Object Detection & Tracking in 4D Radar for Enhanced Autonomous Vehicle Perception

Nick Timaskovs

K00260158

A Final Year Project submitted in partial fulfilment of the requirements of the Technological University of the Shannon for the degree of Bachelor of Science (Honours) in Software Development.

Supervised By:

William Ward

TUS
Midwest
Department of
Information Technology

May, 2023

Abstract

This thesis explores the multifaceted realm of object detection, tracking, and classification within the dynamic landscape of 4D radar data in autonomous vehicles. The advent of 4D radar technology has opened the way for a new era of high-resolution, real-time data acquisition, enabling an unprecedented understanding of the complex dynamics of objects in motion within a given space and time. This research delves into the development and refinement of algorithms and techniques designed to enhance the accuracy and efficiency of object detection in 4D radar data. By harnessing the power of deep learning models, recurrent neural networks, and advanced signal processing techniques, this study aims to address the challenges posed by noisy and cluttered radar data, while also improving the speed and precision of object detection.

In addition to object detection, this thesis investigates the critical aspect of object tracking in 4D radar data. Tracking objects through space and time is a fundamental task for various applications, including autonomous vehicles, surveillance, and air traffic control. The research proposes novel approaches that leverage both historical and real-time radar data to robustly track objects, accounting for unpredictable movements, occlusions, and changing environmental conditions. Furthermore, this study expands the scope by incorporating classification into the framework, enabling the automatic identification of objects based on their radar signatures. By integrating these elements into a unified system, this thesis contributes to the advancement of 4D radar technology, opening new possibilities for applications in various domains where precise object detection, tracking, and classification are paramount in ensuring safety in autonomous vehicles

Acknowledgements

I want to express my deepest appreciation and gratitude to all those who have supported and guided me throughout the journey of completing my Final Year Project (FYP). This accomplishment would not have been possible without their encouragement, wisdom, and patience.

First and foremost, I would like to extend my sincerest thanks to my FYP supervisor, Mr Willaim Ward, for his unwavering support and guidance throughout this project. His invaluable insights, expertise, and constructive feedback have been instrumental in shaping my work and helping me achieve my goals. I am truly grateful for the time and effort he has invested in mentoring me and the confidence he has shown in my abilities.

Secondly, I would like to express my gratitude to my LSU tutor Mark Crowe

Ethical Declaration

I declare that this project and document is wholly my work except where I have made explicit reference to the work of others. I have read the Department of Information Technology Final Year Project guidelines and relevant institutional regulations, and hereby declare that this document is in line with these requirements.

I have discussed, agreed, and complied with whatever confidentiality or anonymity terms of reference were deemed appropriate by those participating in the research and dealt appropriately with any other ethical matters arising, in line with the TUS Research Ethics Guidelines for Undergraduate and Taught Postgraduate Programmes policy document.

Nick Timaskovs   May 1, 2023

Table of Contents

Glossary

ADAS: Advanced Driver Assistance Systems

AI: Artificial Intelligence

AoA: Angle Of-Arrival

AV: Autonomous Vehicle

CNN: Convolutional Neural Networks

CFAR: Constant False Alarm Rate

CW: Continuous Wave

DBSCAN: Density-Based Spatial Clustering of Applications with Noise

DDM: Doppler-Division Multiplexing

FDM: Frequency-Division Multiplexing

FFT: Fast Fourier Transform

FoV: Field of View

FMCW: Frequency-Modulated Continuous Wave

FPS: Frames Per Second

GPS: Global Positioning System

GPU: Graphics Processing Unit

HOG: Histogram of Oriented Gradients

IDE: Integrated Development Environment

LiDAR: Light Detection and Ranging

LSTM: Long Short-Term Memory

MEMS: Micro-Electromechanical System

MIMO: Multiple Input Multiple Output

ML: Machine Learning

mAP: Mean Average Precision

MVP: Minimal Viable Product

NN: Neural Network

OpenCV: Open-Source Computer Vision Library

PV-RCNN: Point-Voxel Feature Set-based 3D Object Detection from Point Clouds

PyTorch: An open-source machine learning library

R-CNN: Region-Based Convolutional Neural Network

RAD: Range-Doppler

RADAR: Radio Detection and Ranging

RAED: Range-Azimuth-Elevation-Doppler

RCS: Radar Cross Section

RD: Range-Doppler

RoI: Region of Interest

RNN: Recurrent Neural Networks

Rx: Receiver antenna

SAE: Society of Automotive Engineers

SLAM: Simultaneous Localization and Mapping

SNR: Signal-to-Noise Ratio

SORT: Simple and Online Real-time Tracking

TDM: Time-Division Multiplexing

Tx: Transmitter antenna

VOD: View-Of-Delft (Dataset)

Table of Figures

Chapter 1 Introduction

This thesis investigates object detection, tracking, and classification using 4D radar in autonomous vehicles, focusing on improving accuracy and efficiency with advanced algorithms, deep learning, and signal processing. It addresses challenges in handling noisy radar data to enhance detection speed and precision.

1.1 Objective

This project explores two main solutions with several objectives in mind, primarily to tackle the challenges of working with 4D RADAR data and to effectively implement object detection and tracking. The first implementation was building majority of the source code from the ground up and then attempting to utilize a detection framework. The second implementation involved contributing to an existing code base, making improvements, and conducting experiments. The overarching goal was to progress in the research and advancements of RADAR technology in the automotive industry, working towards improved autonomous vehicle perception.

This solution's objectives are:

- To overcome the difficulty of working with 4D RADAR data.
- Successfully implement object detection and tracking.
- Experiment with variation in configuration across different epoch ranges for improved results.

The academic objectives:

- To learn more about deep learning.
- To learn and understand 4D RADAR point cloud data structure.
- To learn and understand how object detection, tracking, and classification are done in 4D RADAR.
- Gain experience with academic research.

1.2 Solution Developed

The developed source code is available on Git Hub: https://github.com/nicktmv/four-d-radar-thesis with all the necessary steps contained in the README.md file.

