RADAR MODELING FOR AUTONOMOUS VEHICLE SIMULATION ENVIRONMENT USING OPEN SOURCE

by

Tayabali Akhtar Kesury

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Mechanical Engineering

Department of Mechanical and Energy Engineering

Indianapolis, Indiana

May 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Sohel Anwar, Chair

Department of Mechanical and Energy Engineering

Dr. Andres Tovar

Department of Mechanical and Energy Engineering

Dr. Lingxi Li

Department of Electrical and Computer Engineering

Approved by:

Dr. Jie Chen

Dedicated to my friends and family and everyone who has helped me in my endeavors.

ACKNOWLEDGMENTS

I would first like to thank my advisor Dr. Sohel Anwar for his guidance, support and encouragement not only through the course of this research project but also during my entire tenure at the Purdue School of Engineering, IUPUI. I thank him for providing me an opportunity to work on this thesis. His insight and patience were instrumental in the understanding of this project. I would also like to thank the committee members Dr. Andres Tovar and Dr. Lingxi Li for their prospective comments and feedback.

I express my sincere gratitude to my research colleague Chris Orlin Cardoza for his valuable input throughout this research.

I would like to thank my father Mr. Akhtar Kesury for his teachings and support throughout my endeavors. I am grateful to my colleagues Omkar Parkar, Chris Orlin Cardoza, Anas Patankar and Swaraj Naskar who helped me write this thesis in a more detailed and comprehensive way.

Lastly, I would like to express profound gratitude to my family and friends for their unwavering support and continuous encouragement throughout my years of education and research. This accomplishment would not have been possible without my family, friends, educators, and mentors. Thank You.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Hz	Hertz
dB	Decibels
cm	Centimeters
m	Meters

ABBREVIATIONS

UE	Unreal Engine
HEV	Hybrid Electric Vehicles
LiDAR	Light Detection and Ranging
RADAR	Radio Detection and Ranging
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Application with Noise
ADAS	Advanced Driver Assistance System
PSNR	Peak Signal to Noise Ratio
SNR	Signal to Noise Ratio
GAN	Generative Adversarial Network
SSIM	Structural Similarities
RAM	Random Access Memory
CPU	Central Processing Unit
GPU	Graphic Processing Unit
ENV	Environment.
FOV	Field of View

ABSTRACT

Advancement in modern technology has brought with it an advent of increased interest in self-driving. The rapid growth in interest has caused a surge in the development of autonomous vehicles which in turn brought with itself a few challenges. To overcome these new challenges, automotive companies are forced to invest heavily in the research and development of autonomous vehicles. To overcome this challenge, simulations are a great tool in any arsenal that's inclined towards making progress towards a self-driving autonomous future. There is a massive growth in the amount of computing power in today's world and with the help of the same computing power, simulations will help test and simulate scenarios to have real time results. However, the challenge does not end here, there is a much bigger hurdle caused by the growing complexities of modelling a complete simulation environment. This thesis focuses on providing a solution for modelling a RADAR sensor for a simulation environment. This research presents a RADAR modeling technique suitable for autonomous vehicle simulation environment using open-source utilities. This study proposes to customize an onboard LiDAR model to the specification of a desired RADAR field of view, resolution, and range and then utilizes a density-based clustering algorithm to generate the RADAR output on an open-source graphical engine such as Unreal Engine (UE). High fidelity RADAR models have recently been developed for proprietary simulation platforms such as MATLAB under its automated driving toolbox. However, open-source RADAR models for open-source simulation platform such as UE are not available. This research focuses on developing a RADAR model on UE using blueprint visual scripting for off-road vehicles. The model discussed in the thesis uses 3D pointcloud data generated from the simulation environment and then clipping the data according to the FOV of the RADAR specification, it clusters the points generated from an object using DBSCAN. The model gives the distance and azimuth to the object from the RADAR sensor in 2D. This model offers the developers a base to build upon and help them develop and test autonomous control algorithms requiring RADAR sensor data. Preliminary simulation results show promise for the proposed RADAR model.

1. INTRODUCTION

Autonomous vehicle navigation has a number of benefits, including reducing driver fatigue and increasing the overall safety and efficiency of the operation. The robustness required for a fully autonomous vehicle necessitates the testing and validation of these autonomous navigation algorithms. A realistic simulator that can accurately estimate vehicle conditions as in real-world situations can be used to test and validate the proposed algorithm. Several simulators of this type have been built over the years [1]. Many factors in these simulators are customizable, allowing developers to generate a wide range of scenarios based on real-world events such as rain, dirt, sunlight, and snow. Unreal Engine, an open-source game engine with established functionality for creating a realistic simulated environment, can also accept Geographic Information System (GIS) data to produce 1:1 scale terrain.

However, a survey of the available open-source simulators reveals that RADAR (Radio Detection and Ranging) sensor modeling has received minimal attention. While these simulators have LiDAR (Light Detection and Ranging) sensor models, they do not include a RADAR sensor model. High-fidelity RADAR models are already accessible in private simulation tools like MATLAB's automated driving toolbox, but not in open-source simulators like AirSim, necessitating the construction of suitably realistic automobile RADAR sensor models.

A LiDAR, on the other hand, is more susceptible to weather conditions such as rain and dust. RADAR, on the other hand, operates with frequency modulated continuous wave (FMCW), which is unaffected by these conditions but susceptible to a different set of environmental variables, such as temperature. Despite the fact that both sensors are influenced by their unique object detection distortions, their high-level functionality is equivalent [2]. As a result, we were able to create a LiDAR sensor model to replicate RADAR detection while also giving it with a suitable range.

A RADAR sensor's range is also often greater than that of a LiDAR sensor. Rain and dirt can damage the accuracy of LiDAR and camera sensors, hence a RADAR sensor is required. The suggested RADAR model is created by first applying appropriate field of

view constraints to LiDAR point clouds and then expanding the RADAR range. The point cloud data is then subjected to a density-based clustering method in order to replicate a RADAR signal in the form of a point cluster using DBSCAN to display the objects in a python window.

1.1 Motivation

Given the Problem its is noted that from the review of the open-source simulators available finds that RADAR (Radio Detection and Ranging) sensor modeling has gotten very little attention. These simulators incorporate LiDAR (Light Detection and Ranging) sensor models but not RADAR sensor models. Private simulation tools like MATLAB's automated driving toolbox already have high-fidelity RADAR models, while open-source simulators like AirSim don't, necessitating the creation of sufficiently realistic automotive RADAR sensor models. A LiDAR, on the other hand, is more subject to environmental factors like rain and dust. RADAR, on the other hand, uses frequency modulated continuous wave (FMCW), which is unaffected by these factors but subject to another set of variables, such as temperature. Even though both sensors are affected by their own object detection distortions, their high-level functioning is identical. As a result, we were able to construct a LiDAR sensor model that could imitate RADAR detection while also having a good range. A wide variety of sensors are incorporated in autonomous vehicles, with the sole purpose of making the vehicle safer and prepared for any unpredictable circumstances. Although the current variety of sensors do well, there is still room for improvement in terms of testability [1], [3].

1.2 Background

Recent development in autonomous vehicle instrumentation has led to advancements in developments in AI and machine learning for complete autonomous driving. For these types of systems, performance is dependent on rigorous testing in various condition to make the system robust. Major perception sensors responsible to have a complete autonomous future are Cameras, RADAR and LiDAR.

Camera – Camera's work on the premise of focusing light emitted by objects onto a photosensitive surface via a lens. The image plane, also known as the photosensitive surface, is where light is stored as an image. After light travels through the lens, the light sensitive surface converts the rays into electrons, which are then converted to voltage, amplified, and finally transformed to a number in a metric unit or pixels by an Analog-Digital Converter (ADC) [1].

LiDAR – is also known as Light Detection and Ranging. It is a technology which detects the surface of the objects, their co-ordinates, and the exact size of the object. LiDAR's have now been developed to have resolution as close to a camera have range precision close to 2cm and have exact co-ordinate data. RADAR – is also known as Radio Detection and Ranging is used to detect an object, its speed, and its location. RADAR uses radio waves that travel at a speed that is as close to the speed of light, so it is safe to say there is no major difference in the performance of the sensor.

RADAR and LiDAR are both necessary for the vision of having a complete autonomous driving future.

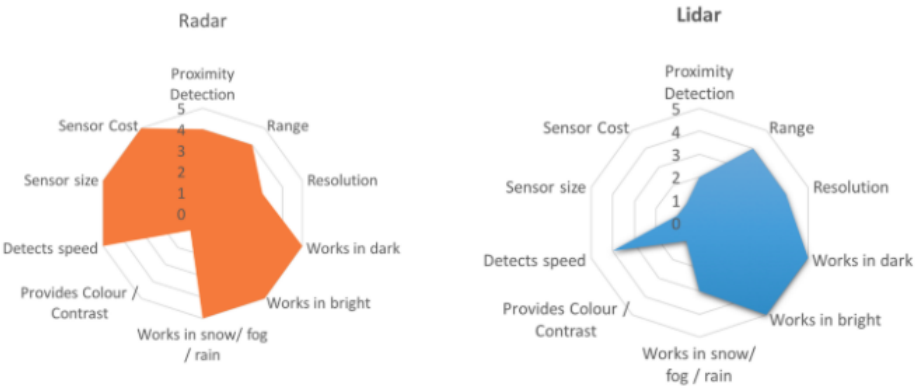

Figure 1.1. COMPARISON OF RADAR AND LiDAR [4]

Table 1.1. Differences between RADAR AND LiDAR.[4]

LiDAR	**RADAR**
We can detect little objects with a short wavelength. A LIDAR can create a 3D monochromatic image of an item that is extremely accurate. Disadvantages of Using LIDAR When it's dark or overcast outside, use it sparingly. The operating height is between 500 and 2000 meters. This is a high-priced technology.	In overcast weather and at night, RADAR can readily function. Operating distance is long. Use of RADAR also has Its Drawbacks Smaller objects cannot be detected at shorter wavelengths. Because of the longer wavelength, RADAR cannot deliver a precise image of an item to the user

1.3 Problem Identification

The thesis is addressing the problem of not having an open-source RADAR sensor model modular to work with multiple autonomous vehicle simulators. There are multiple Autonomous Simulators with RADAR sensor models according to the survey.[5]. But none of the RADAR sensor models address the problem of having a modular RADAR sensor model which could work with multiple simulators. [6].

The requirements for a RADAR sensor model are:

1. Good range and range rate accuracy and long detection range for vehicles
2. Position and velocity detection.
3. Close range clustering for tracking algorithms
4. Capable of working in multiple scenarios and simulators.
5. Flexible to tweak the parameters for different sensor configurations.

2. LITERATURE REVIEW

Several techniques have been presented to model RADAR accurately in the virtual environment. Ray tracing has been used to simulate FMCW automotive RADARs [7]. Here the reflected ray-traced signal was subjected to a pipeline involving FFT and IFTT to extract information about range and azimuth. A map of the power values for the RADAR range and azimuth also provides important information about targets in the environment. To go around the computationally intensive nature of FFTs, Fourier tracing has been used to capture the key features of environmental and automotive RADAR measurements [5], [7], [8]. Modeling a RADAR is still computationally intensive and new techniques are being developed to achieve real-time simulation. Deep Learning has also been used to ensure the real-time capability of the RADAR models in complex environments [5]. The proposed LiDAR point cloud-based RADAR model has the potential to overcome some of the stated limitations.

2.1 Measurements revealing Challenges in RADAR Sensor Modeling for Virtual Validation of Autonomous Driving

As development of perception systems play a crucial role in autonomous vehicles, the primary aim of this research is to conduct experiments to countermeasure the difficulties faced in RADAR sensor modeling. The paper also discusses about the fundamentals of RADAR sensors and emphasize on the shortcomings that are yet to be resolved.

