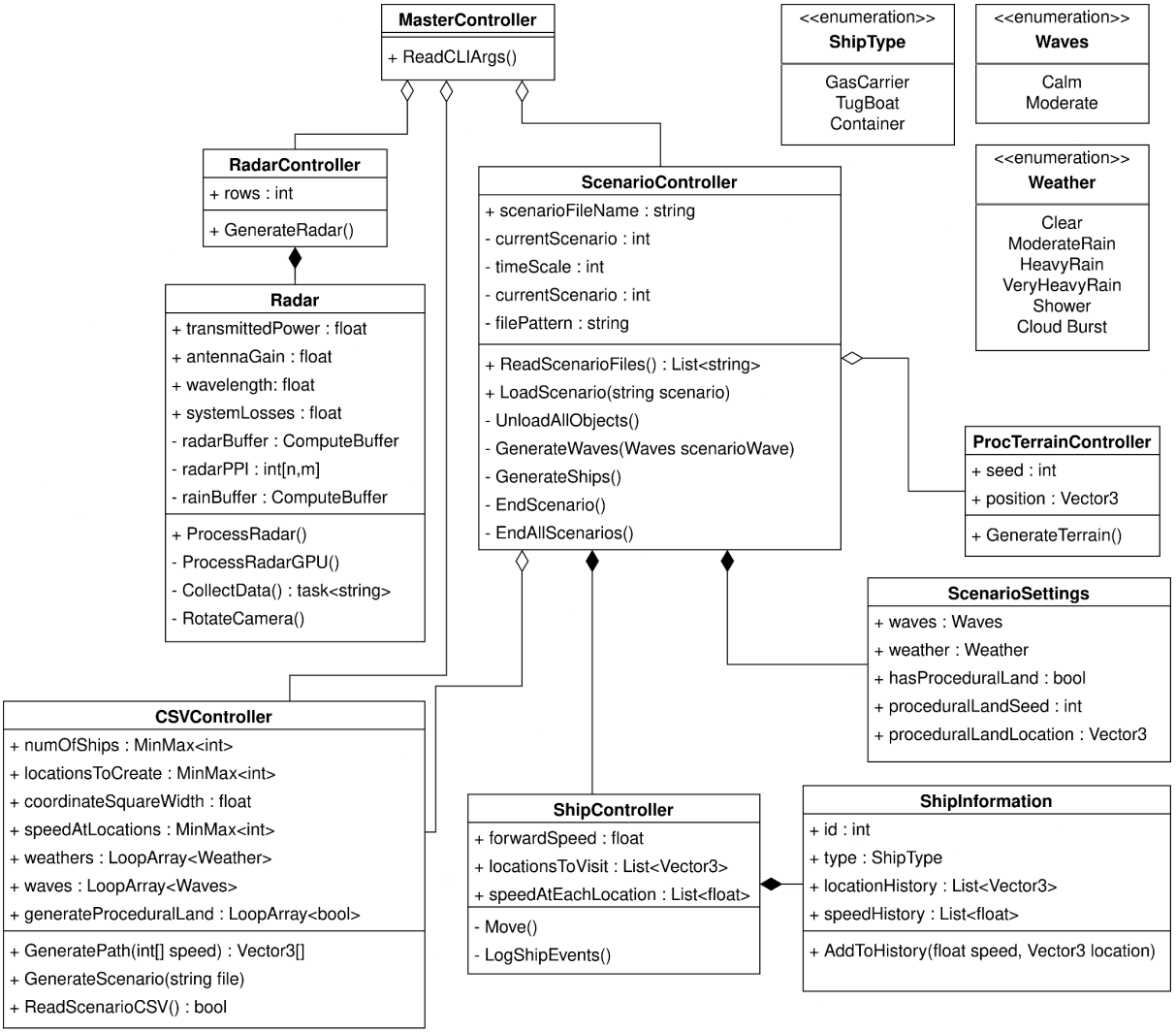

Fig. 6.5. Class Diagram.

6.2.6 Radar Simulation Design

Our project's simulation system incorporates radar simulation as a key component. Drawing from existing literature, we've identified three main approaches to simulating radar detection: ray tracing, ray casting, and low-fidelity methods. Our approach combines ray casting and low-fidelity methods, with additional features to enhance realism. This section outlines the design of our radar simulation system.

Radar Basics

Radar (Radio Detection and Ranging) is a device that uses electromagnetic radiation to detect objects and measure their distances. The basic operation involves transmitting a radio signal, which propagates and reflects off objects. The reflected signal is then received by the antenna, and the time between transmission and reception is used to calculate distance. While modern radars have advanced features and high detection resolution, this fundamental concept remains unchanged.

Radar Equation

The foundation of our radar detection simulation is the radar equation [24]:

$$P_r = \frac{P_t \lambda^2}{(4\pi)^3 d^4} G^2 \sigma, \tag{6.1}$$

where P_r = power received (W),
P_t = power transmitted (W),
G = antenna gain,
λ = wavelength of radar waves (m),
d = range (distance) of target (m),
σ = radar cross section (RCS).

Equation 6.1 relates the power transmitted, wavelength, antenna gain, and target distance to the power received by the antenna. While simplified, it provides a framework for modeling radar detection behavior.

Simulation Design

Our simulation allows customization of P_r, G and for specific radar types, enabling realistic representation of various devices. A unique aspect of our approach is the incorporation of RCS calculation for enhanced realism, which is often omitted in existing simulation approaches. RCS is defined as a target's ability to reflect radar signals. We use the model [24]

$$\sigma = CS \times R \times D, \tag{6.2}$$

where CS = Projected cross section (m^2),
R = Reflectivity,
D = Directivity.

In our simulation, CS is calculated during runtime based on the target's visible cross-section, and D is measured during simulation, considering scatter angles. R is assumed to be 1, and is a computational limitation for our project, which we will discuss in later sections. This comprehensive approach allows for more accurate representation of radar interactions with different objects and materials. Specifically, our approach combines both raycasting, used to find D and CS, and low-fidelity methods through the radar equation to model the behavior of radar and EM waves accurately, which has not been done in the literature.

Data Format

Our simulation generates radar data in a PPI format. The data is represented in a radial coordinate system, where the center represents distance zero, and the outer circle represents maximum range, shown in Fig. 6.6. Points are distributed by azimuth (angle relative to the radar). Computationally, this is stored as a 2D array, with one

dimension representing azimuth and the other representing distance, shown in Fig. 6.7. This array is used to generate the PPI image and serves as input for deep learning detection models.

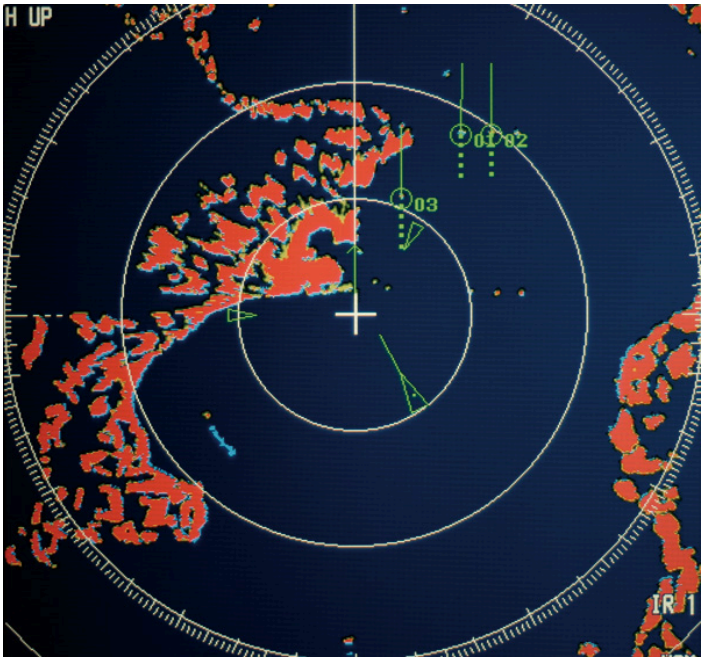

Fig. 6.6: PPI Image.

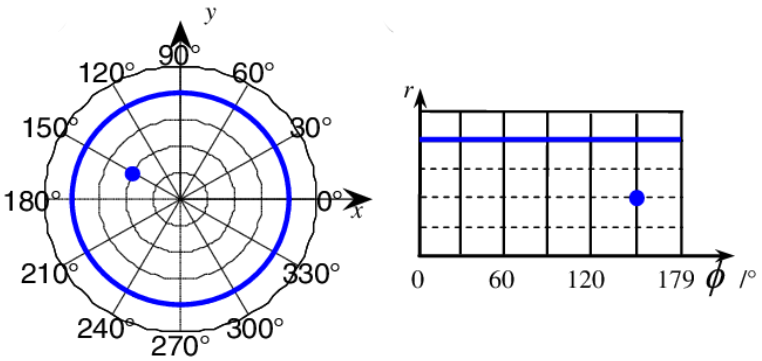

Fig. 6.7: Azimuth angle (left) and distance (right).

Using the radar equation model, our simulation follows a structured process:

- Initialize radar parameters:
  - Set power transmitted (P_t)
  - Set antenna gain (G)
  - Set wavelength (λ)
- For each azimuth:
  - Calculate target distances (R)
  - Estimate Radar Cross Section (σ) for each target
  - Compute received power (P_r) using the radar equation
  - Update the PPI data array

- Generate PPI image from the data array

This approach combines physical modeling with computational efficiency to provide realistic radar simulation results. By incorporating detailed RCS calculations and customizable radar parameters, our simulation offers a more comprehensive and flexible tool for radar simulation and dataset generation.