1.3 Report Structure

This report comprises five main chapters: Literature Review, Analysis and Design, Implementation, Testing and Results and Conclusions.

- Literature Review: This chapter surveys existing research and developments in the field to establish the study's context and foundation.

- Analysis and Design: It outlines the theoretical framework and design methodology used for the study's objectives.

- Implementation: This section describes the practical application of the designed framework and the development of the study's solution.

- Testing and Results: It presents the evaluation methods, testing procedures, and the outcomes of the implemented solution.

- Conclusions: This final chapter summarizes the study's findings, discusses their implications, and suggests areas for future research.

Chapter 2 Literature Review

2.1 Introduction

This chapter presents a comprehensive review of the existing literature on autonomous vehicle perception, focusing on the integration and functionalities of various sensor systems such as LiDAR, RADAR, and computer vision, all underpinned by deep learning technologies. It delves into the critical role of sensor fusion in enhancing the accuracy and reliability of autonomous systems, particularly under diverse driving conditions. The literature review explores significant advancements in radar technology, particularly 4D radar, which provides enhanced data for object detection and environmental understanding, crucial for autonomous navigation.

The discussion extends to the comparative strengths and weaknesses of radar and LiDAR technologies, noting RADAR's robustness in adverse weather conditions and LiDAR's superior precision in object detection. Deep learning models, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are discussed extensively as they form the backbone of modern object detection and classification systems in autonomous vehicles. These models significantly enhance the vehicle's ability to interpret complex scenes, contributing to safer driving decisions.

The chapter aims to encapsulate the core findings from the literature and set a clear pathway for the subsequent chapters of the thesis, emphasizing the ongoing challenges in sensor data fusion and the prospects for future research to enhance the adaptability and safety of autonomous vehicles.

2.2 Autonomous Vehicles

Autonomous vehicles (AVs) are self-driving vehicles that are equipped with a multitude of sensors illustrated by **Figure 1**, including LiDAR, radar, cameras, ultrasonic sensors, and GPS. These sensors provide information about the vehicle's surroundings, capturing data about other vehicles, pedestrians, road infrastructure, and weather conditions. AVs and ADAS also use advanced AI and ML algorithms to process sensor data to make real-time decisions and interact with vehicle Control Systems such as steering, acceleration, and braking. (Synopsys.com, 2023)
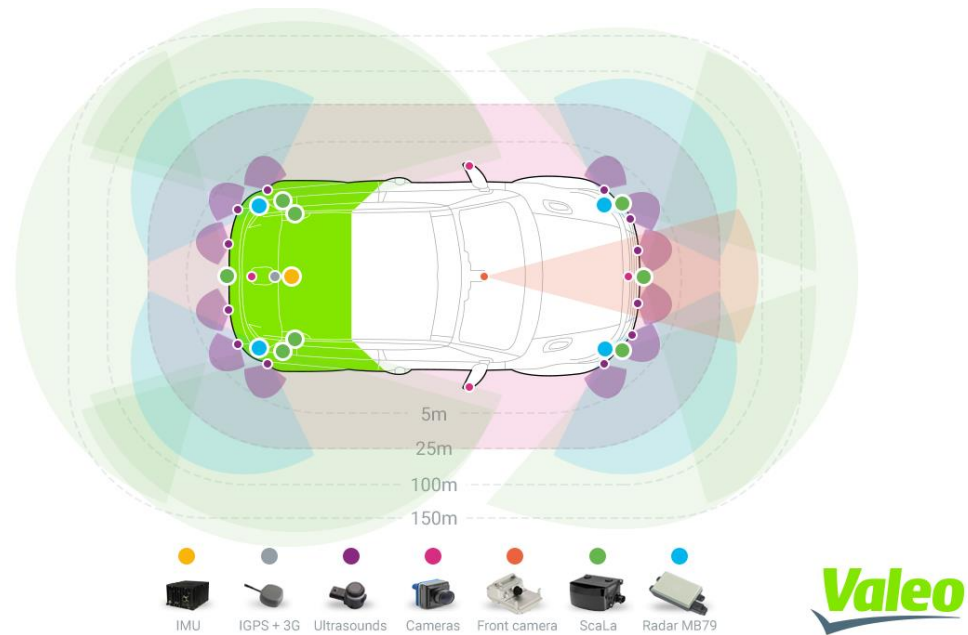
Figure 1: Sensor setup of the Valeo Drive4U prototype. (Valeo 2018)

2.2.1 Autonomous Vehicle Perception.

Perception is a fundamental component of AVs and advanced driver assistance systems (ADAS), referring to how a vehicle senses or "sees", enabling them to understand their environment, including identifying and tracking objects, pedestrians, other vehicles, and road infrastructure such as lane markings, traffic signs, and traffic lights. Perception is done with the use of a multitude of sensors such as RADAR, LiDAR, Cameras, and Ultrasonic.
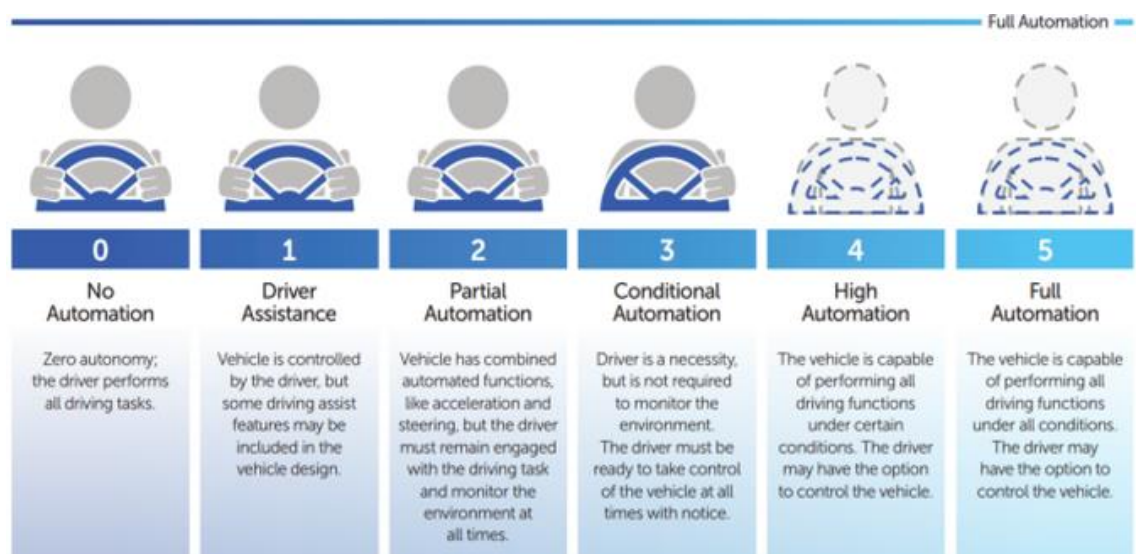
Figure 2: (2.0 A Vision for Safety AUTOMATED DRIVING SYSTEMS, n.d.)