The first experiment conducted by the researchers is 'Occlusion' also known as blocking. The scenario in this case is two cars are driving in the same lane in front of the ego car, causing the front-most car to be occluded (i.e. no direct Line of Sight) [9]. The focus of the test is to examine how blocking influences root cross section values and the number of reflection points. Researchers use ray tracing method to model occlusion. Rays are generated according to the measuring range of the sensor and each ray is propagated through the scene until it hits a surface or is terminated. At any hit-point, the reflected energy is computed and a bouncing ray is shot if the maximum ray depth has not yet been exceeded [9].

The second experiment shows about separability. It gives a gist of how RADAR can distinctly detect different objects that are close to each other. The main factors that affect separability are azimuth angle and range. The test conducted had several vehicles placed as close as possible to each other on a RADAR range radius of 25 meters. The frequency ramp of the RADAR used in this experiment covered a bandwidth of 2 GHz on the carrier frequency 77 GHz. The RADAR output was the beat signal (IF2) and signal processing was performed on a host PC. As a result, range azimuth maps indicate the location of normalized power levels in dB depending on range and azimuth direction [9]. The experiment indicates that separability can be a good performance metric for RADAR sensor models.

In the third experiment, the sensitivity of a car's RADAR cross section is studied using a real word test. The factors taken into consideration are heading and azimuth angle. The experiment consists of the ego car and a car that repeatedly performs dual lane changes. The radial distance was kept to 15 meters and both cars were maintained the same speed. Results of the experiment showed that slight variation in target distance and aspect angle can significantly affect the RADAR cross section which in turn will affect the sensitivity of the sensor model. Also, in this particular experiment environmental aspects were not considered therefore, there is a probability of the environmental factors to influence the sensitivity.

The research implies that despite its complexity, a modular simulation architecture of a RADAR sensor can be derived, where individual parts are (almost) independent. By subdividing a RADAR sensor model into environment simulation, wave-propagation and interaction, sensor hardware model and detection and tracking model, we aim to separate influencing factors while keeping modules interchangeable [9]. Also, the research emphasizes on how validation experiments are important and that a simulation environment is essential for the same.

2.2 A Generic Validation Scheme for real-time capable Automotive Radar Sensor Models integrated into an Autonomous Driving Simulator

The primary essence of this research is to develop and validate radar sensor models integrated into a driving simulator. The researchers mention few important ADAS features

widely used in today's vehicles that include radar sensors, namely, adaptive cruise control and autonomous emergency braking which is one of the essential ADAS feature today.

Further, authors describe some open-source and closed source simulators that use sensor model integration and are already in the simulation world for quite some time. They are as follows:

1. AirSim: Developed by Microsoft originally for autonomous flying devices, the simulator now includes driving applications as well. However, it is still under development for robustness in the autonomous driving application [10].

2. GTA V: Developed as a computer game, nevertheless, it is far more ahead in the physics implementation of a car when integrated with ScriptHook. Irrespective of the fact, GTA V lacks sensor emulation and multiple viewpoints and it cannot be used for commercial purposes.

3. CARLA Simulator: The most widely used open-source simulator for autonomous driving. It has the best autonomous driving algorithms built and are constantly upgrading to increase performance and robustness of the Simulator.

In the next section of the research, authors explain about the radar modeling methods they implemented based on two factors: Feasibility and robustness.

1. Physical models: Researchers initially faced challenging situations while developing a method for radar physics. However, they adopted the shooting and bouncing ray method which proved to be extremely useful in measuring wave propagation, signal detection and reflection. The authors claim that the physical models are robust despite being hard to implement.

2. Scattering Center models: They are easier to build and are moderately robust, i.e., they cannot be used in complex driving situations.

3. Data driven models: Also known as black-box model, these types of models purely feed on data provided to the algorithm. The feasibility depends upon the complexity

of the neural network. These models are robust however, they have low accuracy in some scenarios, this creates a drawback for data driven models as discussed in study.

For Validation, the authors integrated the models with CARLA simulator and built a validation architecture to check performance of different models. The scheme follows a real-world testing where radar detects a list of objects. The scenario is virtually recreated using simulator and synthetic radar data is obtained. This data is used in the real radar system to check performance.

The authors successfully generated results for three different scenarios (models). As a conclusion, the researchers claimed the validation scheme was successful and straightforward. They further claimed that integration of sensor model with simulator can be feasible and extremely useful to recreate real-world scenarios effectively.

2.3 LiDAR to RADAR Translation

The paper closely relates to the thesis topic as it consists of using LiDAR data to translate into RADAR model. In the research, researchers describe how despite being functional in longer ranges and extreme conditions, RADAR sensor modeling is still in its infancy using Deep Learning neural networks. On the other hand, LiDAR has a substantial progress in the deep learning sector.

The primary focus of this research is to develop a RADAR deep neural network model using LiDAR point clouds. Due to the scarcity of well-interpreted RADAR datasets, there is a limitation in RADAR for environmental perception. This motivated the researchers to define the LiDAR to RADAR translation. As the paper progresses, the researchers briefly discuss their contributions and related previous work. They first propose a conditional model called L2R GAN to translate LiDAR data into RADAR. To measure and demonstrate the effectiveness of the model, researchers conduct experiments to generate detail RADAR data. Also, they show that their framework can be used for advanced data augmentation and emergency scene generation by editing the appearance of objects [11].

The paper discusses about Cross-Domain data translation with the help of conditional GAN giving image-to-image translation as the fitting example. According to the researchers,

image-to-image translation can be supervised or unsupervised. In other words, supervised translation focuses on generating ground-truth images with minimal pixel-losses. Furthermore, pix2pix generator uses a U-Net architecture and the discriminator classifies patches (N x N) as real or fake. Also, previous work claims that using multi-scale generator can highly benefit GAN models. On the contrary, some researches use unsupervised way where random samples from different training data sets are used [11].

In the next section of the paper, a brief discussion on RADAR data representations is shown. Researchers explain about how RADAR is the widespread and best choice in the rapidly growing autonomous vehicle world. However, they shed light on few shortcomings such as complexities in RADAR data measurement, as well as contamination and/or distortions in the signals. According to previous works, increasing data abstraction and feature extraction classifies the RADAR data into three different representations:

1. Raw Polar RADAR Spectrum
2. RADAR Spectrum in Cartesian Coordinates
3. 3D RADAR points clouds and 2D RADAR pins [11].

The researchers define that the original RADAR data is in 2D array form and is not available to the users. Nevertheless, the above issue is countered by using Digital Signal Processing (DSP) algorithms such as Fast Fourier Transform (FFT) and Multiple Signal Classification (MUSIC) to obtain polar RADAR data. Also coordinate transformation from polar to cartesian, images can be altered to highest reflection.

For the next part, researchers implement the baseline for pix2pix and pix2pixHD to generate a high-resolution RADAR data/ spectrum from the LiDAR point cloud. The method is a framework for GAN to create image-to-image translation and adding loss. The objective function is defined and is trained to find the saddle point. However, due to range limitations in LiDAR, same range is used for both LiDAR image synthesis. Also, the baseline results do not meet the requirements due to the following drawbacks:

1. Difference in sensor characteristics, pixel-wise correspondence is not observed between LiDAR and RADAR

2. Black regions also known as free space occupy majority of the image area.

3. Pix2pix is designed for edge-guided scene, LiDAR images are unable to create border features.

4. LiDAR to RADAR should be one-to-one mapping which the framework is not able to do [11].

To overcome the black regions issue, researchers use an occupancy grid mask global generator in their framework. The occupancy mask is generated via ray casting through the scene, which is implemented using Bresenham's line rendering algorithm. Along with the global generator, the researchers also developed a local generator to obtain a near perfect RADAR spectrum. To extract region of interests from LiDAR point clouds effectively, researchers utilize the feature encoder network from PointPillars as an extraction network. This network is designed to convert a point cloud into a sparse pseudo-image [11].

Further, researchers conduct experiments to evaluate and compare their model to baselines. The objective function is optimized and since global and local generator require different set of parameters, training strategy used by the researchers is that if the accuracy of discriminator is above 75% training of the dataset is skipped. For training, robotcar dataset is used and quantities used for evaluation are Peak Signal to Noise Ratio (PSNR) and structural similarity (SSIM). As a qualitative metric researchers involve humans as subjects to read about RADAR and pick the real RADAR image from the results.

Experimental results indicate that losses do not improve the quality of local region instead it disrupts the training process. The research's conclusion is that the L2R GAN neural network model is significantly promising in generating RADAR images using global for large regions and local generators for small regions. Also, the model can be used to generate RADAR data for emergency situations such as pedestrian collision warning. The research is a good reference for further learning in ADAS applications.

2.4 Deep Stochastic RADAR Models

The literature aims to build a methodology for the development of stochastic RADAR models with the help of deep learning. The procedure followed by the researchers to develop deep RADAR model is as follows:

A. Inputs: Researchers use deep neural networks as RADAR models to learn the conditional probability distribution end-to- end from data. The inputs to the network are the spatial raster R and the object list O, and the output is the sensor reading Y. Spatial raster is a tensor quantity with two primary dimensions, azimuth and range [12].

B. Architectures: The paper establishes two methods to represent probability distribution with a neural network:

1. Direct Parameterization

2. Sampling using noise as auxiliary input

Based on the two methods, two baseline models are developed, Normal Model and Gaussian Mixture Model.

C. Implementation: The researchers use an encoder-decoder pipeline for development of the model. The encoder takes the raster and object list and produces a latent feature representation x. The decoder takes the feature representation and a randomly generated noise value and produces the predicted sensor measurement [12].

D. Loss Function: In order to reduce losses, Researchers developed a loss function to compensate for the same and added to the encoder calling it 'variational encoder'.

Data is collected by keeping in mind environmental factors such as terrain, weather and road conditions. Further, the deep RADAR models developed are evaluated with the help of the collected dataset and compared with the baseline models.

Results are displayed for the following quantities:

1. Expected Root Mean Square Error

2. RADAR Range Equation

3. Clutter Metrics

For all the above-mentioned metrics variational encoder shows significantly better results than Normal and Gaussian Mixture Model. The paper concludes that, Qualitative and quantitative evaluations against real-world data show that using conditional variational autoencoders trained with both autoencoder loss and adversarial loss produce the best predictions. The VAE is able to produce inter-correlated predictions, and the use of an adversarial discriminator prevents prediction averaging, resulting in more realistic predictions. Models were shown to exhibit power loss following the RADAR range equation and reproduce roadside clutter [12].

2.5 Modeling and Simulation of RADAR Sensor Artifacts for Virtual Testing of Autonomous Driving

The essence of this research is to develop and test mathematical models of the artifacts (distortion in sensor reading) that occur during RADAR measurements causing false detection. Additionally, the article gives a synopsis on the artifact formation process in RADAR processing. The list of artifacts mentioned in the paper is given below:

A. Mirror Reflections: Caused due to different surfaces, it disrupts the measurements of RADAR due to multi-directional propagation. In this type of artifact, the path combinations are classified into four types: indirect-direct, direct-indirect, indirect-indirect and direct. Out of the four, former three are false detections.

B. Aliasing: It causes due to undersampling of data reducing the frequency of the radio wave. The undersampling is a result of lenient sampling theorem embedded in the real-world sensor model which is a result of requiring high computing resources to process acquired data.

C. Limitation of Resolution: Various factors come into play viz., bandwidth, system design, range, azimuth angle, etc. that defines the accuracy of the RADAR measurement. Nevertheless, it cannot be concluded that an increase in the mentioned quantities will significantly increase a RADAR sensor's resolution.

D. Clutter/ Measurement Noise: occurs when detection threshold is exceeded.

E. Non-ideal signal processing causes due to erratic behavior of high-frequency components (nonlinearities in electronic components).

F. Object tracking artifacts: They continuously monitor target states based on noisy sensor readings. The approaches used to develop mathematical models of artifacts are phenomenological, i.e., the method includes data which is found by a not so optimal method although it gives a short-term solution. It is also known as heuristic technique Another method known as the casual method, where artifact development process is added to the sensor model [8]. The paper mainly focuses on the modeling of mirror reflection artifact.