Terrain Generation:

1) Procedural

In order to have a more diversified dataset, we implemented procedural terrain generation to add more noise to PPI images for radars that get close to land. To achieve this, we first generate a heightmap—a grayscale representation where lighter values indicate higher elevations and darker values represent lower elevations, to mimic the natural elevation of terrain—using Perlin noise as shown in Fig. 6.8. Perlin noise is a gradually changing noise that represents land more realistically. This noise map was summed by a falloff map to achieve smooth mountain edges around the border. The falloff map was generated using equation 6.3 to achieve a gradually changing map from white (sea level) to black (above sea level).

$$f(x) = \frac{x^3}{x^3+(10-10x)^3} \tag{6.3}$$

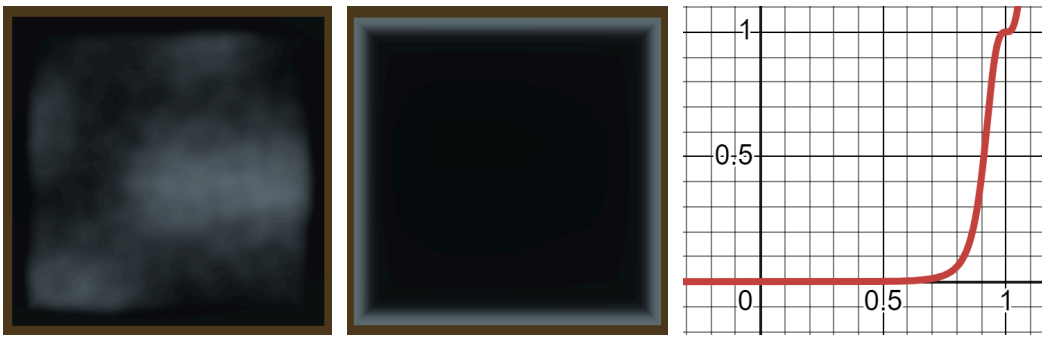

Fig. 6.8: Generated noise map (left) + Falloff map (middle) + Falloff map function (right).

Lastly, we generate a color map by assigning terrain types to the various height ranges and from that color map we construct a 3D mesh as shown in Fig. 6.9. Fig. 6.10 illustrates how octaves in procedural terrain generation add layers of detail: The first octave defines the broad "main outline" of the terrain, while higher octaves (e.g., 2 and 3) introduce progressively smaller details like "boulders" and "small rocks." The lacunarity controls the increase in frequency between octaves (i.e., how quickly new details are added), while persistence controls the decrease in amplitude, meaning how

much each octave contributes to the overall shape. Together, they create natural terrain variations by adding finer and finer details with each octave.

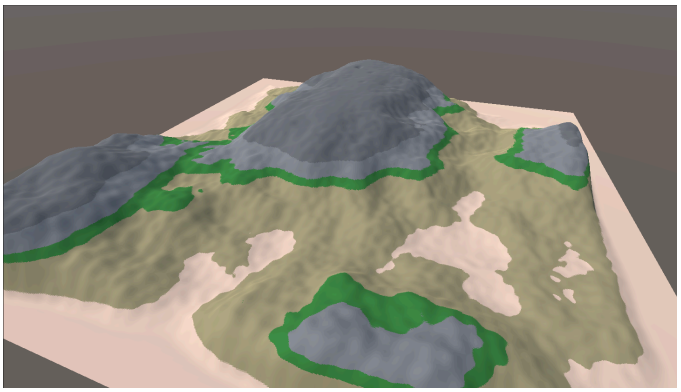

Fig. 6.9: Generated mesh that represents the procedural terrain.

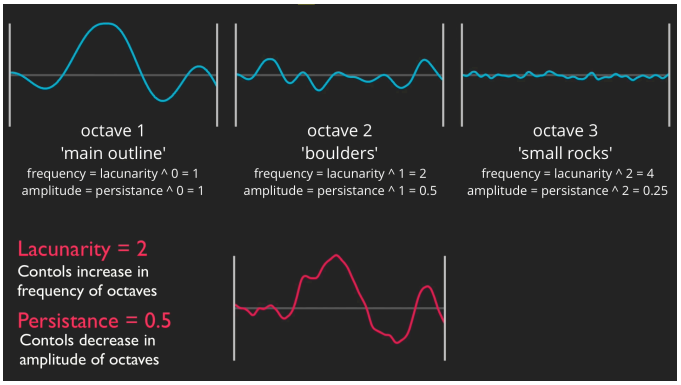

Fig. 6.10: Natural looking terrain achieved through the summation of different octaves.

2) UAE Coastline Generation

In the context of our project, the UAE's coastline is fundamental to our simulation system. Since the goal of our project is to enhance maritime surveillance along the UAE's border, we aim to replicate the coastline as precisely as possible.

Replicating the entire border of the UAE requires extensive data and time. Therefore, to test our system's functionality we focused on accurately replicating the eastern portion of the UAE border, specifically the region around and including Khorfakkan. This area features mountains and cliffs of various shapes and sizes and is frequently subjected to environmental conditions like fog, heavy rain and low pressure storms [26]. This choice allowed us to effectively test our simulation system under different scenarios.

In Unity, terrains are represented as a 2D matrix called heightmaps as shown in Figure 6.12. These heightmaps are also known as Digital Terrain Model (DTM) [25]. The values of these height maps range from 0 to 1 corresponding to the highest and lowest

points of elevation. For generating the UAE's coastline, we imported heightmap images and transformed them into a realistic environment via custom written code. Once the coastline was generated, the desired ground texture was applied to enhance its realism. We also populated our coastline with vegetation and objects such as trees, bushes, and buildings to enhance its visual appearance and ensured all elements blend naturally.

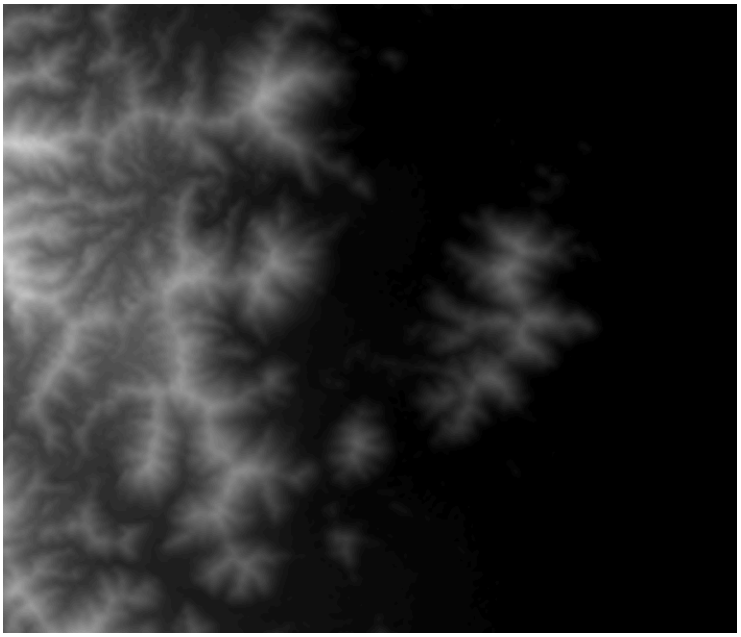

Fig. 6.12: Heightmap representation of the coastline region of Khorfakkan. The grayscale image displays terrain elevations, with lighter areas representing higher elevations and darker areas indicating lower regions.

6.3 Alternative Designs

Unity Engine for the Simulation Environment: Our choice of using Unity for building the simulation is based on the numerous research and documentation available for this engine. This research enables us to learn and expand on previous works related to simulation environments. Additionally, Unity is built with simplicity and intuitiveness in mind allowing us to quickly learn it and focus on building the system. The engine is also more performant compared to other alternatives, which allows us to have a large number of radars and vessels simulated for a realistic test of the system. Finally, Unity offers a deterministic physics engine, which we believe would be helpful in building a deterministic system. Although we chose Unity for our simulation, there were multiple alternatives that we considered before making our choice.

- **Alternative 1 - Unreal Engine:** A powerful engine that is typically used for game development and can be used for building a simulation. This game engine is very powerful and capable, and can be performance heavy. However, the primary reason we chose Unity over Unreal is due to Unity offering a

deterministic physics engine. On the other hand, Unreal has displayed non deterministic behavior in some simulator engines [27]. Furthermore, this engine is more complex to use and learn compared to Unity.

CenterNet Model for RTD for the Onboard Software: We selected the CenterNet model as our primary solution due to its superior results. CenterNet demonstrated the best F1-score in detecting ships while maintaining efficient processing speeds suitable for the onboard software. Although we chose this model, we experimented with several alternatives before making our choice. A more detailed analysis will be presented in the implementation section.

- **Alternative 1 - YOLOv11:** While YOLOv11 is renowned for its object detection capabilities and real-time performance, our experimental results showed that it achieved lower detection accuracy compared to CenterNet when applied to our dataset, making it a less suitable choice for our case.

- **Alternative 2 - STFA-GCN:** The STFA-GCN model presented an innovative approach to ship detection, but its operational requirements proved impractical for our use case. The model requires accumulating multiple frames of PPI images to construct the necessary graph data structure before making predictions. In our radar system, each frame takes approximately 15 seconds to generate, meaning we would need to wait around 150 seconds (2.5 minutes) just to collect enough frames for a single detection if we opted for 10 frames. This substantial delay, combined with the additional time required for graph data construction and processing, would significantly impact our system's responsiveness. The resource-intensive nature of graph construction, coupled with this inherent temporal constraint, made STFA-GCN unsuitable for our real-time detection requirements where minimizing latency was a crucial objective.

Web Application for the Visualization Platform: Our choice to go with a web application is mostly a result of the fast development of web applications, and it is easy to use alongside the simulation. Additionally, as some team members use Linux, cross compatibility with different platforms while having minimal dependencies to install was essential for us.

- **Alternative - Desktop or Mobile Application:** The main reason against the team’s decision to use a desktop or a mobile application for the visualization platform stems from the fact that a web application is cross compatible with different devices unlike desktop or mobile applications. Additionally, web apps usually have a hot reload feature during development, which can speed up the development and testing of the application compared to these alternatives.

7. Project Management Plan

The GANTT chart displays a comprehensive plan of the deliverables alongside the tasks assigned to each team member as shown in Fig. 7.1. Our full GANTT chart is attached with this report.