Illustrated by **Figure 2**, AD and ADAS are categorized into 6 levels, Level 0 (full human control) to Level 5 (complete vehicle autonomy) these levels are defined by the Society of Automotive Engineers (SAE) in their J3016 standard, which provide a framework to understand the progression of autonomy in vehicles.

As ADAS research, testing, and implementation in vehicles continue to grow worldwide, there is an increasing focus on establishing standardized rules and regulations to guarantee their secure incorporation into society. Most commercial vehicles are categorized as Level 1 to Level 2 autonomy due to limitations in sensors, cost factors, and the need for ongoing driver attention and control. These vehicles generally come equipped with ADAS features such as emergency braking, blind spot detection, Adaptive Cruise Control, Automatic Parking, and Lane Assist. (2.0 A Vision for Safety AUTOMATED DRIVING SYSTEMS, n.d.)

At CES 2023, Mercedes-Benz made a significant announcement, stating that it had achieved Level 3 (L3) autonomous driving certification in the United States, specifically from the state of Nevada. It's worth noting that L3 certification is granted at the state level in the US, so Mercedes' system is only considered L3 in Nevada for now. This move by Mercedes is expected to encourage other major automotive manufacturers such as Hyundai-Kia, Stellantis, BMW, GM, and Honda to pursue Level 3 autonomous driving technology, as they have also been reporting progress and plans for L3 rollout. (AUTOCRYPT, 2023).

SLAM algorithms are employed in autonomous vehicles, they enable the vehicle to create a map of its surroundings while determining its position within that map simultaneously. SLAM algorithms empower the vehicle to chart unexplored environments, and engineers utilize this map data for tasks like planning routes and avoiding obstacles. (Mathworks.com, 2023)

AVs use sensor fusion techniques to combine data from multiple sensors, improving the accuracy and robustness of perception systems. **Kalman filters** and **Bayesian** approaches are commonly employed for sensor fusion. (Anwesh Marwade, 2020)

2.2.2 Kalman Filters

The Kalman Filter, developed by Rudolf E. Kálmán in 1960, this algorithm efficiently filters linear discrete data through a recursive solution. It is pivotal for addressing challenges in radar tracking by accurately estimating an object's state from uncertain and imprecise radar measurements, and predicting its future states from past data, thus improving consistent tracking. Understanding prediction algorithms is essential in radar technology. A radar system, updating every few seconds, not only determines an object's current position and velocity but also predicts its future position using Newtonian mechanics:

$$x = x_0 + v_0 \Delta t + \frac{1}{2} a \Delta t^2$$

Where:
x is the target position
x_0 is the initial target position
v_0 is the initial target velocity
a is the target acceleration
Δt is the time interval (5 seconds in our example)

This extends into a three-dimensional Dynamic Model, vital for predicting the object's future state from its current state.

Real-world tracking faces challenges like Measurement Noise from radar specifics and Process Noise from environmental factors like wind or manoeuvres. Using the Kalman Filter, which addresses both types of noise, enhances tracking accuracy and ensures reliable predictions of an object's trajectory and future location. (Alex Becker (www.kalmanfilter.net, 2017)

2.2.3 Bayesian Methods

Bayesian methods are integral to enhancing radar object detection in advanced driver-assistance systems (ADAS) and automated driving (AD) applications through the implementation of Bayesian Gaussian Mixture Models (GMMs). These methods facilitate the development of sophisticated radar sensor models that effectively capture the complex and dynamic nature of vehicle environments.

The Bayesian approach specifically aids in automatically determining the complexity of the statistical models used, which is critical for accurately modelling the radar cross-section (RCS) of different objects. This RCS modelling is essential as it significantly influences the radar's ability to detect and track objects based on how they reflect radar signals. Moreover, Bayesian methods support the incorporation of occlusion effects and RCS-based detectability into the models, ensuring that the simulations account for objects obstructing each other from the radar's view.

Overall, the use of Bayesian methods allows for a flexible and modular framework that adapts to new data and different scenarios, ensuring that the radar sensor models are both robust and scalable (Walenta, Genser, and Selim Solmaz, 2024).

Figure 3 illustrates how a GMM can be used for example, we have a bunch of samples of cars. Different types of cars have different shape types, sizes, and colours, but they all belong to the car category. At this time, the single Gaussian model may not describe the distribution very well since the sample data distribution is not a single ellipse. However, a mixed Gaussian distribution can better describe the problem.
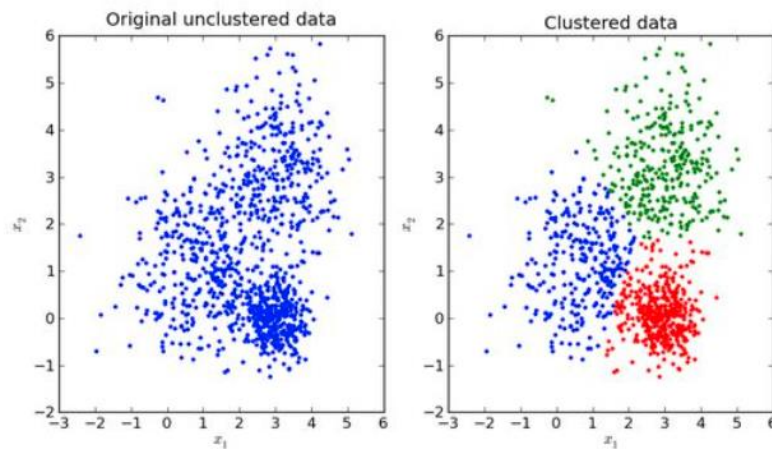
Figure 3: GMM (Robotics Knowledgebase, 2019)