For the mirror reflection artifact, the researchers create a scenario of multipath propagation using three vehicles, two of which are target vehicles who will reflect the RADAR signal back to the ego vehicle. All vehicles are traveling at different velocities. Using Doppler alteration, equation for relative velocity of the RADAR vehicle with respect to other vehicles for different path combinations is developed. The theory was verified and compared in graphs with the help of an actual test. The results showed, the actual mirror detection relative velocity matches with the conceptualization for indirect-indirect path.

In the case of aliasing, the wave reflects back and forth three times, relative velocity exceeds the range which gives unclear data. The method used for modeling limitation of resolution is the vector-projection method. It involves applying a blurring filter on a sharp reflection point [8]. To conclude, the researchers successfully developed and verified simulation models of the artifacts with real world testing. The article was able to bridge the gap between defining artifact and how its formation affects the sensor model.

2.6 Types of Clustering Algorithms

When selecting a clustering algorithm, think about whether it will scale to your dataset. Machine learning datasets can contain millions of samples, but not all clustering algorithms scale well. The similarity between all pairs of instances is computed by many clustering algorithms. In complexity notation, this means that their runtime grows as the square of the number of examples. When there are millions of examples, algorithms aren't realistic [13].

Centroid-based Clustering

In contrast to the hierarchical clustering described below, centroid-based clustering organizes data into non-hierarchical clusters. The most extensively used centroid-based clustering algorithm is k-means. The efficiency of centroid-based algorithms is limited by their sensitivity to initial conditions and outliers.

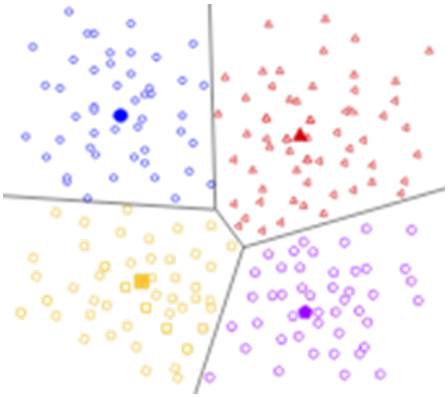

Figure 2.1. Centroid-based Clustering [13].

Density-based Clustering

Clusters of high example density are connected using density-based clustering. As long as dense areas can be connected, arbitrary-shaped distributions are possible. These algorithms struggle with data with a wide range of densities and dimensions. Furthermore, these techniques are not designed to allocate outliers to clusters.

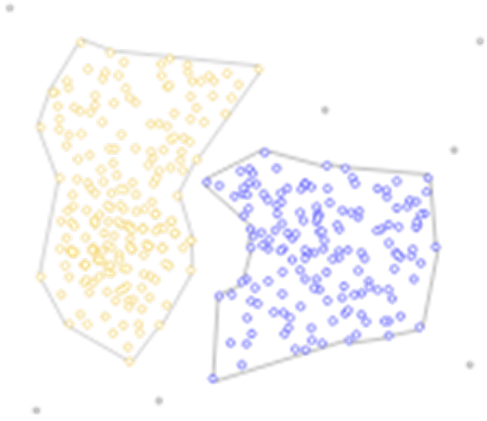

Figure 2.2. Density-based Clustering [13].

Distribution-based Clustering

This clustering method is based on the assumption that data is made up of distributions, such as Gaussian distributions. In Figure 2.3, the data is clustered into three Gaussian distributions using a distribution-based technique. The likelihood that a point belongs to the distribution diminishes as the distance from the distribution's center increases. The bands depict the likelihood drop. You should use a different algorithm if you don't know the type of distribution of your data.

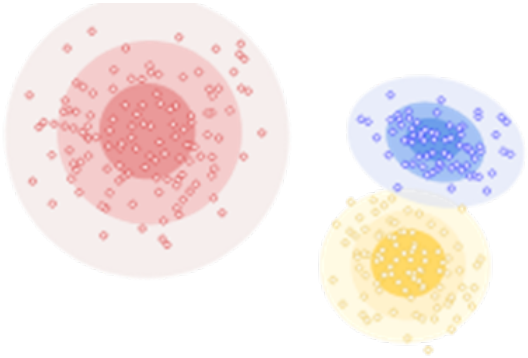

Figure 2.3. Distribution-based Clustering [13].

Hierarchical Clustering

Hierarchical clustering as shown in Figure 2.4, also known as hierarchical cluster analysis, is a method of grouping related objects into clusters. The endpoint is a collection of clusters, each of which is distinct from the others yet the items within each cluster are broadly similar.

Figure 2.4. Distribution-based Clustering [13].

2.7 OVERVIEW

To provide a solution to the problem identified above, the study is outlined as follows. The methodology is discussed below.

2.7.1 RADAR Sensor Model Development

As the identified problem is focused on developing a general RADAR sensor model to be deployed on various environments, a sensor model is developed using clustering methods discussed in chapter 3. The model is based on python, it will be serving as an addition to the LiDAR sensor models used in an autonomous vehicle simulation environment. The model is set up in such a way that it can work with any environment producing point cloud data. The sensor model is modular so the parameters can be tweaked according to the requirement of different RADAR sensor models. A driving environment of urban and off-road settings is for vehicle simulations.

2.7.2 Clustering Method adopted

DBSCAN

DBSCAN is a density-based method that assumes dense regions have clusters. It's also the first clustering technique we've looked at: it doesn't require every point to be allocated to a cluster, thus it doesn't split the data; instead, it extracts the 'dense' clusters and leaves the sparse background as 'noise.' DBSCAN is linked to agglomerative clustering in practice. DBSCAN alters the space in the first phase based on the data density: points in dense regions are left alone, while points in sparse regions are shifted further away. When single linkage clustering is applied to the changed space, we get a dendrogram, which we cut into clusters using a distance parameter (named epsilon or eps in many implementations). At that cut level, any singleton clusters are considered 'noise' and are left unclustered. This has multiple advantages: we obtain agglomerative clustering's manifold following behavior, as well as true clustering rather than partitioning. Better yet, because the method can be framed in terms of local region queries, we can leverage various tactics like kdtrees to achieve remarkable efficiency and scale to dataset sizes that would otherwise be unattainable with algorithms other than K-Means. However, there are a few caveats. Obviously, epsilon is a difficult parameter to choose; you can conduct some data analysis and make a fair guess, but the algorithm can be quite sensitive to the parameter selection. Another parameter (min samples in sklearn) influences the density-based transformation. Finally, the combination of min samples and eps results in a density decision, and clustering only detects clusters with that density or higher; if your data has varying density clusters, DBSCAN will either miss them, divide them up, or lump some of them together, depending on your parameter choices [14].

Table 2.1. Parameters used for DBSCAN.

epsilon	3
min_sample size	20

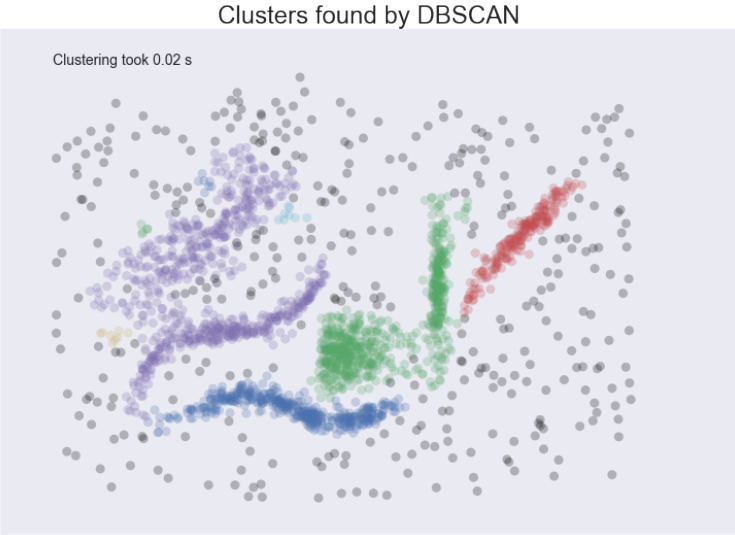

Figure 2.5. Representation of DBSCAN [14]

3. METHODOLOGY

Multiple sample environments were created on Unreal Engine, the selected open-source simulator with an editor to build various simulation environments. It can provide photorealistic visuals, dynamic physics, and effects, as well as reliable data translation, allowing for realistic vehicle simulation. It may also be changed according to the developer's preferred configuration, which is an important factor to consider while choosing Unreal Engine [15]. Unreal Engine has a blueprint visual scripting system that is based on the concept of using a node-based interface to create elements in the gameplay within Unreal Editor. The LiDAR model in the Microsoft AirSim [2] was used as the baseline model to develop our LiDAR model as an Unreal blueprint. This allowed more control over the LiDAR model's parameters and eliminate some of the overhead computational costs that AirSim requires. The LiDAR blueprint uses Unreal's 'LineTraceByChannel' function to send out rays from a specified location. The blueprint controls the number of rays that are sent out per time tick, the duration of the time tick, and the vertical and horizontal range of the emitter. When the line trace collides with a Game Object in the Unreal environment, it returns the location of the hit. The normalized distance to the collision is used to interpolate between two colors. The hit points are then drawn in the unreal environment and colored using the distance from the emitter to the points. A flow chart illustrating this model is explained in Figure 3.2. The LiDAR data is imported in the form of an a.vrml file which consists of the x, y, and z coordinates of the points which strike an object in the world.

3.1 Clustering Algorithms Requirements

An ideal clustering algorithm for RADAR modeling should have the following features.

1. First it should be able to differentiate between many different objects - in clustering terms closely packed different clusters.

2. The number of objects is not known in advance as it depends on the environment and the resolution of the RADAR. Therefore, the model should have robust tuning parameters for a wide variety of scenarios.

3. The clustering algorithm should be able to produce the core points of the cluster.

Two clustering algorithms were considered:

1. K-means-based clustering - K-means clustering assigns every data point to the nearest centroid, which are K separate randomly initiated points in the data. The centroid is shifted to the average of all the points assigned to it once each point has been assigned. The process is then repeated: each point is allocated to the centroid closest to it, and centroids are shifted to the average of the points assigned to them. When no point's assigned centroid changes, the method is complete [14].

2. Density-based clustering algorithm - DBSCAN is a density-based approach that assumes clusters exist in dense locations. It's also the first clustering technique we've looked at: it doesn't require every point to be assigned to a cluster, so the data isn't split; instead, it extracts the 'dense' clusters while leaving the sparse background as 'noise.' In practice, DBSCAN is tied to agglomerative clustering [14].

3. HDBSCAN - Campello, Moulavi, and Sander created HDBSCAN, a clustering technique. It improves on DBSCAN by transforming it into a hierarchical clustering algorithm and then extracting a flat clustering based on cluster stability.[13]

However, the K-means algorithm did not produce desired output because it could not distinguish between local variations within a cluster. On the other hand, the density-based clustering (DBSCAN) algorithm, shown in Figure 3.1, could perform well in forming non-linear clusters and took less computational time which is suitable for real-time applications. Hence this method was selected for the proposed RADAR model. Another potential density-based clustering algorithm is HDBSCAN. DBSCAN has two parameters: epsilon and minimum number of samples whereas HDBSCAN needs only the minimum number of samples. One advantage of DBSCAN over HDBSCAN is that it offers control over the minimum number of points. Since HDBSCAN uses variable epsilon, it can produce clusters of varying densities resulting in more clusters which can, in an off-road environment, also be a bush or just a small bump on the ground. In DBSCAN, epsilon specifies the radius of a neighborhood with respect to the point and minimum number of samples determines the minimum number of

points required to be considered as a cluster [16]. The x, y, and z coordinates from the point cloud data are used to create a data frame which, in turn, is used to produce clusters. The point data is in a three-dimensional format; however, RADAR output is in a two-dimensional data format. Accordingly, the dimensionality of the data frame is changed to produce two components on which the clustering operation is performed.