TASK	START DATE	END DATE	ASSIGNED TO
Enhanced Problem Statement	09/01/2024	09/01/2024	Yousif
Literature Review	09/02/2024	09/04/2024	All team members
Requirements	09/05/2024	09/05/2024	Ali
Cost Estimate	09/06/2024	09/06/2024	Ali
Project Management Plan	09/07/2024	09/07/2024	Ibrahim
Social Impact and Standards	09/08/2024	09/08/2024	Mohammad & Ali
Final Review of Report Part I + Submission on Ilearn	09/09/2024	09/09/2024	All team members
System Architecture (SW/HW)	09/10/2024	09/14/2024	Ali & Ibrahim
Detailed Design			
Simulation of the Proposed System	09/15/2024	09/15/2024	Ibrahim & Yousif
Dataset Creation	09/16/2024	09/16/2024	Yousif & Ali
Machine Learning Models	09/17/2024	09/19/2024	Mohammad & Ali
Visualization Platform	09/20/2024	09/21/2024	Mohammad & Ibrahim
Feedback and Revision	09/22/2024	09/25/2024	All team members
Final Review of Report Part II + Submission on Ilearn	9/26/2024	9/30/2024	All team members
Design Validation and Testing			
Simulation System Testing and Evaluation	10/01/2024	10/01/2024	Ali & Ibrahim
Testing Machine Learning Models Performance	10/02/2024	10/03/2024	Mohammad & Yousif
Visualization Platform Performance Testing	10/04/2024	10/05/2024	Yousif & Ibrahim
Results and Analysis	10/06/2024	10/9/2024	All team members
Feedback and Revisions	10/10/2024	10/15/2024	All team members
Demo Preparation	10/16/2024	10/20/2024	All team members
Demo to Advisor	10/21/2024	10/21/2024	All team members
Final Review of Report Part III + Submission on Ilearn	10/21/2024	10/21/2024	All team members
Final Report Draft I	10/22/2024	10/28/2024	All team members
Advisor Feedback + Revision	10/29/2024	11/03/2024	All team members
Final Report Draft II	11/04/2024	11/10/2024	All team members
Draft Report Submission to Advisor	11/11/2024	11/11/2024	All team members
Final Review of the Report	11/12/2024	11/17/2024	All team members
Submission of Final Report on Ilearn	11/18/2024	11/18/2024	One member of the team
Preparation - Presentation and Demo	11/19/2024	11/22/2024	All team members
Final Rehearsals	11/23/2024	11/24/2024	All team members
Final Presentation	11/25/2024	11/25/2024	All team members
Demo	11/25/2024	11/25/2024	All team members
END			

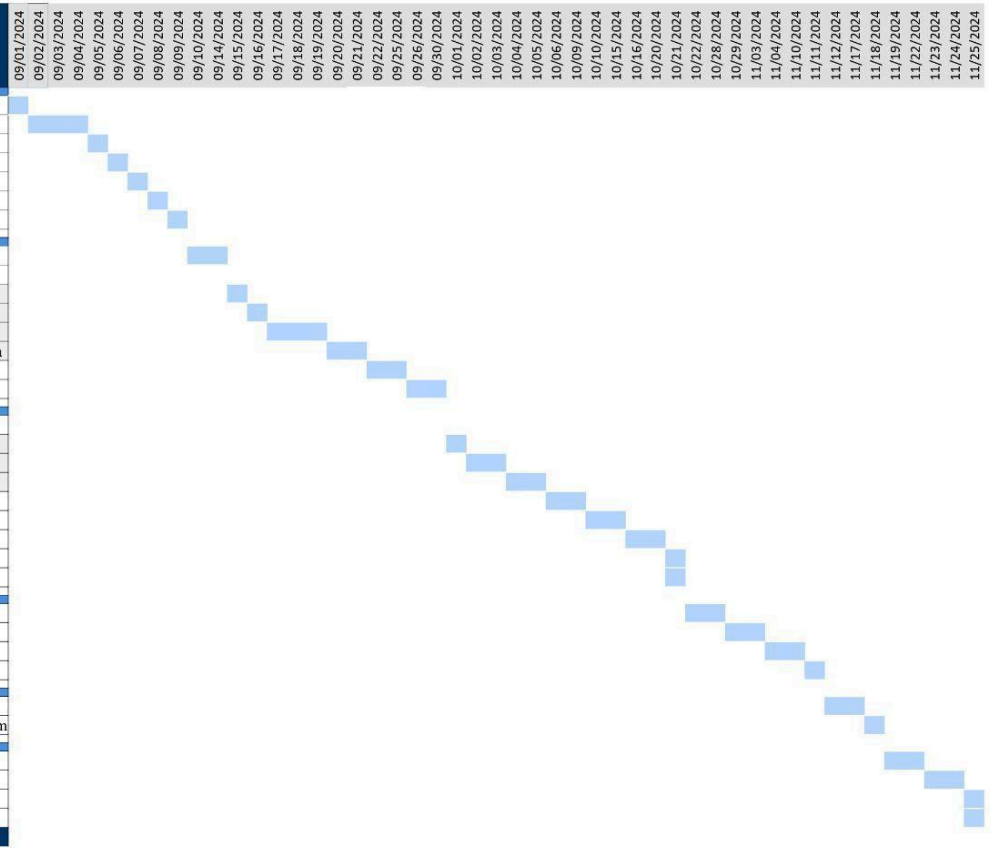

Fig. 7.1: GANTT chart

8. Implementation

8.1 Hardware

As this is a software based project, there are no hardware components.

8.2 Software

8.2.1 Simulation

The simulation system is composed of multiple components that function independently, and provide a realistic representation of different scenarios. Since the simulation was built in Unity, we wrote the following scripts in C#. The only exception to this were the compute shaders, which run on the GPU and are written in high-level shader language (HLSL).

Radar:

Since the core of our project is the radar, we spent plenty of time creating and fine tuning the radar script to ensure the radar accurately and realistically mimics a real world radar.

1) **RadarScript**: The script that is responsible for all the radar's functionality. It achieves this by spawning a depth camera for the radar and utilizing our custom made shader to calculate the depth and normals. Using this information we apply the radar equation to calculate the radar strength. The main functions that allow the script to fulfill its responsibilities are:

    - **`SpawnCameras():`** Creates the camera required to read the depth and normal information.
    - **`ProcessRadarDataGPU(kernelIndex):`** Process the radar data on the GPU using the ProcessRadarData compute shader that we wrote. This function passes the required information to the GPU including the radar equation parameters.
    - **`CollectData():`** An async function that creates a json of the radar information such as the radarID and PPI array. This information is passed outside the simulation to the onboard software using sockets.

2) **`ProcessRadarData.compute`**: In order to optimize the radars, we opted to run part of the intensive code on the GPU. To achieve this, we wrote a compute shader that calculates the distance from the object after identifying if the object has parallel surfaces to the radar and applies the radar equation to get the signal strength. The pseudocode for the radar detection process is provided in algorithm 8.1.

Algorithm 8.1: Radar Processing

```
FOR each azimuth
    Collect radar normal and distance data
    Calculate Cross Section
    Calculate directivity
    Update RCS Array
    Calculate received signal strength
    Add rain data to PPI
    IF azimuth equals zero
          Get ship ground truth coordinates
          Generate bounding boxes for ships
          Generate JSON with PPI data and metadata
          Send JSON through websocket
    ENDIF
ENDFOR
```

Controllers:

To enhance the operation of the simulation, we have created several controller scripts that manage the various sections of the simulation:

1) **CSVController**: This script manages the scenario CSVs. A scenario file is defined by a csv file that contains all information regarding ship paths. Additionally, each scenario has a scenario setting defining the weather and wave condition and procedural land seed. More specifically, the script aims to create a new CSV with each ship having a randomly generated path to follow, and it has a helper function to read the CSV while storing the information in lists for ScenarioManager to make use of. These functions are essential for the script and are defined as follows:

    - **`GenerateScenario(filename):`** Creates a CSV with customizable parameters such as number of ships, number of locations the ships will visit, and their speed while moving to a location. Since these parameters can be customized, we can also randomize it to create a large amount of CSVs for dataset generation ensuring the CSVs are all unique.

    - **`ReadScenarioCSV(shipsInformation, shipLocations, scenarioSettings):`** Reads the scenario CSV and stores the data in their respective lists. First, shipsInformation stores the ship ID, name, and type. Second, shipLocations store the ship ID, x and z coordinates, and the speed. The first coordinate is the starting position of the ship and the remaining coordinates are the locations the ship will visit in order. Finally, scenarioSettings currently only stores information about the waves.

2) **ScenarioController**: This script is responsible for generating the scenarios and keeping track of the objects related to ships and waves. The script's main functions are reading the scenario file(s) name(s) from a directory, loading the scenario(s) parameters, and ending the scenario(s). Furthermore, the script keeps track of scenarios to generate all scenarios in the directory consecutively. The main functions of this script that ensure it is capable of achieving its objectives are the following:

    - **`ReadScenarioFiles():`** Stores the scenario file names that were located in a specified directory. These scenario files are later read when running the scenario.
    - **`LoadScenario():`** The function that reads the scenario CSV utilizing the CSVController, and loading all objects defined in the CSV. Additionally, the script informs the ships with their respective locations to visit by utilizing the ShipController script.
    - **`EndScenario():`** This function unloads all the objects in the current scene.
    - **`RunNextScenario():`** If the user wants to run more than one scenario, this function keeps track of when the scenario has ended and loads the next scenario.

3) **ShipController**: This script is responsible for operating the movement of the ships. It does this by implementing the following functions:

    - **`MoveShip():`** This function uses the ship locations it received from the scenario manager to apply a specified force that results in the required speed. Additionally, it handles the ship rotation to ensure the ship always faces its next destination.
    - **`LogShipEvents():`** This coroutine logs the ship events (speed and location) every couple of seconds. This information is used for debugging and ensuring the radar is correctly detecting the ships.

4) **RadarController**: The script that manages the creation of radars. It is capable of generating the radars in a lattice network by keeping track of radars at each row. Additionally, it is capable of rotating or moving the entire lattice network.

    - **`GenerateRadars(numOfRadars):`** Generates multiple radars in a lattice network structure. This function also gives the radars a unique ID to ensure they can properly transfer their data separately.

Procedural Terrain Generation:

To add variability and generate a diversified dataset, we implemented procedural land generation. These are the classes to achieve that:

1) **MapGenerator:** This class takes the map inputs from the user and generates the desired map. These inputs are the map chunk size, level of detail (to increase/decrease the terrain 'resolution'), noise scale, number of octaves, persistence, lacunarity, seed, offset, mesh height multiplier, mesh height curve, and the different terrain types. Then the user can click a button to generate a map.

2) **MapDisplay:** Takes the noiseMap and returns it into a texture then applies that texture into a plane in our scene.

    - **`DrawTexture(texture):`** applies a texture to the textureRenderer and scales the plane (or object) to match the dimensions of the texture (width and height).
    - **`DrawMesh(meshData, texture):`** creates a mesh from mesh data, applies the mesh and texture to the mesh filter and mesh renderer, and assigns the mesh to the collider for physics interactions.

3) **MeshGenerator:** Generate a 3D terrain mesh based on a height map, applying height multipliers and curves to control the shape and appearance of the terrain. It consists of two key parts: the mesh generator class, which defines how to create a mesh from heightmap data, and the mesh data class, which stores and manages the mesh's vertices, UVs, and triangles.

    - **`GenerateTerrainMesh(heightMap, heightMultiplier, heightCurve):`** Takes a 2D heightmap, multiplies the height values by a height multiplier and curve, and generates a 3D mesh with vertices, UVs (for texturing), and triangles.