2.3 Overview of Radar

Radar technology has become the basis for modern sensing and surveillance systems for several decades. Its applications span from military, aviation and maritime navigation to weather monitoring and autonomous vehicles. Radar technology has its roots in the early 20th century, with significant advancements occurring during World War II. Early radar systems used microwave signals to detect and locate objects. Notable developments include the British Chain Home system and the American SCR-270 radar. These systems provided critical advantages in terms of early warning and target tracking. (Bloom, 2020)
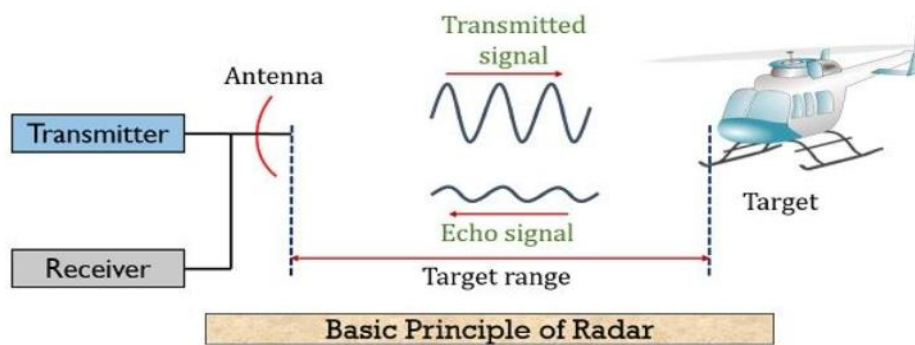
Figure 4: Basic Radar Operation (Roshni Y, 2019)

Illustrated by **Figure 4**, Radar operates on the principles of emitting electromagnetic waves, using a transmitter antenna (Tx) and receiving their echoes via a receiver antenna (Rx) after reflecting off objects and analyzing the time delay and **Doppler shift** of these echoes to determine an object's range, speed, and direction. Key components include transmitters, antennas, receivers, and signal processing units.

Doppler Shift Definition: "*the apparent difference between the frequency at which sound or light waves leave a source and that at which they reach an observer, caused by the relative motion of the observer and the wave source*". (*Doppler* effect | Definition, Example, & Facts| Britannica, 2023)

2.3.1 FMCW Radar Signal Processing.

There are many different types of RADAR, all used in various use cases as discussed previously however mentioned in section [2.3], FMCW is the most used in automotive. and is relevant to this use case.

FMCW operates using linear frequency-modulated continuous-wave signals to measure range, angle, and velocity. Based on regulations they can use two frequency bands: 24 GHz and 77 GHz, with a preference for 77 GHz due to its wider bandwidth, higher Doppler resolution, and smaller antennas great for long-range use cases. (Udemy, 2023)

FMCW signals have key parameters as shown in **Figure 5:** start frequency (fc), sweep bandwidth (B), chirp duration (Tc), and slope (S). A frame consists of chirps with a frame time of (Tf). A frequency mixer combines received and transmitted signals to produce two signals: sum frequency $fT(t)+fR(t)$ and difference frequency $fT(t)-fR(t)$. with a low-pass filter used to obtain the intermediate frequency (IF) signal. Complex exponential IF signals are achieved in practice using a quadrature mixer. (Zhou et al., 2022)

Range and Doppler velocity can be estimated from the IF signal, leading to a 2D complex data matrix called the Range-Doppler (RD) map. The angle information is obtained using multiple receivers or transmit channels. (Zhou et al., 2022)
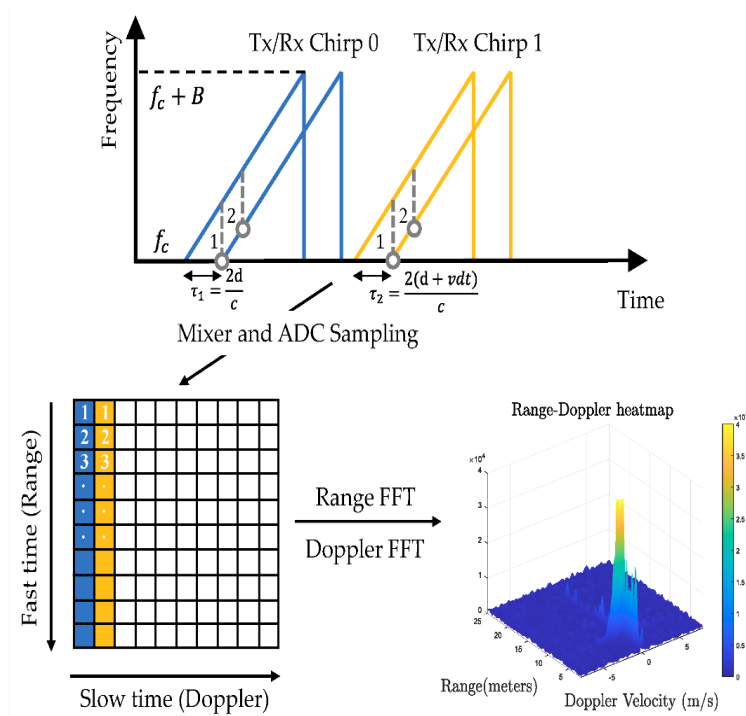
Figure 5: Radar Tx/Rx signals and the range-Doppler map (Zhou and Yue, 2022)

Angle estimation in radar can be achieved using SIMO radar, which involves a single transmitter antenna and multiple receive antennas. By measuring the phase change between adjacent receive antennas, the direction of an object can be calculated using the formula $\Delta\phi = 2\pi d\sin\theta/\lambda$, where θ represents the object's angle, and d is the antenna spacing. (Zhou et al., 2022)

To achieve maximum angle precision, the antenna spacing can be set to $\lambda/2$, and a third Fast Fourier Transform (FFT) is used for processing.