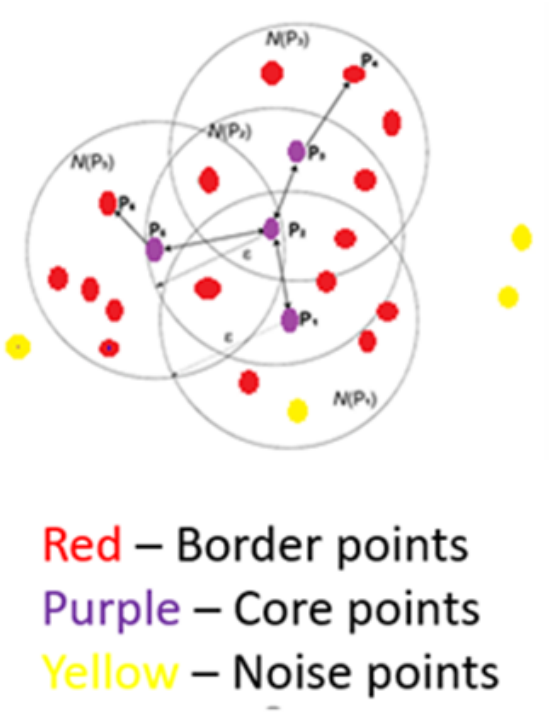

Figure 3.1. Visual representation of dbscan [3]

The clusters then used to produce the core points using the 'core_indices' function. The core points are plotted showing the output in a RADAR output format, ignoring the outlier points which do not form a cluster. Figure 3.2 shows the flowchart of this clustering algorithm using DBSCAN.

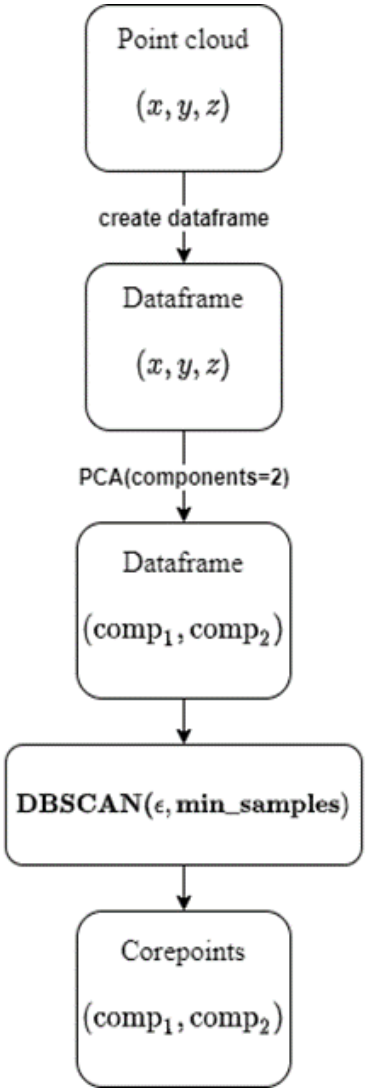

Figure 3.2. Flow chart of Clustering Algorithm

This figure shows a flow chart of the clustering performed on the clipped point cloud data, as seen a data frame is extracted from the pointcloud data. The 3D data frame is then converted to a 2D data frame and then clustering is performed on the 2D data frame giving us two components. From this component the centroids of the objects are extracted and plotted in the python window.

3.2 LiDAR MODEL

The LiDAR system uses Unreal's LineTraceByChannel function to send out a ray from a specified location. This is done 10,800 times ranging 30 degrees vertically and 360 degrees horizontally. When the line trace collides with a Game Object in the Unreal environment, it returns the location of the hit. The normalized distance to the collision is used to interpolate between two colors. The points are drawn in the unreal environment with a visual indication of the distance.

To send out rays from a specific place using Unreal's 'LineTraceByChannel' function. The blueprint determines the number of rays sent out per time tick, as well as the duration of the time tick and the emitter's vertical and horizontal range. The position of the hit is returned when the line trace collides with a Game Object in the Unreal environment. To interpolate between two colors, the normalized distance to the collision is employed. The hit points are then colored based on the distance between the emitter and the points in the unreal environment. Figure 3.2 shows a flow chart that explains this model. An a.vrml file containing the x, y, and z coordinates of the points that strike an item in the world is used to import LiDAR data. Table 3.1 Shows the parameters of the LiDAR which can be set as per user preference.

Table 3.1. LIDAR parameters.

Parameter	**Description**
Number_of_channels	Number of lasers of LiDAR
Range	Range in Meters
Points_per_second	Number of points conceded per second
Rotation_per_second	Number of rotations per second.
HFOV	Horizontal field of View of LiDAR
VFOV	Vertical field of view of the LiDAR
X Y Z	Position of LiDAR relative to the vehicle (in meters)
Roll Pitch Yaw	Orientation of LiDAR relative to the vehicle (in degrees)

Table 3.2. Proposed LIDAR parameters.

Parameter	**Description**
Number_of_channels	16
Range	80
Points_per_second	10000
Rotation_per_second	10
HFOV	-45,+45
VFOV	-4,4
X Y Z	(0,0,-1)
Roll Pitch Yaw	(0,0,0))

Table 3.2 Shows the proposed LiDAR parameters for the RADAR model, these parameters would give the clipped data according to the FOV of the RADAR.

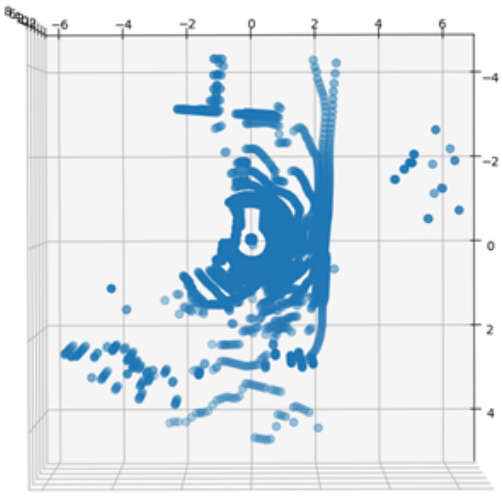

Figure 3.3. Flow chart of clustering algorithm

Figure 3.3 Shows the sample LiDAR data which is generated if the point cloud data is not clipped according to the FOV of the RADAR. As seen the LiDAR gives a much denser point cloud which is at much higher resolution.

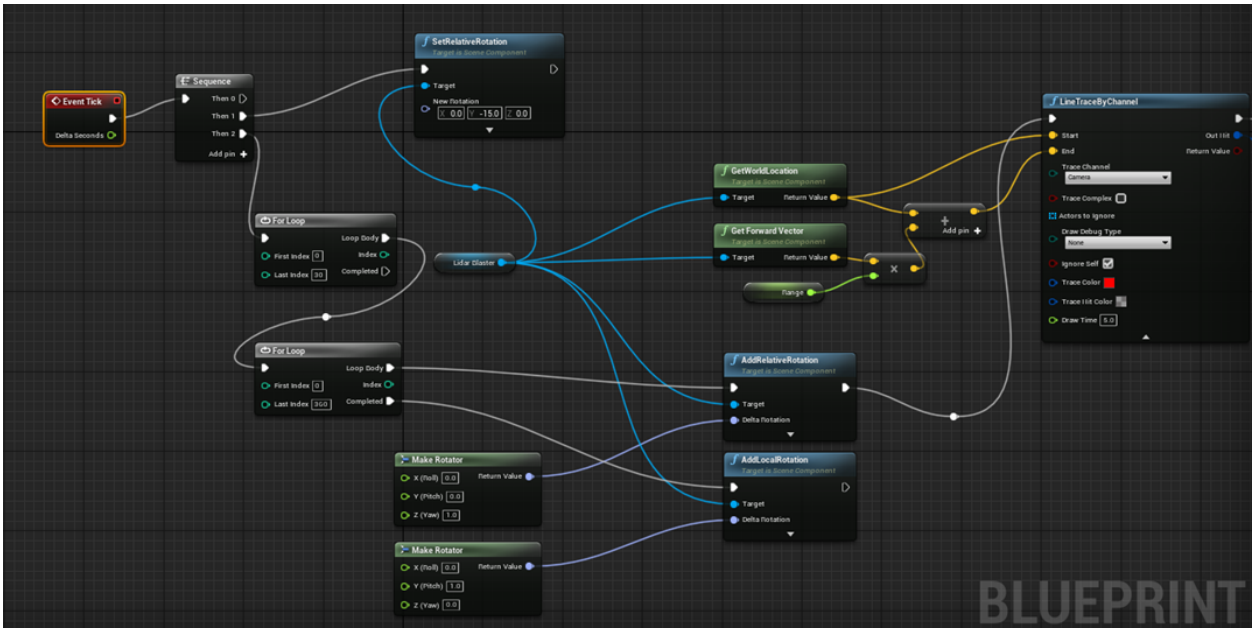

Figure 3.4. LiDAR BLUEPRINT from UE PART1. This picture shows PART 1 of the LiDAR model in Unreal Engine.

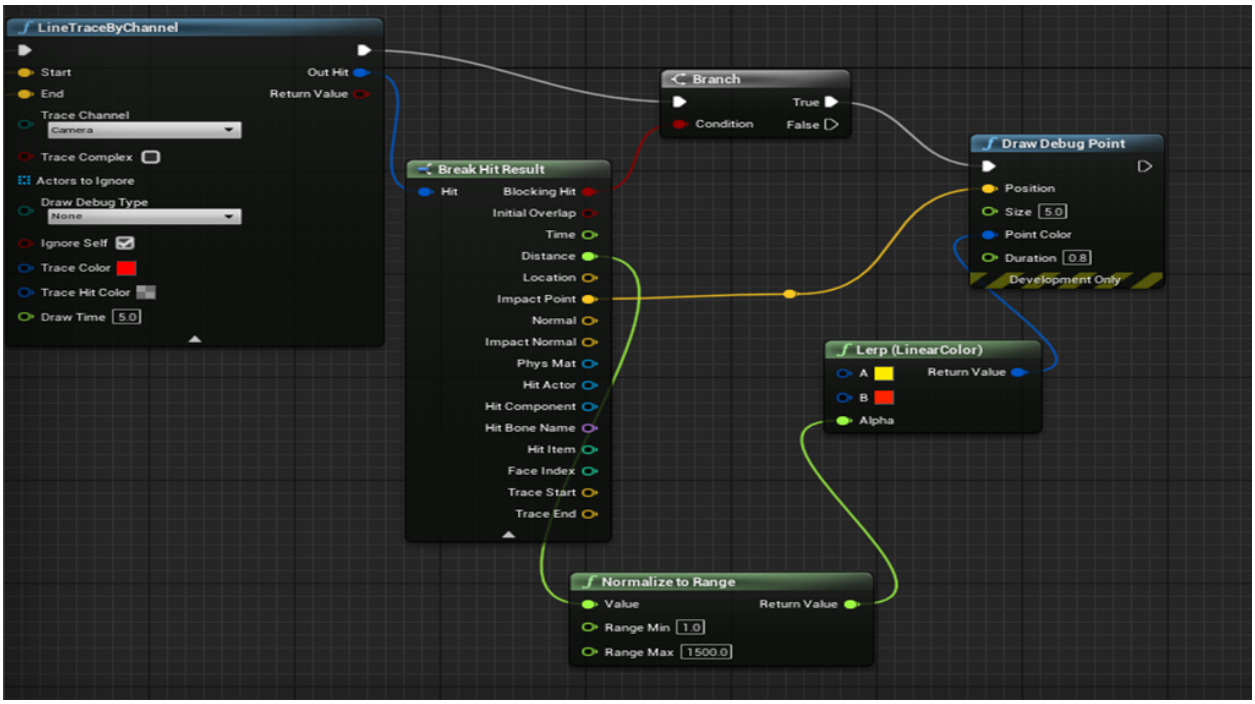

Figure 3.5. LIDAR BLUEPRINT from UE PART2. The above picture shows the LiDAR model developed on UE and how it is displayed in the environment.