4) **ProcTerrainController:** Generates and controls terrain in a scenario-based environment. It creates a different terrain for each scenario, aiding in the generation of varied datasets for deep learning model training.

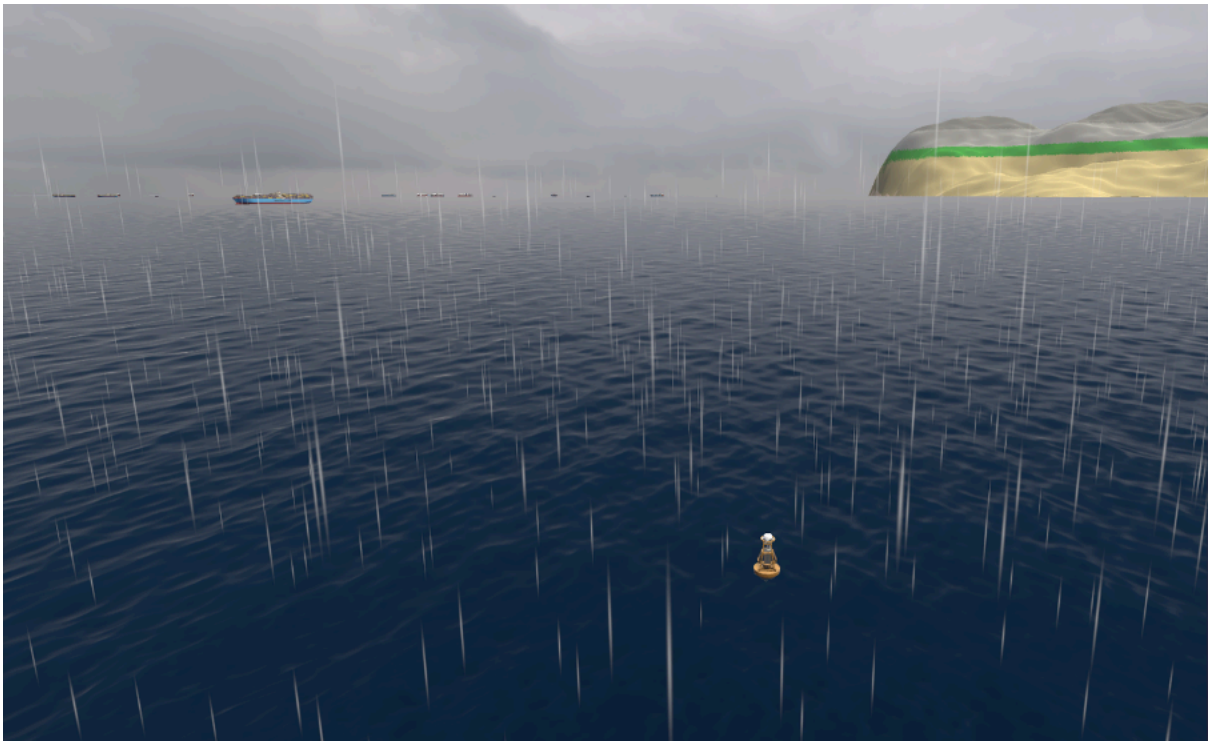

Fig. 8.1: A screenshot of the simulation. The radar (bottom yellow buoy) can be seen, as well as vessels (at the horizon) and weather, and procedurally generated land (right side).

UAE Coastline Generation:

We replicated the coastline with the high degree of accuracy to achieve the project's goal of improving maritime surveillance along the UAE's border. These are the details of UAE coastline implementation:

The `Terrain` component is attached to the GameObject in Unity. In the code, RuntimeTerrain Generator script interacts with the GameObject allowing modifications to its Terrain component.

RuntimeTerrainGenerator: This script contains the code used to generate the UAE coastline for our simulation system. The key methods and their functionalities developed in the script are described below.

- **`ApplyHeightmap():`** This function applies heightmap texture to the terrain. It samples the grayscale height data from the `heightmapTexture` using bilinear sampling. It normalizes and applies the sampled heights to the terrain. It also smooths the heights over multiple iterations for natural terrain shaping. The function effectively shapes the terrain's physical features based on the heightmap.

- **`BilinearSample(texture, x, y):`** This function uses bilinear interpolation to extract a smooth height value from a height map texture by blending the colors of neighboring pixels. It takes a texture and (x,y)

coordinates as input where these coordinates correspond to specific pixel locations on the heightmap. It finds four closest pixels around a given position which are directly above, below, left and right. It then computes the distance between these pixels and blends their grayscale values to produce a smooth height value, enabling the creation of a realistic terrain.

`SmoothHeights(heights):` The purpose of this function is to smooth the sharp and abrupt height changes between neighboring points on the terrain. It does this by averaging each height point value with its neighboring values. This results in a gradual transition between heights leading to a smoother and natural looking terrain.

- **`ApplyTerrainLayers():`** This function applies a texture layer such as sand to the terrain's surface. It initializes and applies a `splatmap`, which effectively textures the entire surface of the terrain.
- **`GenerateTerrain():`** The purpose of this function is to manually regenerate the terrain in Unity based on the latest heightmap texture and settings. It combines all terrain-generation functions into a single process. Before the terrain regenerates, this method makes sure that essential components such as `Terrain` component and `heightmapTexture` are assigned. Upon successful verification, the function then sets the terrain size, applies the heightmap and terrain layer. Additionally, it can optionally spawn trees, bushes, houses and set the skybox material.

8.2.2 Radar Onboard Software

The onboard software runs for each individual radar, it is responsible for processing the incoming data, running the deep learning model to detect ships, turning the data from relative coordinates (relative to the radar) to absolute latitude, longitude, and getting the data to the database.

8.2.2.1 General Structure

The radar onboard software comprises multiple components, each handling specific parts of the radar data processing pipeline. This section describes the main functionalities and their roles within the system.

1) **Coordinate Conversion (`locations.py`)**:

This script handles the conversion of Unity coordinates (relative to the radar) to absolute latitude and longitude coordinates. It takes the x and y positions in meters and applies formulas based on Earth's radius and trigonometric functions to obtain the new latitude and longitude based on a central point. This conversion is essential for aligning radar data with real-world locations.

2) **YOLO Inference (`yolo_infer.py`)**:

This script processes the radar PPI data using the pretrained YOLO model. First, the PPI data is normalized and converted into an image. Then, the model detects ship locations within the PPI, outputting the coordinates of detected ships as [distance, azimuth] pairs. The script converts the locations into [latitude, longitude] pairs and updates the database with the new detections.

3) CenterNet inference (`ceneternet_infer.py`)

This script processes the radar PPI data live using the pretrained CenterNet model. The PPI data is received through websocket, and the script runs the model to detect ship locations. The output of the model is an array of ship locations [distance azimuth]. The script converts the locations into [latitude, longitude] pairs and updates the database with the new detections.

4) **Onboard YOLO Processing and Visualization (`onboard-yolo.py`)**:

This is the main control script for the radar's onboard software. It establishes a WebSocket connection to receive real-time radar data from Unity, invokes the pretrained YOLO model for ship detection, and converts Unity coordinates to latitude and longitude. Finally, it updates the radar's location and detection results in the database so that they can be retrieved by the visualization platform.

8.2.2.2 Dataset Generation (`generateDataset.py`):

This script controls various parameters in the simulation environment. It launches the Unity simulation system with parameters specified in the `configuration.yaml` file, and creates different scenarios based on these parameters. These parameters are defined in Table 8.1.

Parameter	Description
`nships`	Number of ships in a scenario defined by a range [minAmount, maxAmount]
`nLocations`	Number of locations a ship visits during a scenario defined by a range [minAmount, maxAmount]
`coordinateSquareWidth`	Width of ship generation space
`speed`	Ship movement speed (in knots) defined by a range [minSpeed, maxSpeed]
`radarRows`	Number of rows in radar lattice network
`radarPower`	Power transmitted in W

`radarGain`	Gain of the radar in dB
`waveLength`	Wavelength of radar in m
`radarImageRadius`	Width of the output data array (pixels)
`antennaVerticalBeamWidth`	Vertical angle of radar beam
`antennaHorizontalBeamWidth`	Horizontal angle of radar beam
`rainRCS`	RCS value for a raindrop
`nRadars`	Number of radars in a scenario
`nScenarios`	Number of scenarios to generate
`scenarioTimeLimit`	Time limit for a scenario before ending and moving to next
`weather`	List of Weather conditions to cycle through for each scenario
`waves`	List of Wave conditions to cycle through for each scenario
`proceduralLand`	List of bool to cycle through (whether procedural land is generated or not)
`generateDataset`	Bool to generate a dataset

Table 8.1: List of configurable parameters in the simulation

Each radar has its own WebSocket connection to stream data in real-time. The script receives radar data through WebSockets, processes it (e.g., by clipping the data range for consistency), and saves it in a structured JSON format with metadata, including PPI data and ship information.

8.2.2.3 Deep Learning

Following the literature review, we applied the following models for ships detection:

1) **CenterNet:**

The CenterNet model is designed to detect ships by identifying key points on PPI images. It uses a heatmap-based method where each ship target is represented by a peak on the heatmap, allowing the model to predict the presence and location of ships.

A. **Data Augmentation (`augmentations.py`)**:

The radar data is enhanced through several augmentation techniques to improve model robustness. These include flipping and shifting the radar images, allowing the model to generalize better across varying data

distributions. The dataset size increases proportionally based on the number and types of augmentations applied.

B. **Dataset Preparation (`dataset.py`)**:

The dataset preparation script loads radar data from JSON files, processes PPI data, and generates heat maps based on known ship positions. A 2D Gaussian heatmap is generated for each ship, centered at its location. This process produces labeled heatmaps for supervised learning, which represent ships as hotspots, making it easier for the model to learn spatial patterns as shown in Fig. 8.2.