The angular resolution of a SIMO radar depends on the number of receive antennas, but this number is limited by the cost of signal processing.

MIMO radar employs multiple transmit and receive channels, creating a virtual array with many channels. Various techniques like time-division multiplexing (TDM), frequency-division multiplexing (FDM), and Doppler-division multiplexing (DDM) are used to ensure signal separation.

TDM is straightforward, where different transmit antennas take turns sending signals. DDM sends all signals simultaneously but reduces Doppler velocity resolution.

Once signals are separated, a 3D tensor, called the RAD tensor, is created by stacking Range-Doppler (RD) maps. Angular resolution can be enhanced using super-resolution techniques like Capon, MUSIC, and ESPRIT.

The radar detection process involves coherent integration to improve signal-to-noise ratio (SNR), followed by a constant false alarm rate (CFAR) detector for peak detection. Angle estimation is then applied, resulting in a point cloud with range, Doppler, and angle measurements.

In conventional radars, only azimuth angles are resolved, while 4D radars output both azimuth and elevation angles.

Low CFAR thresholds are used in safety-critical applications for high recall. Spatial-temporal filtering techniques, such as DBSCAN and Kalman filtering, are employed to reduce errors caused by clutter and interference.

2.4   Overview of LiDAR

LiDAR, an abbreviation for "Light Detection and Ranging," constitutes a remote sensing method utilizing laser light for precise distance measurements. This technique facilitates the generation of intricate 3D maps of surroundings. LiDAR functions by emitting quick laser pulses and precisely gauging the duration for the reflected light to travel back.

LiDAR functions through a typical system comprising three fundamental elements:

Laser Source: Utilizing lasers, LiDAR emits rapid bursts of light, usually within the near-infrared spectrum. These laser pulses travel through the atmosphere until encountering objects such as terrain, trees, or structures.

Scanner: To ensure comprehensive data collection, a scanner, often a rotating mirror or micro-electromechanical system (MEMS), directs the laser pulses in various directions.

Detector: An incredibly sensitive instrument, the detector measures the time taken for the laser pulses to reach objects and return to the LiDAR system. This principle of "time-of-flight," in conjunction with the constant speed of light, facilitates precise distance calculations.

When a laser pulse strikes an object like a tree or a structure, the light reflects to the LiDAR system. The scanner, typically a rotating mirror or MEMS, captures and directs the returning light to the detector. The detector, being highly sensitive, measures the time it takes for light to travel back and forth between the LiDAR system and the object. This principle of time-of-flight, combined with the constant speed of light, enables LiDAR to accurately compute the distance to an object.

2.5   Overview of 4D mm Radar

4D-imaging radar represents a cutting-edge iteration of mm-wave radar technology that surpasses conventional radar's capabilities. It deploys echolocation and utilizes time-of-flight measurement principles to create a representation of objects within a 3D setting. The four dimensions include **Range, Azimuth, Elevation, and Doppler velocity.** It also provides some other low-level capabilities such as radar-cross-section (RCS) or signal-to-noise ratio (SNR). (Everythingrf.com, 2021)

To create a detailed representation of the surrounding environment for a vehicle, a 4D imaging radar employs a Multiple Input Multiple Output (MIMO) antenna array. This array can consist of numerous antennas that transmit signals towards objects in the vicinity of the device and subsequently collect the reflected signals. The information gathered by these antennas is then utilized to construct a point cloud shown in **Figure 6**, depicting the region encompassing the vehicle in high detail. This point cloud captures not only the location and movement of objects but also their shape and texture, allowing for an unprecedented level of environmental awareness. By analysing the Doppler shifts in the returned signals, the system can determine the velocity of moving objects, adding a dynamic aspect to the static scene captured by traditional radar systems. The use of a MIMO antenna array significantly enhances the radar's resolution and accuracy, enabling the detection of smaller objects and finer details in complex urban and highway environments. Consequently, 4D imaging radar plays a crucial role in enhancing the safety and efficiency of autonomous driving systems, offering a comprehensive understanding of the vehicle's immediate surroundings and potential hazards.
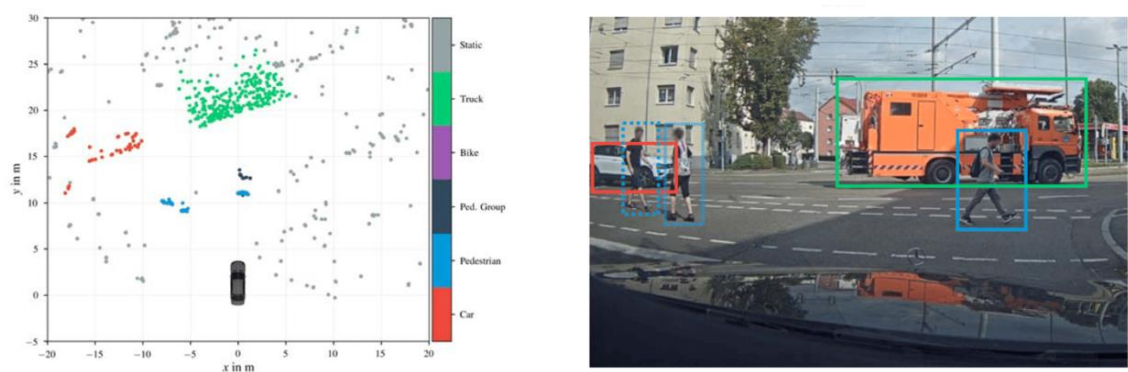
Figure 6: 4D RADAR detections projected onto an image. (Zhou et al., 2020)

2.6   RADAR Vs LiDAR

This section explores the differences between RADAR and LiDAR, while also comparing their advantages and disadvantages.

2.6.1   4D Radar Advantages.