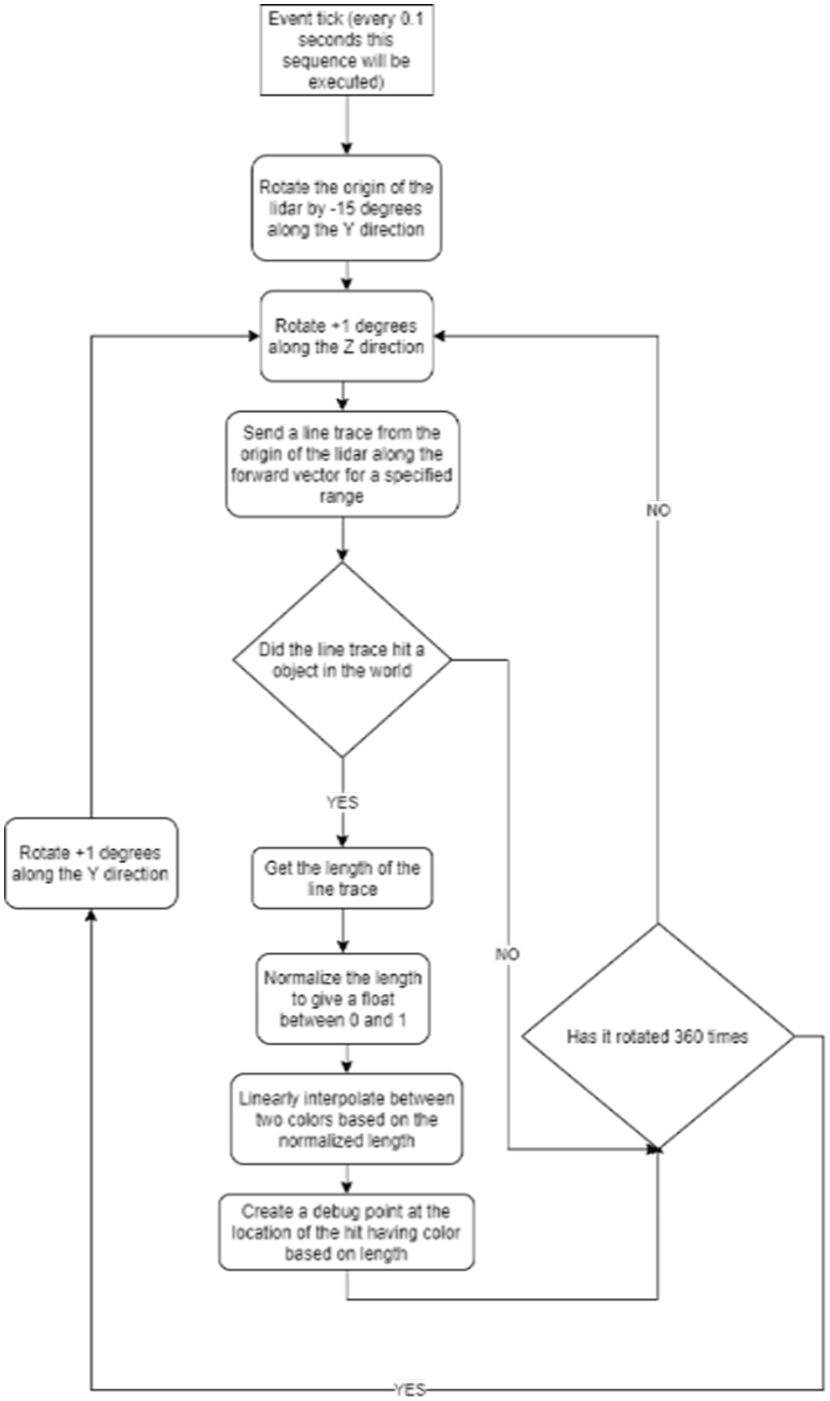

Figure 3.6. FLOW CHART OF LiDAR SCANNER. Shows the flow of the LiDAR model developed on Unreal Engine (UE).

3.3 RADAR MODEL

RADAR sensors work on the premise of echoes being reflected. Electromagnetic waves are emitted by a transmitter and reflected back from the surfaces of adjacent objects [17]. The distance to the object from the sensor is calculated using the speed, distance, and time relationship after the reflected waves are registered by the receiver.

$$D = c * T/2 \tag{3.1}$$

Where, D: Distance to an object c: 3x108 m/s, speed of propagation of electromagnetic waves T: Time delay, from emission to reflection to being received

Electromagnetic waves are susceptible to attenuation by other radio waves in the environment, lowering measurement accuracy. Extreme weather conditions can also have a minor impact on RADAR performance, especially if foreign particles block the receiver or transmitter [17]. Rain and snow, for example, generate echo signals that can obscure the desired target echoes [17]. A RADAR also can't tell the difference between little things in the way of the radio waves and the targeted targets. Interference from nearby RADAR or other transmission sources can also cause reflected echoes to be distorted, reducing measurement accuracy. However, in comparison to LiDAR, a RADAR is a less expensive component that does not require a lot of processing power.

In the proposed RADAR model, In the horizontal plane, the LiDAR data spans 360 degrees. A RADAR, on the other hand, has a defined horizon that is related with the type of RADAR (e.g. short range, medium range). We used a medium-range automotive-grade RADAR with a horizontal field of view of 45 degrees for this study. As a result, the LiDAR point cloud data is cropped to only a 45-degree viewing angle of the vehicle's front. The dimensionality of the clipped data is then reduced from three to two dimensions. To create the simulated RADAR clusters, the two components of the data frame are clustered using the point density-based clustering algorithm DBSCAN [16].

In the first phase, DBSCAN changes the space based on the data density: points in dense areas are left alone, while points in sparse areas are relocated further away. We get a dendro-

gram when single linkage clustering is performed to the altered space, which we divide into clusters using a distance parameter (named epsilon or eps in many implementations). Any singleton clusters are considered 'noise' at that cut level and are left unclustered. However, there are a few caveats. Obviously, epsilon is a difficult parameter to choose; you can conduct some data analysis and make a fair guess, but the algorithm can be quite sensitive to the parameter selection. Another parameter (min_samples in sklearn) influences the density-based transformation. Finally, the combination of min samples and eps results in a density decision, and clustering only detects clusters with that density or higher; if your data has varying density clusters, DBSCAN will either miss them, divide them up, or lump some of them together, depending on your parameter choices

Table 3.3. Test parameter for radar model.

Trial	**min_samples**	**epsilon**
1	50	16
1	50	0.5
2	30	1
3	20	3
4	20	2
5	10	2
6	100	2
7	70	2
8	70	3
9	100	5
10	100	10

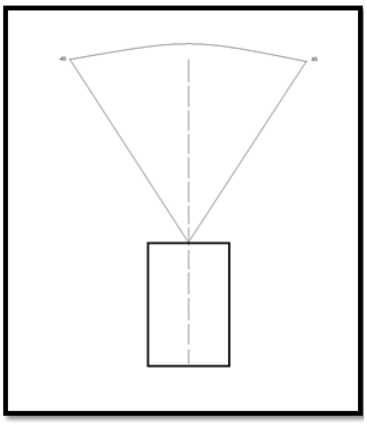

Figure 3.7. Horizontal FOV of Radar. This Figure shows how the Pointcloud was clipped to a FOV of 90 degrees from a 360 degree point cloud generated by the LiDAR.

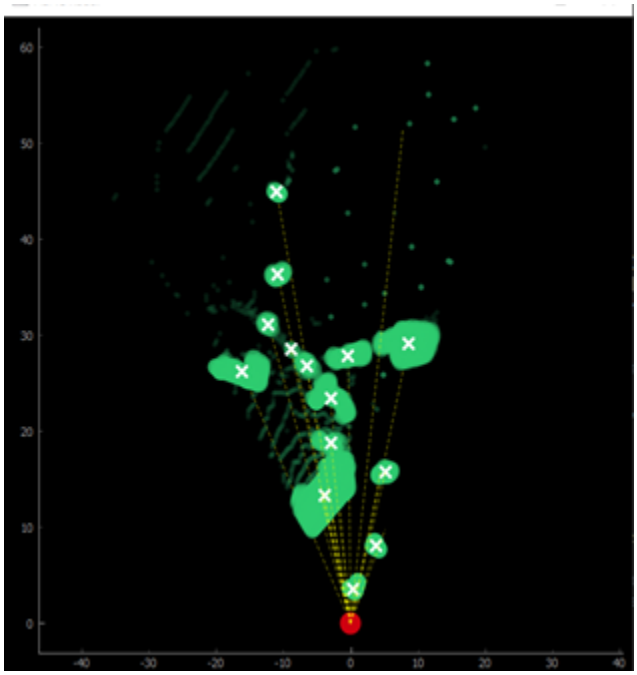

Figure 3.8. EXAMPLE OF SIMULATED RADAR. The figure above shows a simulation sample of the RADAR sensor model.

4. SIMULATION EXPERIMENTAL SETUP

4.1 Unreal Engine

Unreal Engine 4 (UE4) is a set of development tools for video games, architectural and automotive visualization, linear film and television content creation, broadcast and live event production, training and simulation, and other real-time applications.

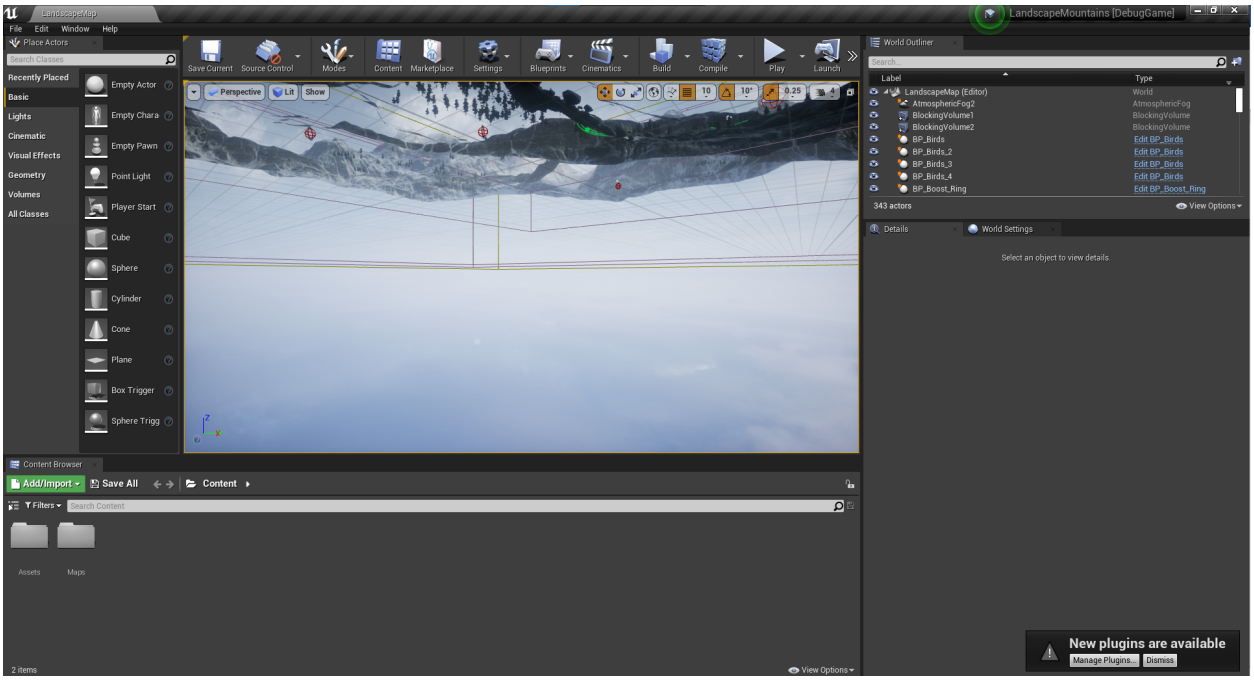

Figure 4.1. Unreal Engine (UE) Environment VSCODE Environment.

Figure 4.1 Shows an example of the UE environment which has multiple game objects which can be modified according to the scenario we need. It has many environments available and are open source with the source code available on GitHub. For this particular environment we had multiple game objects depicting a natural landscape and some foliage in it which is an actor in Unreal Engine.