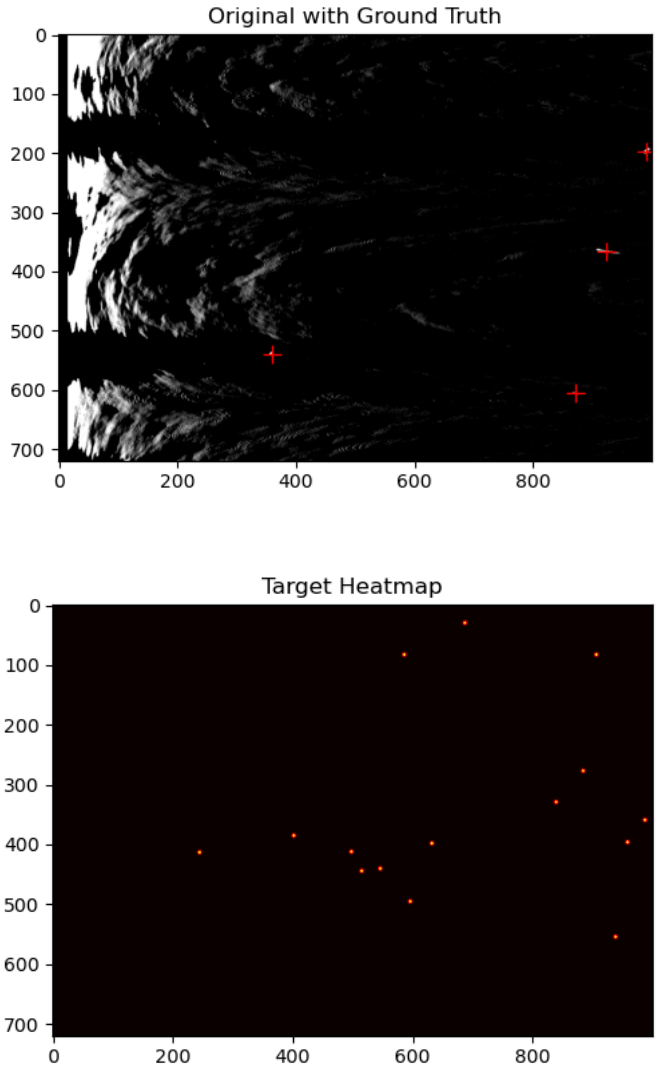

Fig. 8.2: The Original image with ground truth (top) - Target heatmap (bottom).

C. **Model Structure (`centernet.py`)**:

The CenterNet model architecture uses a ResNet-based backbone for feature extraction, followed by upsampling layers to generate a heatmap that

highlights potential ship locations as peaks. The model uses a Focal Loss function to focus on correctly identifying ship locations while downplaying less relevant regions. Additionally, it incorporates DBSCAN clustering on the detected peaks to group closely located points into clusters.

D. **Training and Evaluation (`main.py`)**:

The training script loads the prepared and augmented dataset, applies the CenterNet model, and optimizes it using Focal Loss. An example of a predicted heatmap produced by the mode is shown in Fig. 8.3. The model is trained over multiple epochs, with early stopping to prevent overfitting and learning rate adjustments based on validation performance. The best results we accumulated with CenterNet were precision = 0.931, recall = 0.945, and F1-score = 0.938. The confusion matrix is shown in Fig. 8.4.

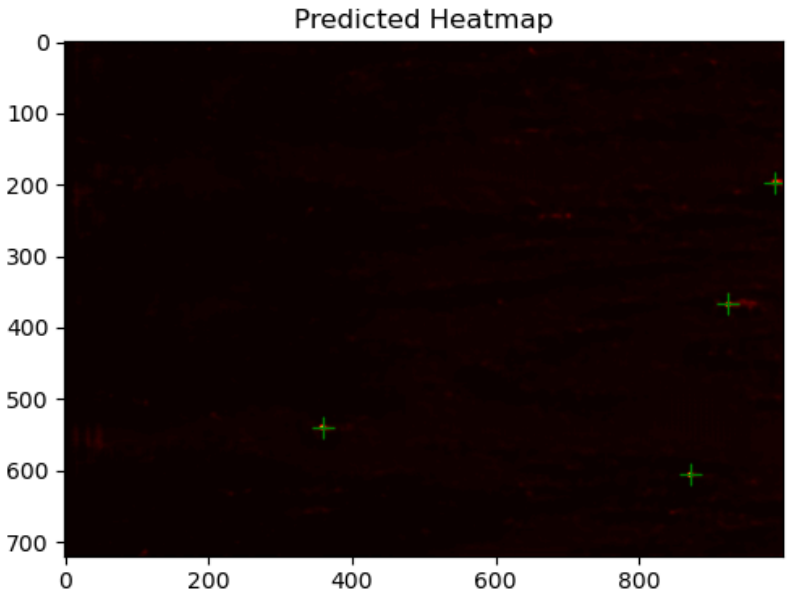

Fig 8.3: An example of a predicted heatmap

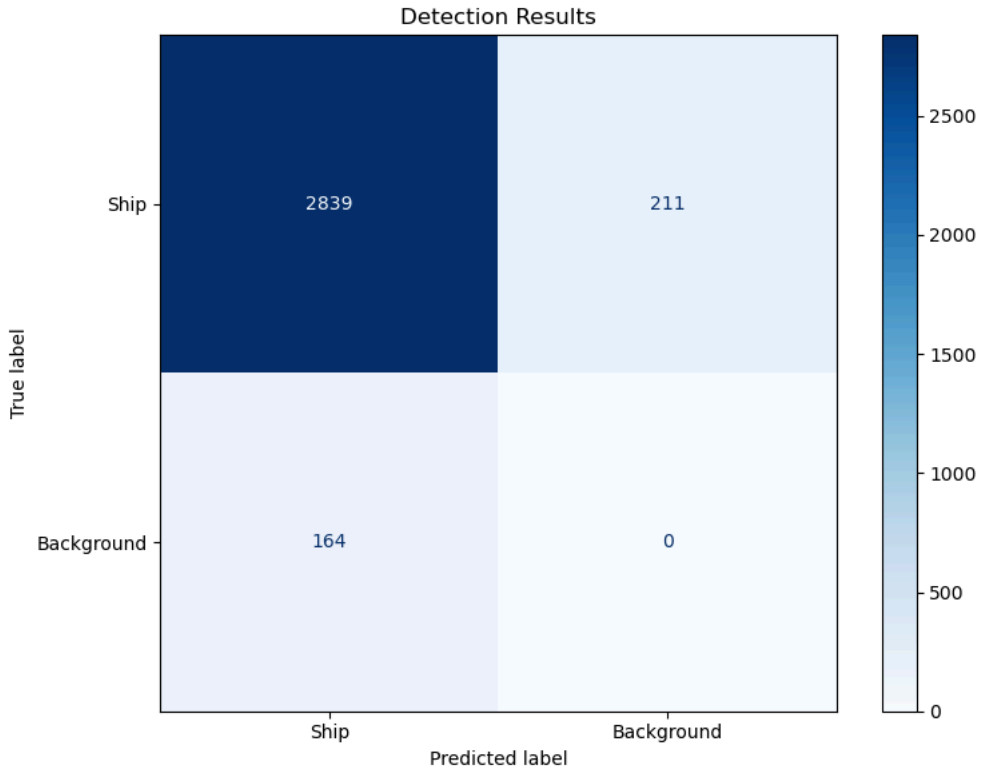

Fig 8.4: The confusion matrix from CenterNet on a test dataset.

2) **YOLOv11:**

The YOLO model detects ships by generating bounding boxes around detected targets. YOLO processes images in a single pass, making it ideal for real-time applications like radar ship detection. The YOLO implementation includes steps for data preparation, model training, and testing.

A. **Data Preparation (`train.py`):**
Radar data is read from JSON files, which include PPI arrays and ship bounding information. The data is processed to create images with bounding boxes, formatted for YOLO training. The bounding boxes are normalized according to YOLO's requirements specified in the `ultralytics` module, making each ship a detection target in the generated images as shown in Fig. 8.5. The dataset is divided into training and validation subsets, with images saved in separate directories.

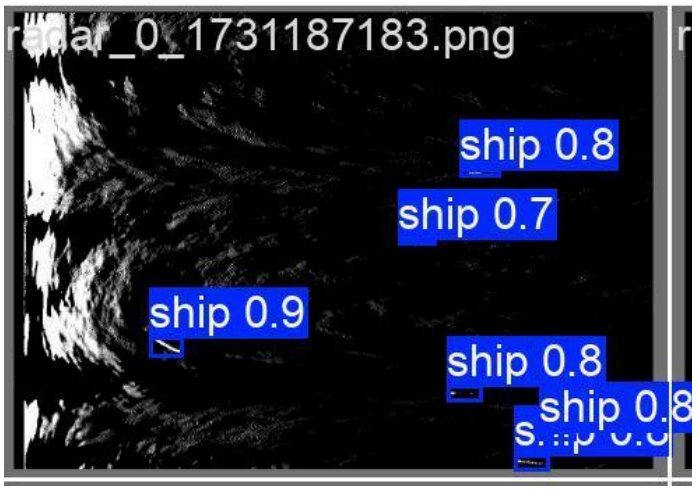

Fig. 8.5: Predicted bounding boxes for a single PPI image.

B. **Model Training (`train.py`):**
The model is loaded and trained on the processed dataset for multiple epochs. Each radar image is passed through the model to adjust weights, learning to detect ships accurately. The best results acquired were precision = 0.88, recall = 0.77, and F1-score = 0.82 as shown in Fig. 8.6.

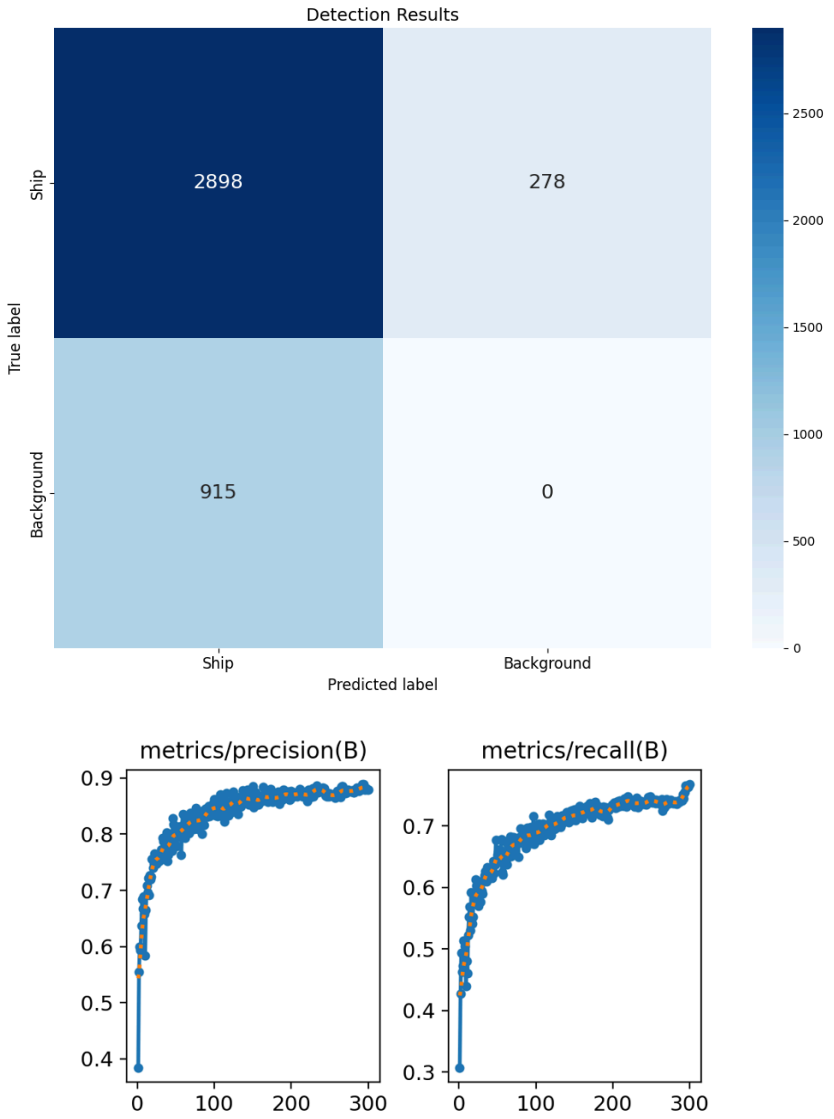

Fig. 8.6: Confusion matrix (up) - Precision and recall vs # of epochs (down).