Unlike RGB cameras, which utilize the visible light spectrum (384-769 terahertz), and LiDAR systems, which operate in the infrared spectrum (361-331 terahertz), Radars operate at significantly longer radio wavelengths (77-81 gigahertz). This characteristic allows Radars to provide reliable measurements (Cohen, 2023) even in adverse weather conditions such as rain, fog, or snow, ensuring reliable performance in various environments. (Yang et al., n.d.)

RADAR operates independently of ambient lighting conditions, making it ideal for 24/7 surveillance and autonomous driving at night (Zhou et al., 2020). By breaking down each azimuth into an array of intensity values distributed radially, RADAR introduces an additional dimension that LiDAR lacks. This unique feature enables RADAR to construct a top-down, image resembling a photograph, a task that a LiDAR unit cannot accomplish without incorporating multiple channels. (Navtech Radar, 2023)

4D RADAR offers notable advantages in terms of long-range detection capabilities, with a range extending beyond 200 meters, as well as robustness. Millimetre-wave RADAR can penetrate certain non-metallic obstacles, such as plastic and fabric allowing for RADAR to be seamlessly hidden behind a bumper for an aesthetic look. (Xx, Xx and Xxxx, n.d.)

2.6.2 4D RADAR Disadvantages.

By integrating data from various sensors, such as visual sensors and LIDAR, the system can enhance its understanding of the driving environment. Deep learning (DL) techniques have played a pivotal role in this, with a multitude of advanced deep neural networks (DNNs) being employed for perception tasks. Thanks to the substantial learning capacity of DL, there has been a significant improvement in the performance of these tasks. Many DL frameworks have been explored for processing both image and LIDAR data, as they provide ample data for training and validating deep neural networks. In contrast, research on RADAR-related DL studies has been limited, primarily due to the sparsity of RADAR data. (Zhou et al., 2020). For more research to be completed using RADAR more data is required. LiDAR excels in providing higher accuracy by being able to output over 100,000 points per frame while 3D RADAR outputs only 1,000 points per frame. (Hesai Webmaster, 2023)

Illustrated by **Figure 7**, certain objects that are clearly visible in LiDAR data, such as cars and pedestrians, may appear blurred or less well-defined when observed through RADAR data. (AR, 2021)
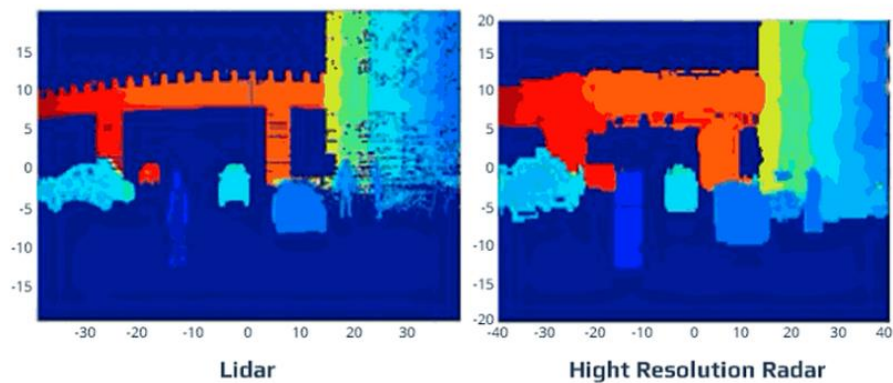
Figure 7: LiDAR VS RADAR Point Cloud (AR, 2021)

To conclude **Figure 8:** summarises some of the advantages of RADR over LiDAR, however, because of RADAR's limited resolution and the absence of semantic features, RADAR-based technologies "*for detection and tracking, vehicle self-localization, and HD map updating currently*" not as advanced as other perception sensors in highly autonomous driving. Nevertheless, research efforts in RADAR technology have been on the rise, thanks to the unique advantages offered by RADAR sensors. Enhancing the quality and imaging capabilities of millimetre-wave (MMW) RADAR data, along with exploring the full potential of RADAR sensors, is essential for gaining a comprehensive understanding of the driving environment. (Zhou et al., 2020)
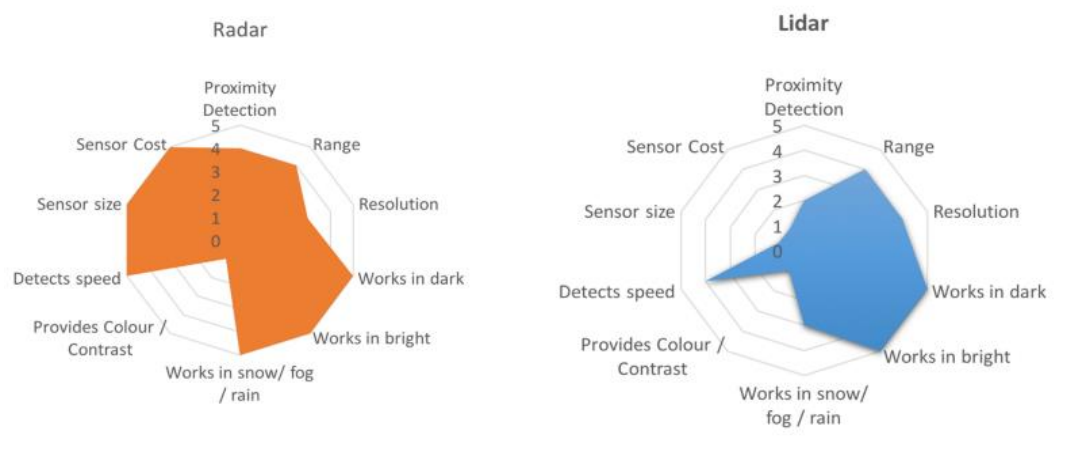
Figure 8: RADAR VS LiDAR Graph (Barnard, 2016)

2.7   Deep Learning

Deep learning models consist of a series of interconnected layers, like the human brain which consists of millions of interconnected neurons that cooperate to acquire knowledge. (Gillis, Burns and Brush, 2023) these layers create a continuous and trainable model using backpropagation. Researchers have extensively studied these layers and neural network structures in recent years, leading to enhancements in data feature representations. The utilization of extensively annotated datasets has enabled the training of models with larger parameter counts, resulting in improved performance on public benchmarks and challenges. Deep Learning applications employ a hierarchical set of algorithms referred to as an artificial neural network (ANN). The architecture of such an ANN is inspired by the biological neural networks found in the human brain, enabling a learning process that surpasses the capabilities of conventional machine learning models. (Levity.ai, 2023)

This section will focus on the state-of-the-art deep learning models for computer vision tasks such as object detection, tracking and classification.