4.2 Python Environment

The RADAR model is developed on Python 3.8 using the following python libraries:

1. PYQT

2. SKLEARN

3. NUMPY

4. SCIPY

5. PANDA

Any IDE can be used to deploy the python environment, for this case VSCODE was used as it is open source

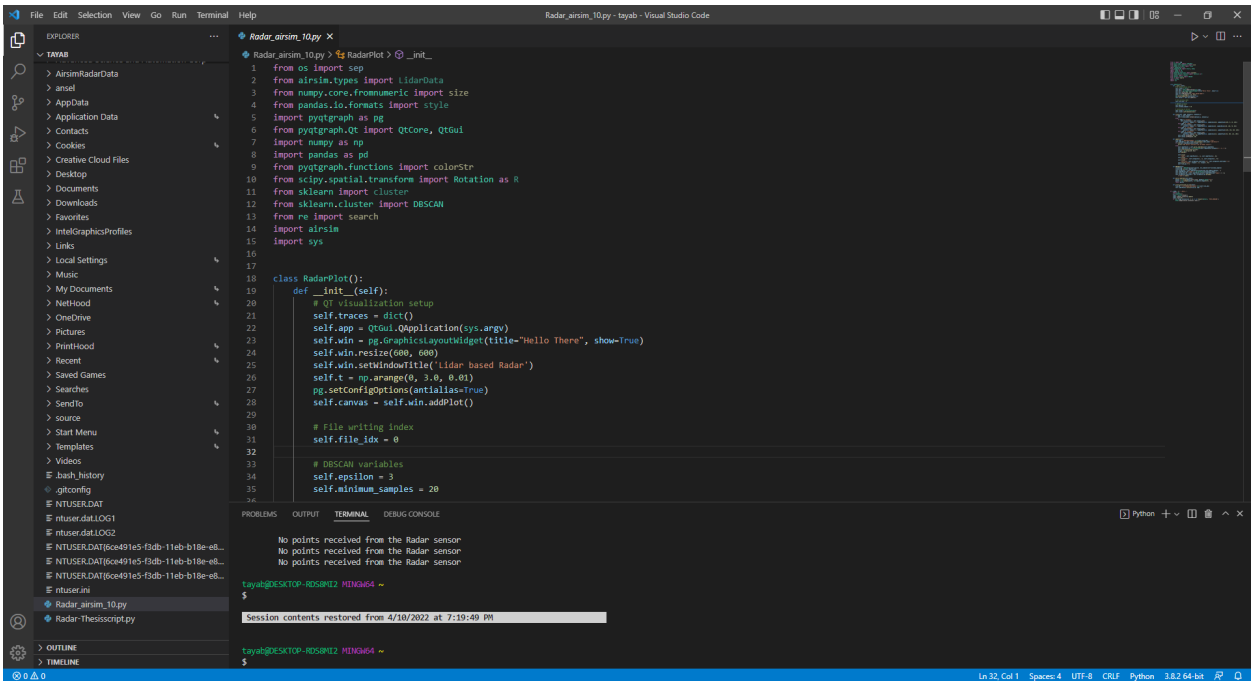

Figure 4.2. VSCODE Environment.

Figure 4.2 Shows VSCODE which can run the RADAR model, as soon as the python code is run a python window is displayed which shows the RADAR sensor outputting the range and azimuth of the objects with respect to the vehicle with the vehicle always being the center.

4.3 Vehicle

An SUV was used for the simulations as it is versatile and can be used in multiple settings and the RADAR sensor is mounted on the front grill of the vehicle where it is general mounted in many vehicle.

Figure 4.3. SUV Vehicle for simulation.

The Figure 4.3 shows an SUV with the RADAR mounted on the front grill. RADAR sensors are mounted in the front of the vehicle for ADAS applications such as Adaptive Cruise control and collision avoidance systems.

4.4 RADAR Specification

In the proposed RADAR model, In the horizontal plane, the LiDAR data spans 360 degrees. A RADAR, on the other hand, has a defined horizon that is related with the type of RADAR (e.g. short range, medium range). We used a medium-range automotive-grade RADAR with a horizontal field of view of 45 degrees for this study.

Table 4.1. RADAR Specification. This table depicts the radar specification used for the RADAR sensor model.

Parameter	**Mid-Range Requirement**
Minimum range	3m
Maximum range	80m
Azimuth field of view	>90deg
Vertical field of view	8.0deg
Update Interval	10ms

5. SIMULATION RESULTS

A number of Simulation environments were considered in order to test the proof of concept of the RADAR Model. The environment were built on Unreal Engine 4.26 with different actors and setting in each of the built test environment as shown in Figure 5.1. Sample results of the RADAR model are shown in figure 5.2. The environments used were:

1. Offroad terrain – For the Offroad terrain, Landscape environment from UE was used as that is an open source environment with its source code available and could be modified according to the needs of the project.

2. Urban terrain – For the Urban terrain a city environment was used which had prebuilt vehicle models and random constant traffic which gave a good understanding of the object.

3. Rain environment – The rain environment was developed by using the default game development model in Unreal Engine and then adding rain particles in the environment.

Figure 5.1. Sample Unreal Engine Test Environment.[3]

Table 5.1. Parameters used for Radar Model.

epsilon	3
min_sample size	20

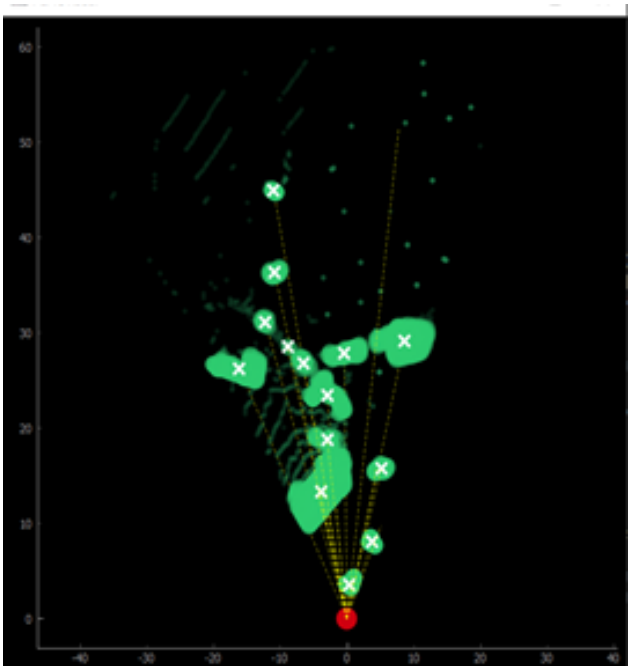

Figure 5.2. Simulated Radar Plot with Clustered Point Cloud after Data is Clipped and Normalized.[3]

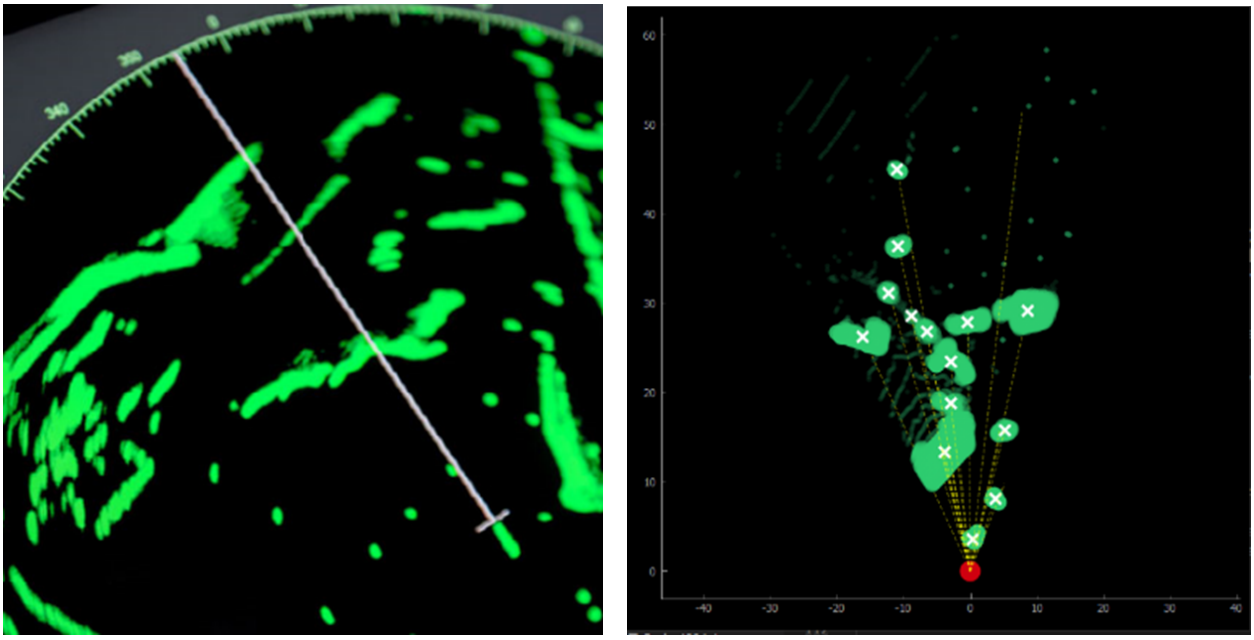

Figure 5.3. Comparison Of Simulated (Left) Actual Radar Sensor Data and (Right) Simulated Radar Plot.

Figure 5.3 compares the plot of generated RADAR data to a typical plot from a real RADAR output The essence of the actual RADAR output image exhibited in the left figure is captured by the simulated RADAR output image on the right.

5.1 Off-Road Terrain

The off-road terrain was developed from the landscape terrain which can be found in in learn on Unreal Engine and is open source.

Figure 5.4. Landscape Environment With Trees On Unreal Engine. This Figure Shows The Suv In A Landscape Environment Looking At The Scenario Of Multiple Trees And Some Small Rocks.

Figure 5.5 shows the RADAR of the scene shown in Figure 5.4. As seen in the plot the RADAR shows that it has successfully detected the trees and been able to see some of the rocks on the ground. As the min_sample size is set to 20 any object which reflects less than 20 points will not be detected by the RADAR model.

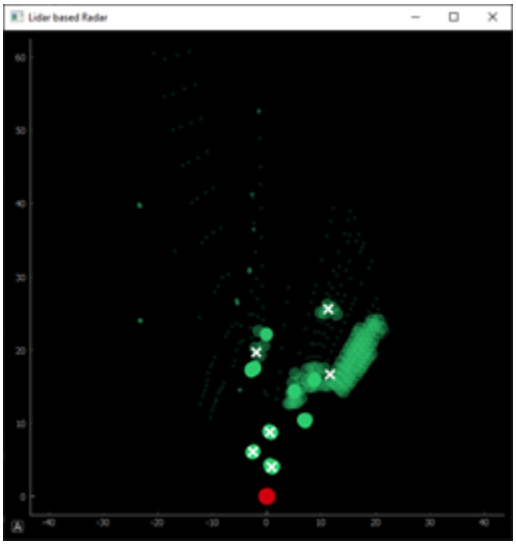

Figure 5.5. Radar Plot Of The Landscape Environment With Trees. This Figure Shows The Radar Plot Of The Landscape Environment Shown In Figure 5.4.

Figure 5.6. Landscape Environment With Open Space On Unreal Engine. This Figure Shows The Suv Pointing to the Open Field and we are Expect No Objects Detection in The Environment. However, it is Worthy to not that the Side Objects and Hills Maybe Detected as an Object.

Figure 5.7 shows the RADAR plot for Figure 5.6 and it shows that the model is not picking up any objects in the plot.