C. **Model Export (`test.py`):**
After training, the YOLO model is exported to NCNN format, a lightweight format suitable for embedded applications, ensuring compatibility with onboard systems.

The CenterNet model achieved better results compared to YOLO as shown in Table 8.2. As a result, we would use it in the onboard software to detect ships' locations.

Method	Precision	Recall	F1-score
YOLOv11	0.88	0.77	0.82
CenterNet	**0.93**	**0.95**	**0.94**

Table 8.2: Precision, recall, and F1-score comparison between CenterNet and YOLO models.

8.2.3 Visualization platform

The visualization platform serves the purpose of visualizing the output data from the onboard software in the form of a map. The outputs are in the form of latitude, longitude points on the map corresponding to detected ships. The platform shows the outputs of multiple radars simultaneously, by reading from the central database. This is the part that brings the project to life, showing how the final output of the entire concept as a single data layer, to be used in more complex military software.

The visualization platform has 2 parts: the API (backend), and the web based (front-end). The API serves to allow the radar's onboard software to add detections to the database as per the schema in section 6.2.4. The API also allows the front-end to read the database and request the detection data.

The API and database definition are done together in the same file using an object-relational mapper (ORM). The ORM (SQLAlchemy) with FastAPI allows us to define the database schema and create an API all in the same file allowing for easy readability and writability.

The API endpoints exposed are shown in table 8.3.

Request Type	Endpoint	Description
POST	/radars	Creates an new radar in the database
GET	/radars	Get list of all radars
PATCH	/radars/{radar_id}/location	Update a single radar's location
GET	/radars/{radar_id}	Get a single radar entry
PUT	/radars/{radar_id}/last-active	Update the last active attribute of a radar
POST	/detections	Create a new detection
GET	/detections/recent	Get list of detections within the last n minutes
DELETE	/detections/by_radar/{radar_id}	Delete all the detections of a particular radar
GET	/detection/by_area	Get list of detections within a given area

Table 8.3: RESTful API endpoints of radars and detections.

The frontend is a map that makes calls to the API and displays the detections on the map as markers. The frontend is implemented using React with a mapbox react

wrapper to allow for display of the map. The front-end makes calls to the API to get a list of detections and displays the most recent detections on the map. An example is shown on Fig. 8.7.

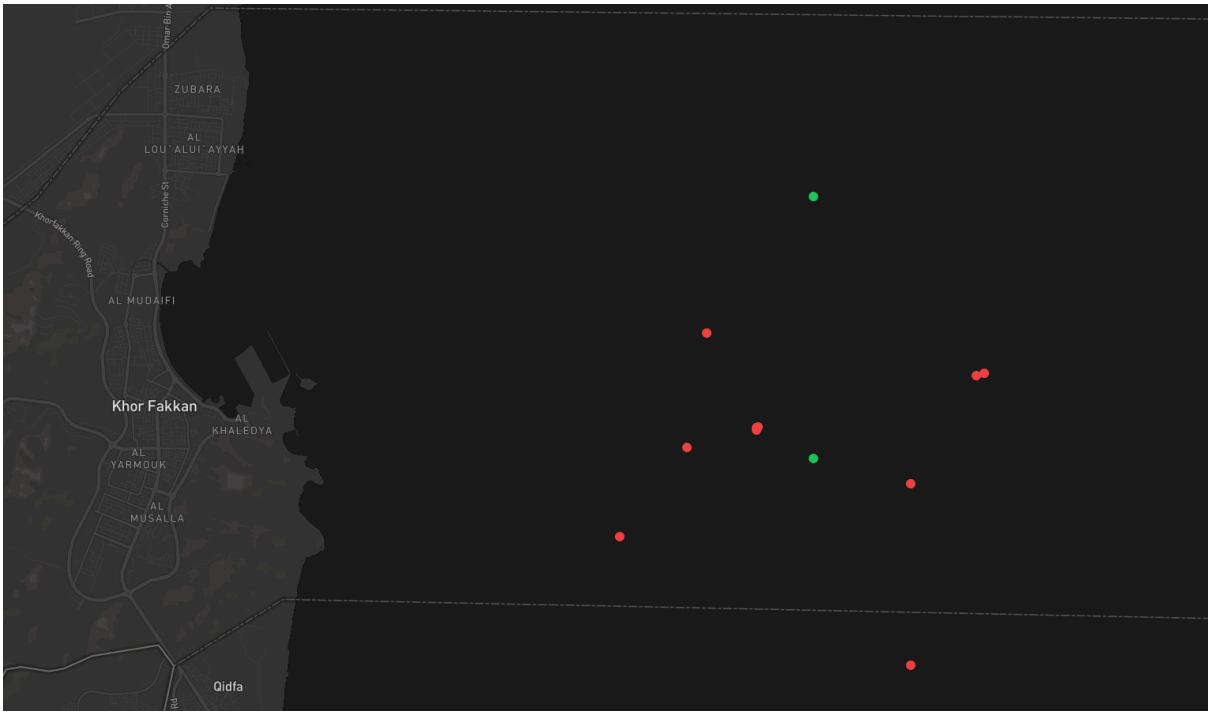

Fig. 8.7: The visualization map showing two radars (green) and multiple detections (red) at the beach of KhorFakkan.

8.3 Integration

8.3.1 Networking

In our system, the simulation system acts as the server while the radar onboard software is the client. This setup allows us to efficiently transport data from the simulation to outside it. The data transfers between these two systems using WebSocket communication that allows real-time and bi-directional communication. This communication is essential for running the entire system, as it facilitates the flow of data from the simulation to the onboard software for ship detection and subsequently to the visualization platform for mapping. The process emulates real-world data transfers from coastal guard systems to central onshore centers.

Data transferred consists of radar data (PPI images), along with additional metadata:

- PPI Images:
  - The PPI is a 2D array that represents radar scan data, where each cell stores intensity per azimuth and distance.
- Metadata:
  - Radar ID: A unique identifier for the radar.

- Timestamp: The time when the data was captured.
- Ship Data: The ground truth of the ship locations alongside the bounding boxes of each ship.
- Radar Position: The position of the radar in the simulation.
- Max Range: The maximum detection range for the radar.

This information is saved in a JSON structure and sent to the onboard software for processing and visualization. This is an example of how each JSON file looks like:


```
{
	"id": 2,
	"timestamp": 1731172429,
	"range": 5000.0,
	"PPI": [
		[0.0, 0.0, 0.0, 0.0, 11.0, 31.0, 131.0, 191.0...]
	],
	"ships": [
		{"Id": -3234, "Azimuth": 10.5, "Distance": 4304.67334,
		"Bounds": "0 4304.673 10.5 264.1514 242.2614"}, ...],
	"radarLocation": {
		"x": -0.105, "y": 100.2, "z": -912.826
	}
}
```


An illustration of how each PPI image is shown in Fig. 8.8.