2.7.1 Convolutional Neural Networks (CNNs).

Images captured by cameras are represented as 3D matrices with pixel values ranging from 0 to 255 for each of the Red, Green, and Blue (RGB) channels. When utilizing images as input for a machine learning model, it becomes computationally intensive. For instance, if we consider an image with dimensions 200x200x3, a single fully connected layer requires 120,000 parameters to process each pixel. This conventional ANN cannot efficiently handle high-dimensional input data and cannot directly learn spatial features from the input. (Ouaknine, 2022)

In recent years, Convolutional Neural Networks (CNNs) have been explored to address these challenges. CNNs utilize Convolutional layers that are comprised of a grid of neurons, with a requirement that the previous layer also consists of a grid of neurons in a rectangular shape. Each neuron in this layer receives inputs from a rectangular segment of the preceding layer, with the same set of weights applied to this rectangular segment for all neurons within the convolutional layer. Consequently, the convolutional layer essentially performs an image convolution operation on the previous layer, using these weight values to define the convolution filter. (Joel Markus Vaz and Balaji, 2021)
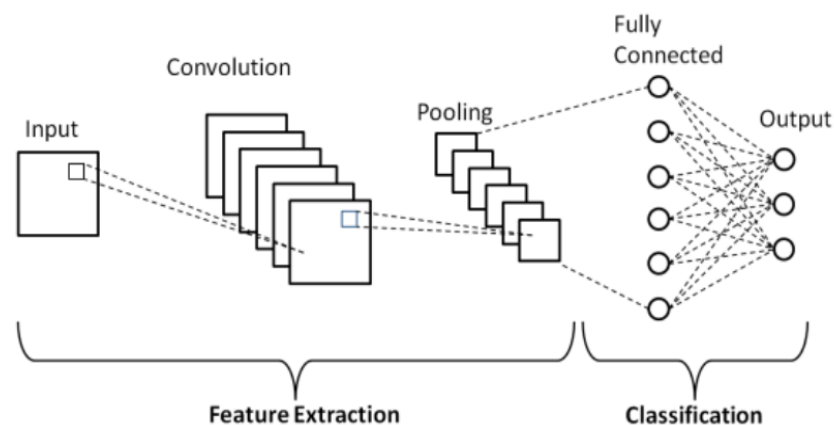
Figure 9: Diagram of a CNN (ResearchGate, 2019)

Within each convolutional layer, there can be multiple grids, each of which obtains inputs from all the grids in the preceding layer, utilizing potentially distinct filters. Following each convolutional layer, there is a pooling layer illustrated in **Figure 9**.

The pooling layer selects small rectangular blocks from the convolutional layer and reduces them to a single output from that block through subsampling. Various pooling methods can be employed, such as averaging, taking the maximum value, or using a learned linear combination of the neurons within the block. In this context, our pooling layers consistently utilize max pooling, which means they pick the maximum value from the block they are pooling. (Joel Markus Vaz and Balaji, 2021)

Finally, after a series of convolutional and max-pooling layers, the neural network's high-level reasoning occurs through fully connected layers. A fully connected layer connects all the neurons from the previous layer, whether it was a fully connected, pooling, or convolutional layer, to every neuron within itself. Fully connected layers do not have a spatial arrangement (they can be envisioned as one-dimensional), making it infeasible to introduce convolutional layers after a fully connected layer. (Gibiansky, 2014)

2.7.2 Recurrent Neural Networks (RNNs).

Recurrent neural networks (RNNs), like other deep learning methods, have been around since the 1980s, but their true capabilities have become evident only recently. The introduction of long short-term memory (LSTM) in the 1990s, along with greater computing power and the abundance of data, has propelled RNNs to the forefront of machine learning. (Kalita, 2022)

An RNN is a neural network designed for handling sequential data, and it finds application in various fields, including temporal series analysis, such as music, video, and stock market data, as well as Natural Language Processing tasks like textual analysis and translation.

In an RNN, the output from the previous step serves as input to the current step, unlike traditional neural networks, where inputs and outputs are treated independently. (GeeksforGeeks, 2018)
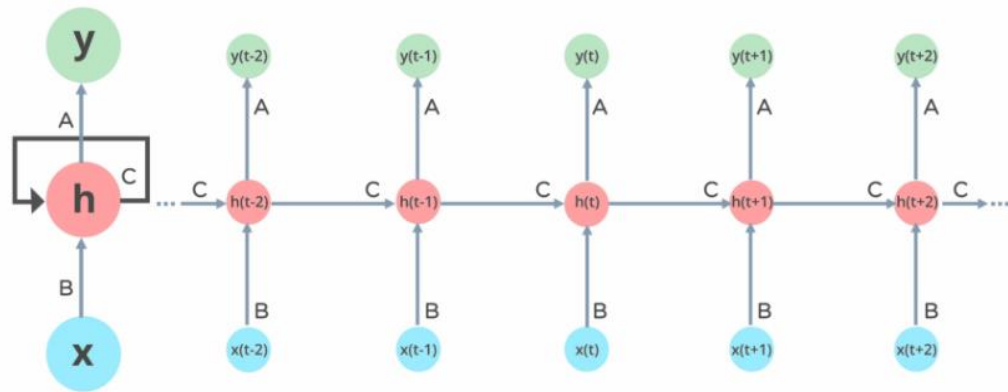
Figure 10: RNN Architecture (Avijeet Biswal, 2020)