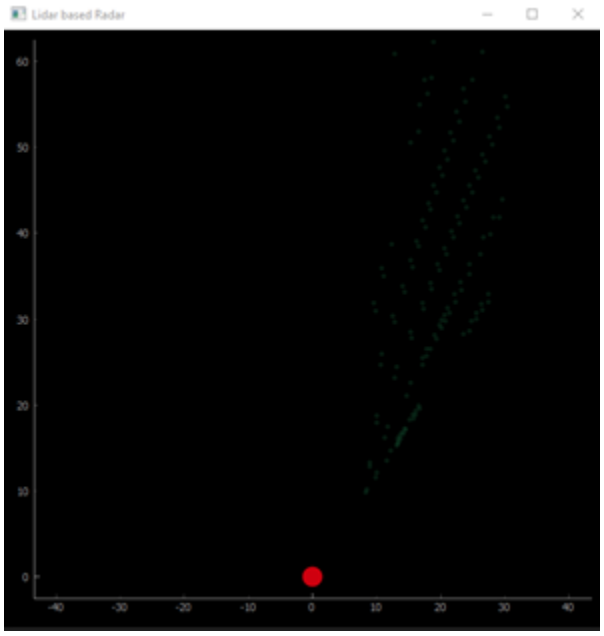

Figure 5.7. Radar Plot Of The Landscape Environment With Open Space.

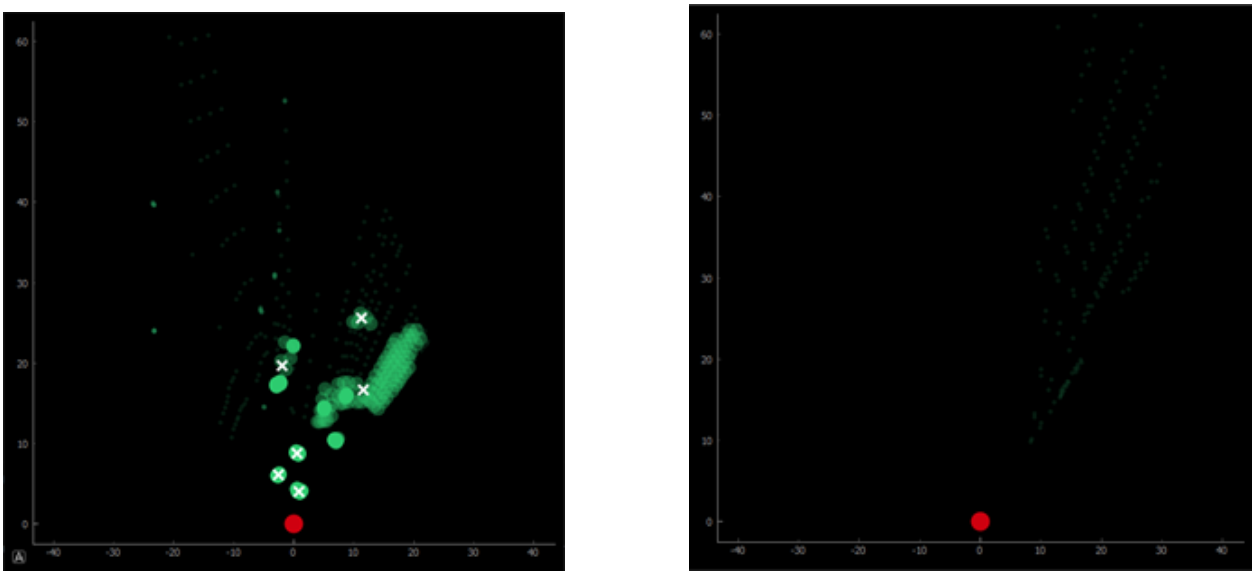

Figure 5.8. Comparison Of The Radar Plot. The Above Picture Depicts The Comparison Of The Plot Where Multiple Objects Are Detected In Figure 5.5 And No Objects Detected In Figure 5.7.

5.2 Urban Terrain

Urban terrain is a city environment with multiple vehicle models and random traffic in the simulation. This gives us the ability to test the sensor working even in a city environment and can be used to have real traffic scenario as well as test the ADAS algorithm. As shown in Figure 5.9.

Figure 5.9. City Environment On Unreal Engine With Vehicles.

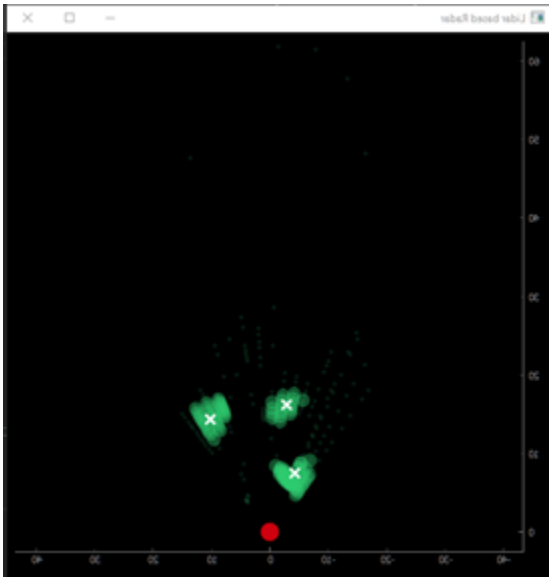

Figure 5.10. Radar Plot Of City Environment With Vehicles. This Figure Shows The Radar Plot Of The Scenario Shown In Figure 5.9. It Is Seen That The Plot Can Detect The Vehicles As Objects However It Does Not Give Us The Exact Size Of The Object Seen In The Scenario Like The Bus.

Figure 5.11. City Environment On Unreal Engine With Buildings. The Above Figure Shows The Vehicle Travelling On Open Road Without Any Vehicles To Show The Detection Of Building In The Figure 5.12.

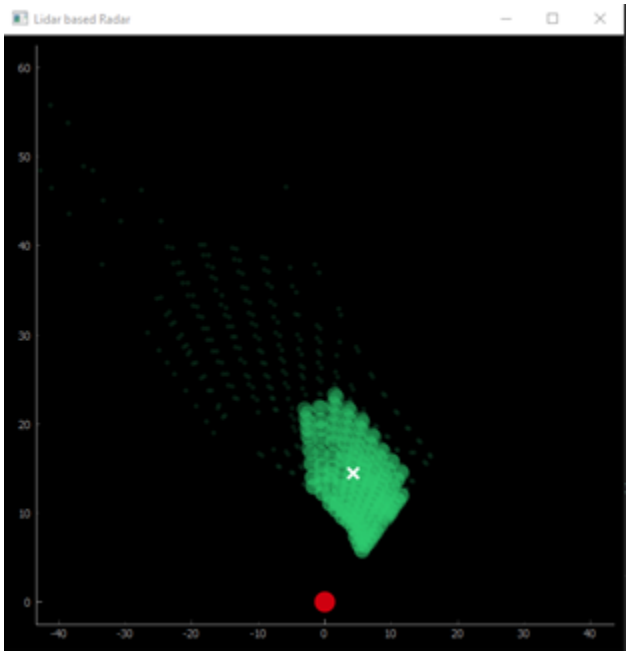

Figure 5.12. RADAR Plot of City Environment with Buildings.

5.3 Rain Environment

The Environment was developed on Unreal Engine using the default game settings and adding rain as particles in the whole environment. As seen the the sensor model currently is not affected by the rain particles in Figure 13.

Figure 5.13. Rain Environement On Unreal Engine.

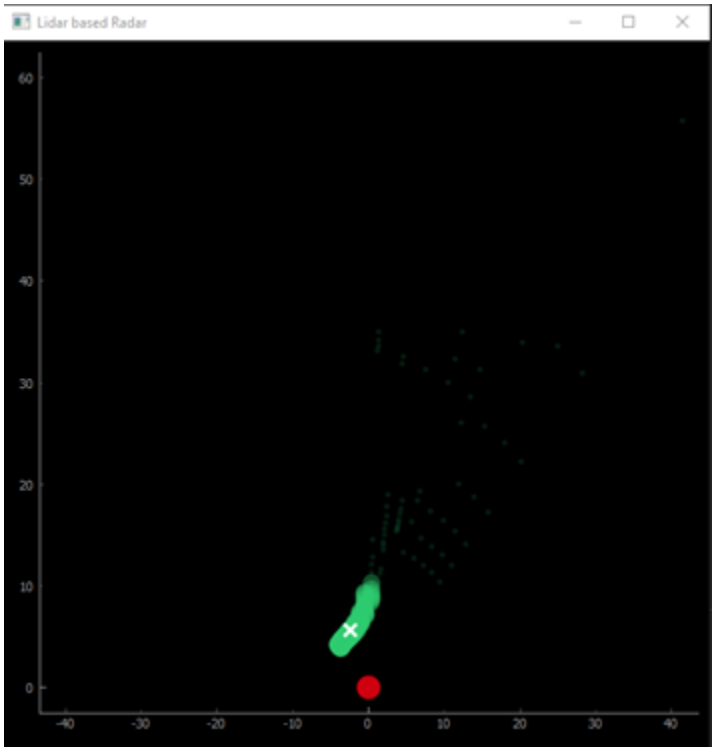

Figure 5.14. Radar Plot Of Rain Environment.

Figure 5.15. Rain Environment With Offroad Rain Setting On Unreal Engine. The Above Figure Shows The Vehicle Travelling In The Offroad Terrain In The Rain Environment.

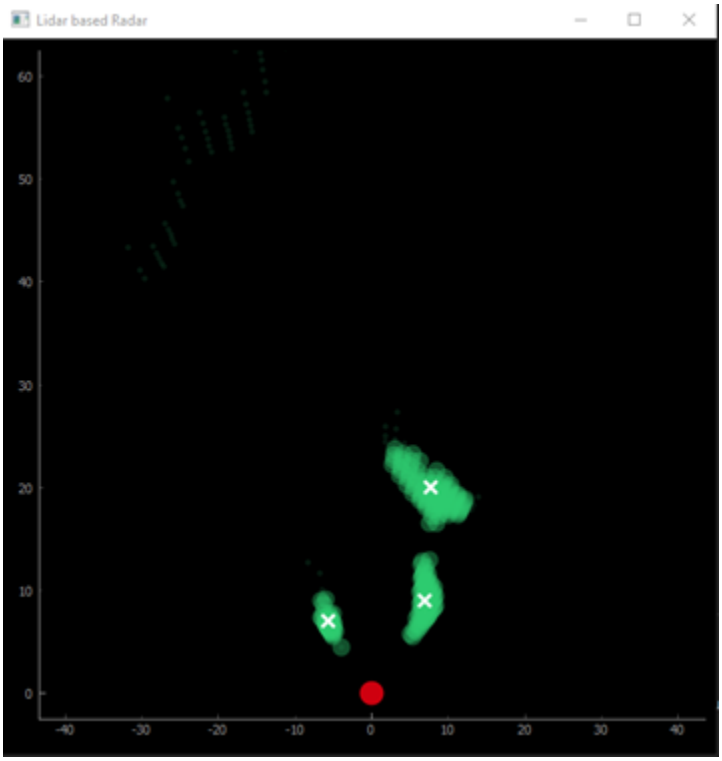

Figure 5.16. RADAR Plot Of Offroad Rain. The Figure Shoes The Simulated Radar Plot With Reference To The Figure 5.15.

5.4 Results and Discussion

To validate the results, we compared the results of the number of the objects detected by the RADAR to the number of objects detected by the pointcloud generated by the LiDAR data by normalizing the data and using the same clustering algorithm to see how the results defer as the LiDAR generates more number of points on an object.

The proof of concept shows the usability of the RADAR model on multiple environments and many environments can be created on Unreal Engine. More scenarios and iterations can be conducted, and test cases can be designed keeping the model in consideration.

As we know the parameters used for the RADAR are:

e = 2

min_samples = 20

Table 5.2. Comparison between RADAR and LIDAR.

Parameter	**OFF-ROAD ENV**	**URBAN ENV**	**RAIN ENV**
Objects detected by RADAR	5	3	3
Objects detected by LiDAR	7	5	4
Accuracy %	71.5	60	75

5.4.1 Discussion

From the above result we can infer that the overall target detection accuracy of 69% accurate this is due to few of the unsolvable failures of the RADAR sensor Model.