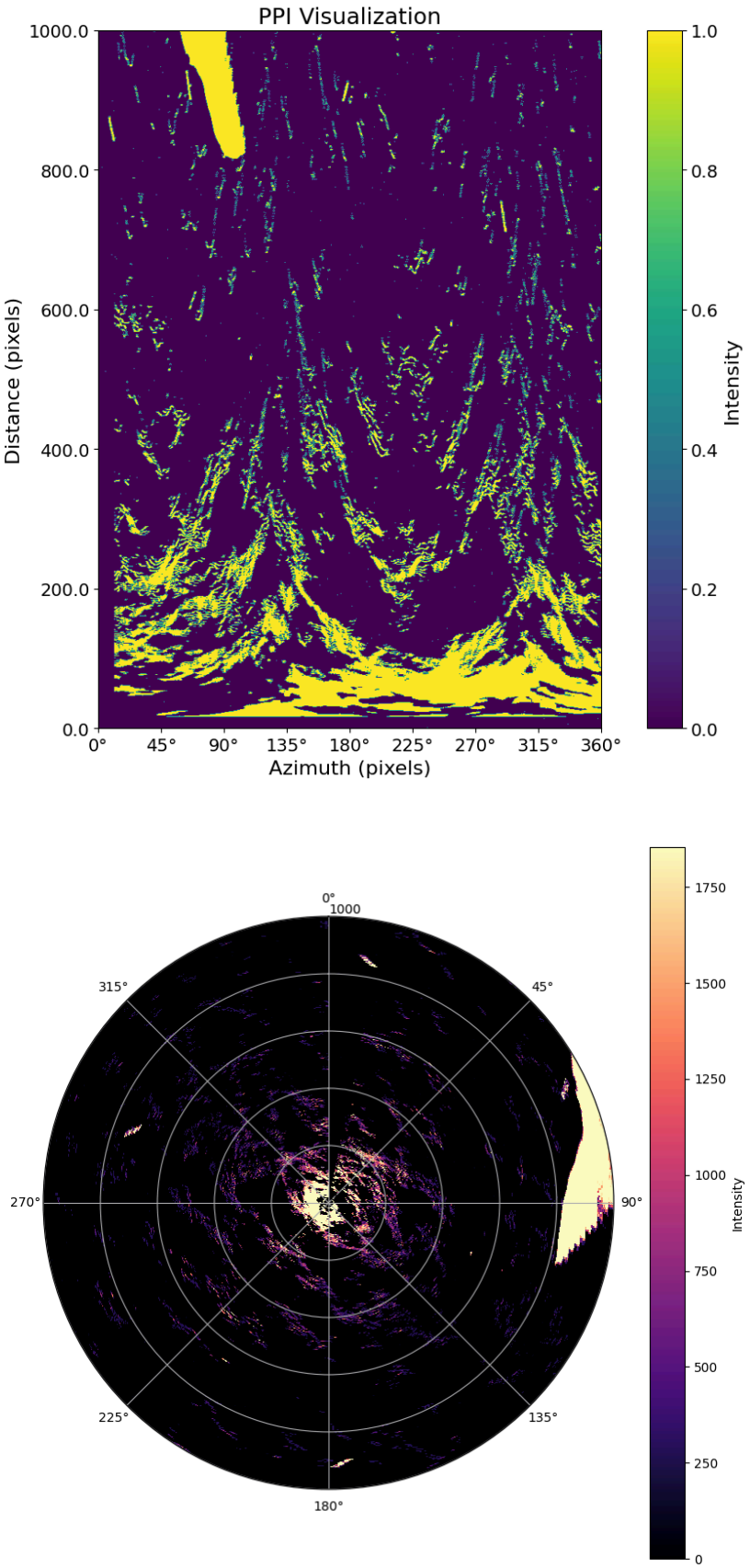

Fig. 8.8: A cartesian (up) and a polar (bottom) representation of the PPI data in a JSON file.

How the Data is Transferred:

WebSocket Communication:

1. Simulation System:

    1.1. The RadarScript initializes a WebSocket connection to a server at a path determined by the radar's ID for uniqueness.
    1.2. Radar data is continuously collected and processed within Unity, and once the data for each scan is ready, it is serialized to JSON format.
    1.3. RadarScript then sends this JSON data over the WebSocket connection using the `Broadcast()` function, making it available to all clients connected to that WebSocket channel.

2. Onboard Software:

    2.1. The `onboard-yolo.py` script acts as the client that connects to the WebSocket server running in Unity. It listens for incoming messages on a specific WebSocket endpoint, e.g.: (`ws://localhost:8080/radar{radarID}`).
    2.2. It continuously receives the radar data in JSON format.
    2.3. The onboard software performs model inference on the received PPI data, and makes an API call to save the detections to the database.

9. Validation, Verification and Performance Analysis Plan

1. Testing:

Simulation:

For the simulation we have adopted a mix of white box testing of the code and black box testing for the UI.

The white box testing consisted of writing automated unit testing for validating that the code behaved as expected based on the functional requirements. The core of the automated testing is the controllers (radar controller, scenario controller, ship controller, procedural land generation controller) of the system as they are responsible for a large chunk of the code. As shown in Fig. 9.1, our unit tests all passes.

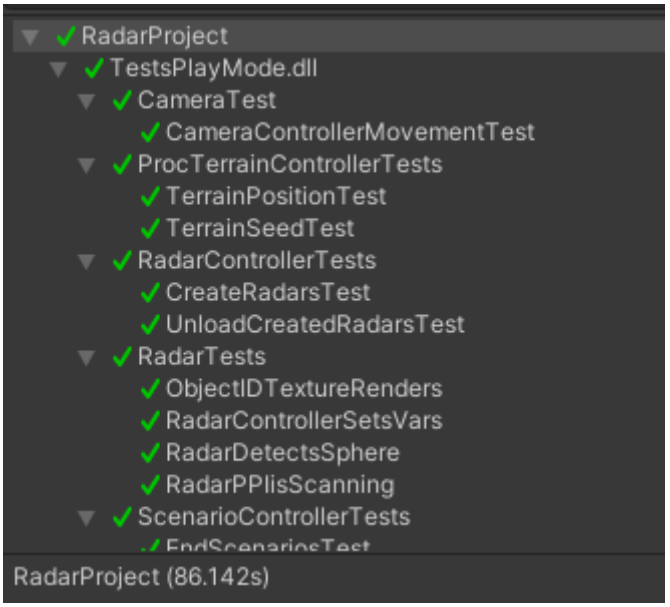

Fig. 9.1: Unit tests passing with a time of 86 seconds.

As for the UI, we adopted black box testing to confirm that the UI behaves as expected. Most of the UI consists of buttons, but there are two input fields. Therefore, we opted to test the entire UI using black box testing by deriving the test cases based on our requirements.

Test Case No.	Description	Expected Outcome	Input/s	Status
1 (FR1)	Input Data	User runs simulation with the specified values: FR1.1. Vector3 float or integer	FR1.1. Float, Strings, negative and positive numbers	PASS

		FR1.2. Integer	FR1.2. Float, Strings, negative and positive numbers FR1.3. Press any weather condition button FR1.4. Press any wave condition button	
2 (FR2)	Generating multiple scenarios	Can only generate scenarios if the input is a positive integer	1. Negative Numbers 2. Floats 3. String	PASS
3 (FR3)	Load Scenario	Scenario Loads	Press load scenario button	PASS
4 (FR4)	Start Simulation	Simulation Starts	Run the simulation program	PASS
5 (FR5)	Pause Simulation	Simulation is paused	Press pause button	PASS
6 (FR6)	End Scenario or simulation	Scenario or simulation ends	Press end button	PASS
7 (FR7)	Modify simulation speed	Simulation speeds up or slows down	Choose a value on the Integer Slider	PASS
8 (FR8)	View log	The log file is opened outside the simulation	Press ‘Open’ button	PASS
9 (FR9)	Simulation is started and saves radar messages	Radar messages are saved to a folder	User runs the generateDataset script	PASS

Table 9.1: Test cases for the simulation system.

Additional Tests

We added a test for the procedural land generation controller script. Specifically, we are testing if the positioning of the terrain is correctly applied when the user provides input for the position. The `TerrainPositionTest()` checks that the terrain is generated at the

expected location with a small tolerance for accuracy. Additionally, we tested the seed functionality in the `TerrainSeedTest()`, ensuring that the procedural terrain generation produces consistent results when the same seed is used across multiple runs, validating the reproducibility of the terrain based on the seed value.

We added additional tests for radar detection and business logic. Namely, we test the detection of generated objects, ensuring that the radar logic works as intended. Another test checks if the radar doesn't detect anything if there is nothing in the scene. This is to ensure that the radar doesn't generate interference without any objects around it.

To confirm the functionality of the generated UAE coastline, we implemented additional tests. These tests verify the correct terrain dimensions and right number of spawned objects such as trees, bushes and houses. We also developed tests to verify that the heightmap is applied correctly and checked slope constraints to make sure that the objects are not spawned over steep terrain regions. Additionally, we also provided tests for confirming exclusion zones to ensure that the objects are not spawned in restricted areas.

These test cases collectively validate the correct generation and placement of environmental elements in the coastline scene, ensuring proper implementation of user-defined parameters and respecting exclusion zones.

Through our automated unit testing, we were able to cover 76% of the code as shown in Fig. 9.2.

Name	Covered	Uncovered	Coverable	Total	Line coverage
MainCode Assembly	1847	765	2612	6232	70.7%

Fig. 9.2: Code coverage of our unit testing.

Radar Onboard Software:

The onboard software uses black box testing to test the functional requirements. Much like the simulation, automated tests were written to test the functionality of the requirements.

Test Case No.	Description	Expected Outcome	Input/s	Status
10 (FR10)	Onboard software receives radar information	Onboard software has received socket message	Radar message received from web socket.	PASS

11 (FR11)	Onboard software stores vessel locations	Detections are saved to the database	Deep learning model outputs	PASS

Table 9.2: Test cases for the radar onboard software.

Visualization Platform:

The visualization platform is tested using black box testing.

Test Case No.	**Description**	**Expected Outcome**	**Input/s**	**Status**
12 (FR12)	User can change map scale and orientation	Map scale and orientation changes with user input	Presses zoom or drags the map	PASS
13 (FR13)	User manually refreshes the map	Vessel locations are updated on the map	User presses the refresh button	PASS

Table 9.3: Test cases for the visualization platform.

10. Project Global, Economic, Societal Impact

The main impacts of our project span across economical and societal levels as discussed below.

Global Impact: Our proposed project does not have any global impact as it aims to strengthen maritime surveillance specifically.

Economical Impact: Our project offers an opportunity for the defense departments to save costs in maritime surveillance. Unlike manual maritime patrolling that involves continuous deployment of surveillance vessels and crews, we intend to use a network of marine radars. Our proposed solution will enhance the surveillance efforts by primarily reducing operational and maintenance costs of surveillance vessels, and reduce the need for continuous human monitoring. Although we can not get exact cost estimates, the U.S requested a budget of $13.8 billion for the coast guard with roughly $1 billion dedicated to the procurement, construction, and improvement of vessels [28]. Our proposed solution will cost around $300k for approximately an area of $10km^2$ while accounting for the cost of radars and charging them. Through this approach, the need for expensive surveillance methods can significantly be reduced, leading to cost savings. Additionally, the open-source dataset will allow researchers, engineers, and developers to use a large, ready-made dataset without the high costs of collecting new data. In turn, this can speed up progress in various fields while lowering the costs of data acquisition.

Societal Impact: The saved funds obtained from utilizing our radar network can be directed towards welfare initiatives such as education and healthcare to improve public well being. Furthermore, the defense departments can also utilize the saved funds to improve its defense system and strengthen military performance thereby strengthening the country's overall competitive position in the maritime territory. Additionally, given the UAE's strategic geographical location and being a global center for maritime trade, the country's maritime border is vulnerable to external threats and illegal activities. Ensuring security of the border is vital, and our project incorporates robust detection methods that contribute to improving maritime security, creating a safer environment along the coastline of the UAE. Our publicly available dataset will also enable other organizations to replicate and build upon our work, possibly improving maritime safety and security on a broader scale.

11. Standards

In our project, several standards are essential to ensure reliable and safe operations. As our project consists of multiple coding languages, we are following several coding conventions. First, for C#, we are following Microsoft code guidelines based on the .NET runtime C# coding style. Additionally, we are following PEP 8 for Python coding standards ensuring our code is easily readable and follows best practices for

maintainability and security. Finally, we are following the ECMAScript standard for our Javascript code in the visualization platform.