1. The Model parameters have to be tweaked according to the environment and the attributes of the object. The model fails in many offroad situations when there are small hills as the model detects the peaks of the small hills as objects.

2. The Model cannot detect the velocity of dynamic objects. This can be attained by tracking an object through multiple frames to compute the velocity of the object. But that was not the scope of this study as it only serves as a proof of concept.

3. The RADAR model is not able to differentiate between objects placed very close to each other such as the two trees placed very close to each other in the offroad environment in Figure 5.5.

5.4.2 Challenges

There are few challenges with the RADAR sensor model with respect to the performance of the overall model.

1. Getting the correct epsilon and min_samples value. – It is very important to get the correct epsilon and min_samples value in order to detect objects correctly. Even though multiple iterations were conducted to find the optimal value on epsilon and min_samples. We can only know more by testing the model on a greater number of scenarios in order to see the behaviour by chaging the type of objects. As seen in The Landscape scenario it is noticed that sometime the sensor model cannot differentiate between two very close placed objects. For example, the two trees are considered as one and also some of the rocks could have been detected which were not detected in the sensor model Figure 5.5.

2. False Positives. – In the experiment conducted it was found that there are some false positive objects detected by the sensor model due to the undulation in the terrain which is considered as an object by the senor model. As seen in the urban environment we there are few false positives where there are no vehicles present in the field of view but is detected by the sensor model. This is due to the epsilon value which needs to be optimum.

5.4.3 Model Limitations

1. The model is limited to have a 2D frame and can only visualize the in a 2D data frame which is not flexible to display higher resolution if necessary.

2. The model cannot measure the minimum electrical cross sections cross section of the object.

3. Since the model is a proof of concept, there are no noise models integrated into the model to do have failures due to fault detection and sensor failure.

4. The model is not affected due to temperature conditions which would affect the performance of the sensor in real life conditions.

6. CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

In this thesis, a methodology to implement an open-source RADAR sensor model modular to run with any simulator producing pointcloud data for autonomous vehicle simulations is presented which gives the object's range and azimuth with respect to the car using density-based clustering DBSCAN to cluster points from an object and displaying it in 2D using python libraries to display the RADAR. The model was tested on three different environments:

1. Offroad environment
2. Urban Environment
3. Rain weather condition.

For an open-source simulation platform using LiDAR point cloud data, the thesis proposed a less computationally intensive RADAR model. The preliminary results demonstrate that by utilizing a density-based clustering algorithm like DBSCAN, a LiDAR point cloud can be used to replicate a RADAR. A RADAR model like this can be used in an open-source simulation environment like Unreal Engine without adding a lot of processing power. The RADAR model is modular and flexible since it works on a separate python environment it can be used with any simulator producing pointcloud data. The model displays robustness however there are some complexities for the sensor model such as setting the right values for the epsilon and min_samples of DBSCAN. The Model has not been testing against detecting smaller objects such as motorcycle or atvs.

Also, RADAR is also used in aerospace, the model also needs to be tested in those scenarios as it might be useful to see the long range detection specially on drones since currently LiDAR sensors are very heavy for small drone applications which depend on ultrasonic and RADAR sensors for object detection and avoidance systems.

6.2 Future Scope

Robust simulation testing is required for ADAS and autonomous vehicle algorithms. Multiple test cases can be developed to conduct multiple iterations of the test. The system was simulated on a personal computer, therefore the number of resources such as RAM, CPU, and memory allocated were limited. If the software elements were connected to a computer that could run the Unreal engine environment and sensor models separately can help in creating more realistic simulations which need a lot more GPU power.

The model may be evaluated against changes in environmental variables like dust, snow and other various weather conditions in the future. A noise model could also be incorporated to simulate RADAR sensor failure scenarios. The effects of temperature on the RADAR sensor can also be studied. RADAR sensors are affected by extreme temperatures which is a complex challenge.

1. The RADAR sensor can also be tested for Aerospace applications such as helicopters and airplanes and small sized drones which have challenges in using many sensors and maintaining the weight of the aircraft.

2. The RADAR should be sensitive to all the attributes of the RADAR sensor the RADAR should also see the actor's electrical size (RADAR cross section)

3. Since the RADAR model is a proof of concept to further validate the model, it should be further validated by comparing the results with actual RADAR data collected by hardware-in-loop testing and real world environment testing.

REFERENCES

[1] M. M. Rohde, J. Crawford, M. Toschlog, K. D. Iagnemma, G. Kewlani, C. L. Cummins, R. A. Jones, and D. A. Horner, "An interactive physics-based unmanned ground vehicle simulator leveraging open source gaming technology: Progress in the development and application of the virtual autonomous navigation environment (vane) desktop," in *Unmanned Systems Technology XI*, International Society for Optics and Photonics, vol. 7332, 2009, p. 73321C.

[2] (2019). "Github - airsim." Date Accessed: 2020, [Online]. Available: https://github.com/Microsoft/AirSim.

[3] T. A. Kesury, C. O. Cardoza, and S. Anwar, "Radar modeling for autonomous vehicle simulation environment with open-source utilities," in *ASME International Mechanical Engineering Congress and Exposition*, American Society of Mechanical Engineers, vol. 85628, 2021, V07BT07A029.

[4] (2016). "The chicago manual of style." [Online], [Online]. Available: https://cleantechnica.com/2016/07/29/tesla-google-disagree-lidar-right.

[5] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, p. 648, 2019.

[6] U. Chipengo, A. Sligar, and S. Carpenter, "High fidelity physics simulation of 128 channel mimo sensor for 77ghz automotive radar," *IEEE Access*, vol. 8, pp. 160 643–160 652, 2020.

[7] R. B. Banerjee, "Development of a simulation-based platform for autonomous vehicle algorithm validation," Ph.D. dissertation, Massachusetts Institute of Technology, 2019.

[8] M. F. Holder, C. Linnhoff, P. Rosenberger, C. Popp, and H. Winner, "Modeling and simulation of radar sensor artifacts for virtual testing of autonomous driving," in *9. Tagung Automatisiertes Fahren*, 2019.

[9] M. Holder, P. Rosenberger, H. Winner, T. D'hondt, V. P. Makkapati, M. Maier, H. Schreiber, Z. Magosi, Z. Slavik, O. Bringmann, *et al.*, "Measurements revealing challenges in radar sensor modeling for virtual validation of autonomous driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 2616–2622.

[10] M. Jasiński, "A generic validation scheme for real-time capable automotive radar sensor models integrated into an autonomous driving simulator," in *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*, IEEE, 2019, pp. 612–617.

[11] L. Wang, B. Goldluecke, and C. Anklam, "L2r gan: Lidar-to-radar translation," in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[12] T. A. Wheeler, M. Holder, H. Winner, and M. J. Kochenderfer, "Deep stochastic radar models," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2017, pp. 47–53.

[13] L. McInnes, J. Healy, and S. Astels, "Hdbscan: Hierarchical density based clustering," *The Journal of Open Source Software*, vol. 2, no. 11, Mar. 2017. DOI: 10.21105/joss.00205. [Online]. Available: https://doi.org/10.21105%2Fjoss.00205.

[14] (2019). "Clustering in machine learning." Date Accessed: 2020, [Online]. Available: https://developers.google.com/machine-learning/clustering/clustering-algorithms.

[15] (2021). "Unreal engine." Date Accessed: 2020, [Online]. Available: www.unrealengine.com/en-US.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, vol. 96, 1996, pp. 226–231.

[17] A. Madan, *Acoustic simultaneous localization and mapping (slam)*, 2021.

APPENDIX A. PYTHON CODE

```
from os import sep
from airsim.types import LiDARData
from numpy.core.fromnumeric import size
from pandas.io.formats import style
import pyqtgraph as pg
from pyqtgraph.Qt import QtCore, QtGui
import numpy as np
import pandas as pd
from pyqtgraph.functions import colorStr
from scipy.spatial.transform import Rotation as R
from sklearn import cluster
from sklearn.cluster import DBSCAN
from re import search
import airsim
import sys

class RADARPlot():
    def __init__(self):
        # QT visualization setup
        self.traces = dict()
        self.app = QtGui.QApplication(sys.argv)
        self.win = pg.GraphicsLayoutWidget(title="Hello There", show=True)
        self.win.resize(600, 600)
        self.win.setWindowTitle('LiDAR based RADAR')
        self.t = np.arange(0, 3.0, 0.01)
        pg.setConfigOptions(antialias=True)
        self.canvas = self.win.addPlot()

        # File writing index
        self.file_idx = 0

        # DBSCAN variables
        self.epsilon = 3
        self.minimum_samples = 20

        # Connecting to Unreal Engine
        self.client = sim.CarClient()
        self.client.confirmConnection()

    def trace(self, name, dataset_x, dataset_y):
        if name in self.traces:
            self.traces[name].setData(dataset_x, dataset_y)
        else:
```

```
        if name == "origin":
            self.traces[name] = self.canvas.plot(
                pen=None, symbol='o', symbolPen=None, symbolSize=20,
                symbolBrush=(207, 0, 15, 255))
        elif name == "RADAR":
            self.traces[name] = self.canvas.plot(
                pen=None, symbol='o', symbolPen=None, symbolSize=5,
                symbolBrush=(30, 130, 76, 50))
        elif name == "centroid":
            self.traces[name] = self.canvas.plot(
                pen=None, symbol='x', symbolPen=None, symbolSize=15,
                symbolBrush=(255, 255, 255, 255))
        elif name == "corepoint":
            self.traces[name] = self.canvas.plot(
                pen=None, symbol='o', symbolPen=None, symbolSize=15,
                symbolBrush=(46, 204, 113, 100))
        self.canvas.setXRange(-40, 40)
        self.canvas.setYRange(0, 60)

def update(self):
    file_name = "RADARData\RADAR" + str(self.file_idx) + ".txt"
    self.RADARData = self.client.getLiDARData(LiDAR_name="LiDARSensor")
    if (len(self.RADARData.point_cloud) < 3):
        print("\tNo points received from the RADAR sensor")
    else:
        self.RADARPoints = self.parse_RADARData(self.RADARData)
        self.RADARPoints_df = pd.DataFrame(self.RADARPoints,
        columns=['x','y','z'])
        self.write_pointcloud_to_file()
        origin = np.array([[0, 0]])
        self.DBSCAN()

        self.trace(
            "RADAR", self.RADARPoints[:, 1], self.RADARPoints[:, 0])
        self.trace(
            "corepoint", self.corepoints[:, 1], self.corepoints[:, 0])
        self.trace(
            "centroid", self.corepoints_centroids['y'],
            self.corepoints_centroids['x'])
        self.trace("origin", origin[:, 1], origin[:, 0])
        self.file_idx += 1

def DBSCAN(self):
    clustering = DBSCAN(eps=self.epsilon,
```

```
            min_samples=self.minimum_samples)
            clustering.fit(self.RADARPoints)
            self.corepoints = self.RADARPoints[clustering.core_sample_indices_,
            :]
            self.RADARPoints_df['label'] = pd.DataFrame(clustering.labels_)
            self.corepoints_df = self.RADARPoints_df[self.RADARPoints_df['label'
            ] != -1]
            self.corepoints_centroids = self.corepoints_df.groupby(
                ['label']).mean()

        def parse_RADARData(self,data):
            points = np.array(data.point_cloud, dtype=np.dtype('f4'))
            points = np.reshape(points, (int(points.shape[0]/3), 3))
            return points

        def write_pointcloud_to_file(self):
            file_id = 'RADARData\RADAR'+str(self.file_idx)
            self.RADARPoints.tofile(file_id, sep=',')

    if __name__ == '__main__':
        import sys
        p = RADARPlot()
        timer = QtCore.QTimer()
        timer.timeout.connect(p.update)
        timer.start(20)
        if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):
            QtGui.QApplication.instance().exec_()
```

PUBLICATION

Kesury, T., Cardoza, C., Anwar, S., (2021). "Radar Modeling For Autonomous Vehicle Simulation Environment With Open-Source Utilities." IMECE2021-72055.