11.1 Summary

Maritime surveillance is essential for maintaining security and stability in today's interconnected world especially as maritime activities continue to advance. All countries rely on maritime surveillance for defense against threats such as terrorism and illegal activities. Carrying out effective surveillance is essential for monitoring and safeguarding the nation's borders. The traditional surveillance methods that have existed over the years have limitations in terms of range and coverage leaving gaps in security and defense. Our project aimed to overcome these challenges by developing a marine radar network system for the defense department of the UAE. Current maritime surveillance in the country mainly relies on transponders, coast guard patrols and other technologies. However, this approach is less effective and impactful in areas where coast guards are absent. This can lead to unregistered vessels avoiding detection and entering unauthorized maritime zones posing risks to the nation. Conducting effective surveillance of the entire maritime space of the UAE is expensive because of the requirement for numerous patrol ships and crew members.

The primary goal of our project was to simulate a lattice network of buoy mounted radars. We proposed a cost effective approach by creating an unmanned system. To achieve this, we built a simulation that encompasses the ocean, land, ships, and the network of radars while mimicking real world radar detection to create a representative scenario. Additionally, we built our own dataset of PPI images generated from the simulation and trained a CenterNet model that achieved an F1-score of 0.938 . Each node in the network acts independently and runs the lightweight model to get the ship's latitude and longitude respective to the radar. Finally, this information is sent to the database and visualized on a map. We believe our project has the potential to significantly strengthen maritime surveillance by providing an additional, independent layer of data to complement existing sources.

11.2 Future work

In future work, we aim to implement secure data propagation to enhance the system's data integrity and confidentiality. While our current simulation system uses a simplified communication approach discussed in the "8.3.1 Networking" section, a real-world implementation would require a more robust data propagation method, particularly since marine radars typically lack direct internet access. This would involve implementing a relay system where data propagates from radar to radar until reaching the nearest unit with a direct connection to the central server. Currently, there is no implementation for secure wireless communication; however, a critical improvement will be to encrypt radar data both during transmission and in storage within the database to safeguard against potential security threats. This would involve implementing secure protocols like TLS for data transmission and encryption

standards such as AES-256 for database storage. Additionally, introducing reflectivity characteristics for various materials within the simulation is advised. At present, all materials are treated uniformly, meaning the radar reflections do not vary based on the entity's surface properties. The simulation also lacks variety in weather conditions, which significantly impact radar performance in real-world scenarios. By accounting for material-specific reflectivity and diverse weather patterns, we can achieve more realistic radar interactions, further enhancing the simulation's fidelity.

There remain areas for future work and refinement. While our current focus is on a simulation-based proof of concept, transitioning to real-world implementation necessitates hardware integration and further research. Several critical aspects lie beyond the scope of our current project but are essential for the system's success. These include:

1. Power delivery solutions for buoys deployed in remote maritime areas.
2. Mitigation strategies for radar-radar interference.
3. Development of robust data fusion techniques to handle overlapping radar signals.
4. Implementation of longevity mechanisms to safeguard hardware against environmental damage and wear.

References

[1] United States Army, "OE Threat Assessment: United Arab Emirates," TRADOC G-2 Intelligence Support Activity (TRISA), Complex Operational Environment and Threat Integration Directorate (CTID), December 2012.

[2] A. N. Ince, E. Topuz, E. Panayirci, and C. Işik, *Principles of Integrated Maritime Surveillance Systems*, 1998. doi:10.1007/978-1-4615-5271-0.

[3] S. Rudys *et al.*, “Investigation of UAV detection by different solid-state marine radars,” *Electronics*, vol. 11, no. 16, Aug. 2022. doi:10.3390/electronics11162502.

[4] T. Tagarev, K. Atanasov, V. Kharchenko, and J. Kacprzyk, Eds., "Maritime Surveillance and Information Sharing Systems for Better Situational Awareness on the European Maritime Domain: A Literature Review," in Digital Transformation, Cyber Security and Resilience of Modern Societies, I. Tikanmäki, J. Räsänen, H. Ruoslahti, and J. Rajamäki, Eds., ch. 8, pp. 117-135, Mar. 2021. doi: 10.1007/978-3-030-65722-2.

[5] B. Şengül, F. Yılmaz and Ö. Uğurlu, "Safety–Security Analysis of Maritime Surveillance Systems in Critical Marine Areas," *Sustainability,* vol. 15, *(23),* pp. 16381, 2023. doi: 10.3390/su152316381.

[6] S. Sotirov, A. Chavdar. “Improving AIS reliability”, presented at the IAMU Annual Conference, Varna, Bulgaria, (2017).

[7] J. Wang, K. Zhou, W. Xing, H. Li, and Z. Yang, “Applications, evolutions, and challenges of drones in Maritime Transport,” *Journal of Marine Science and Engineering*, vol. 11, no. 11, Oct. 2023. doi:10.3390/jmse11112056.

[8] G. Soldi et al., "Space-Based Global Maritime Surveillance. Part I: Satellite Technologies," in IEEE Aerospace and Electronic Systems Magazine, vol. 36, no. 9, pp. 8-28, 1 Sept. 2021, doi: 10.1109/MAES.2021.3070862.

[9] M. Johnsson and L. Bergman, “Radar and sea clutter simulation with Unity 3D game engine,” thesis, 2023.

[10] K. Vasstein, E. Brekke, R. Mester, and E. Eide, “Autoferry Gemini: a real-time simulation platform for electromagnetic radiation sensors on autonomous ships,” *IOP Conference Series. Materials Science and Engineering*, vol. 929, p. 012032, Nov. 2020, doi: 10.1088/1757-899x/929/1/012032.

[11] S. A. Davar, T. Chemander, J. Laurenius, and P. Tibom, “Virtual Generation of Lidar Data for Autonomous Vehicles,” dissertation, 2017.

[12] Z. Yun and M. F. Iskander, "Ray Tracing for Radio Propagation Modeling: Principles and Applications," in IEEE Access, vol. 3, pp. 1089-1100, 2015, doi: 10.1109/ACCESS.2015.2453991.

[13] B. G. Leite, M. Pereira, E. Szilagyi, and E. A. Tannuri, “Low-Fidelity Radar Implementation for Real-Time Ship Manoeuvring Simulator with Unity3D,” *TransNav*, vol. 17, no. 4, pp. 945–951, Jan. 2023, doi: 10.12716/1001.17.04.21.

[14] M. Y. Martin, S. Winberg, M. Y. A. Gaffar, and D. Macleod, “The design and implementation of a ray-tracing algorithm for signal-level pulsed radar simulation using the NVIDIA® OptiXTM engine,” *Journal of Communications*, pp. 761–768, Jan. 2022, doi: 10.12720/jcm.17.9.761-768.

[15] X. Mou, X. Chen, J. Guan, B. Chen and Y. Dong, "Marine Target Detection Based on Improved Faster R-CNN for Navigation Radar PPI Images," *2019 International Conference on Control, Automation and Information Sciences (ICCAIS)*, Chengdu, China, 2019, pp. 1-5, doi: 10.1109/ICCAIS46528.2019.9074588.

[16] N. Su, X. Chen, J. Guan and Y. Huang, "Maritime Target Detection Based on Radar Graph Data and Graph Convolutional Network," in *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1-5, 2022, Art no. 4019705, doi: 10.1109/LGRS.2021.3133473.

[17] X. Chen, J. Guan, X. Mu, Z. Wang, N. Liu, and G. Wang, “Multi-Dimensional Automatic detection of scanning radar images of marine targets based on radar PPINet,” *Remote Sensing*, vol. 13, no. 19, p. 3856, Sep. 2021, doi: 10.3390/rs13193856.

[18] X. Mou, X. Chen, J. Guan, Y. Dong, and N. Liu, “Sea clutter suppression for radar PPI images based on SCS-GAN,” *IEEE Geoscience and Remote Sensing Letters*, vol. 18, no. 11, pp. 1886–1890, Nov. 2021, doi: 10.1109/lgrs.2020.3012523.

[19] J. Wang and S. Li, "Maritime Radar Target Detection in Sea Clutter Based on CNN With Dual-Perspective Attention," in IEEE Geoscience and Remote Sensing Letters, vol. 20, pp. 1-5, 2023, Art no. 3500405, doi: 10.1109/LGRS.2022.3230443.

[20] X. Chen, X. Mu, J. Guan, L. Ningbo, and W. Zhou, “Marine target detection based on Marine-Faster R-CNN for navigation radar plane position indicator images,” *Frontiers of Information Technology & Electronic Engineering/Frontiers of Information Technology & Electronic Engineering*, vol. 23, no. 4, pp. 630–643, Apr. 2022, doi: 10.1631/fitee.2000611.

[21] X. Chen, J. Guan, Z. Wang, H. Zhang and G. Wang, "Marine Targets Detection for Scanning Radar Images Based on Radar- YOLONet," 2021 CIE International Conference on Radar (Radar), Haikou, Hainan, China, 2021, pp. 1256-1260, doi: 10.1109/Radar53847.2021.10028264.

[22] N. Su, X. Chen, J. Guan, Y. Huang, X. Wang and Y. Xue, "Radar Maritime Target Detection via Spatial–Temporal Feature Attention Graph Convolutional Network," in IEEE Transactions on Geoscience and Remote Sensing, vol. 62, pp. 1-15, 2024, Art no. 5102615, doi: 10.1109/TGRS.2024.3358862.

[23] W. Jiang, Y. Ren, Y. Liu, and J. Leng, “Artificial neural networks and deep learning Techniques Applied to radar target Detection: a review,” *Electronics*, vol. 11, no. 1, p. 156, Jan. 2022, doi: 10.3390/electronics11010156.

[24] “Estimation of Radar Cross Sectional Area of Target using Simulation Algorithm,” *International Journal of Research Studies in Electrical and Electronics Engineering*, vol. 4, no. 2, 2018, doi: https://doi.org/10.20431/2454-9436.0402003.

[25] T. Gao and J. Zhu, "A Survey of Procedural Content Generation of Natural Objects in Games," 2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Jeju Island, Korea, Republic of, 2022, pp. 275-279, doi: 10.1109/ICAIIC54071.2022.9722677.

[26] Sharjah24, “Extreme rainfall in Khorfakkan,” *Extreme rainfall in Khorfakkan*, 2024. https://sharjah24.ae/en/Articles/2024/02/12/Extreme-rainfall-in-Khorfakkan (accessed Sep. 28, 2024).

[27] G. Chance *et al.*, “On determinism of game engines used for Simulation-based Autonomous Vehicle Verification,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 20538–20552, Nov. 2022. doi:10.1109/tits.2022.3177887.

[28] USCG Budget, https://www.uscg.mil/budget/