Illustrated by **Figure: 10** "x" represents the input layer, "h" corresponds to the hidden layer, and "y" denotes the output layer. The network's performance is enhanced by utilizing parameters A, B, and C. At each time step "t," the present input is a composite of both the current input at x(t) and the previous input at x(t-1). The output at any given time is looped back into the network to refine the model's output. (Avijeet Biswal, 2020)

Recurrent Neural Network (RNN) models offer several advantages, such as their ability to handle sequences of varying lengths, their consistent number of parameters regardless of input size, and their capacity to incorporate historical context by sharing parameters across timestamps. Nevertheless, RNNs are hindered by their slow training speed and the issue of vanishing gradients, which tend to occur towards the end of sequences. Additionally, these models struggle to effectively capture long-term dependencies due to information loss during sequence processing. (Ouaknine, 2022)

2.7.3 Region-Based Convolutional Neural Network (R-CNN).

This approach begins by employing a selective search technique Initially, it segments an image into smaller regions and then merges them hierarchically using various colour spaces and similarity metrics (Jasper et al., 2013). This process yields a limited set of region proposals that may potentially contain objects of interest. The R-CNN model, proposed by (Girshick et al., 2016), combines this selective search method for region proposal generation with deep learning for object classification illustrated in **Figure 11.**
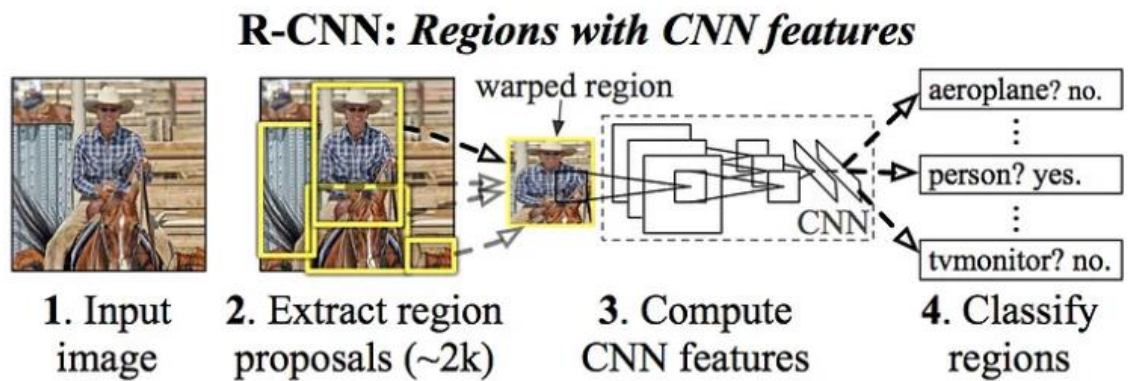
Figure 11: Diagram of a RCNN (Gandhi, 2018)

Each region proposal is resized to match the input requirements of a Convolutional Neural Network (CNN), which produces a 4096-dimensional feature vector. This feature vector is then used as input for binary Support Vector Machine (SVM) classifiers, one for each class. Additionally, it is used in a linear regressor to adapt the shapes of the corresponding bounding boxes, reducing location errors. (Hearst et al., 1998)

The CNN is trained on the 2012 ImageNet dataset for image classification and then fine-tuned using region proposals that have an Intersection over Union (IoU) greater than 0.5 with the ground-truth bounding boxes. Two versions of the model are created: one using the 2012 PASCAL VOC dataset and the other using the 2013 ImageNet dataset with associated bounding boxes. SVM classifiers are also trained for each class within both datasets. The top-performing R-CNN models have achieved a mean Average Precision (mAP) score of 62.4% in the 2012 PASCAL VOC challenge, representing a substantial 22.0-point improvement over the second-best result on the leaderboard. Furthermore, they achieved a 31.4% mAP score on the 2013 ImageNet dataset, surpassing the second-best result on the leaderboard by 7.1 points. (Ouaknine, 2022)

2.8   Object Detection & Tracking

Object detection represents a fundamental task in the realm of computer vision and holds significant importance in enabling autonomous driving. Autonomous vehicles heavily rely on their ability to perceive their surroundings, ensuring safe driving performance. To achieve this, they utilize object detection algorithms, which accurately identify objects such as pedestrians, vehicles, traffic signs, and barriers within their proximity.

Deep learning-based object detectors are instrumental in the real-time identification and localization of these objects. (Balasubramaniam and Pasricha, n.d.)

This section explores the current state of the art in object detection and highlights the open challenges associated with integrating these technologies into autonomous vehicles.

2.8.1 Traditional Object Detection Techniques.

The modern progression of object detection techniques was initiated two decades ago with the Viola-Jones detector, in 2001, Paul Viola and Michael Jones introduced an object recognition framework for real-time human face detection. This framework employs sliding windows to scan an image at various locations and scales to identify human faces. The search is based on "haar-like" features, named after Alfred Haar, who pioneered the concept of haar wavelets. These wavelets serve as the image's feature representation. To expedite detection, an integral image is utilized, ensuring that the computational effort for each sliding window remains independent of its size. (baeldung, 2022).

The authors also employ the Adaboost algorithm for feature selection, which identifies a small set of features that are particularly useful for face detection from a large pool of random features. Additionally, the algorithm incorporates Detection Cascades, a multi-stage detection approach aimed at reducing computational overhead. This means that it prioritizes less computation on background windows and focuses more on potential face targets. (Borah, 2020)

A few years later, the Histogram of Oriented Gradient (HOG) detectors gained popularity, especially for detecting pedestrians (Tyagi, 2021). These HOG detectors were subsequently expanded into Deformable Part-based Models (DPMs), representing the first models geared towards the detection of multiple objects. (Davies, 2022)

Around 2014, the surge in interest surrounding deep neural networks led to a significant breakthrough in multiple object detection with the introduction of the Regions with Convolutional Neural Network (R-CNN) deep neural network model. This innovation resulted in a remarkable 95.84% enhancement in Mean Average Precision (mAP) over the existing state-of-the-art methods.

This pivotal advancement not only redefined the effectiveness of object detectors but also made them appealing for entirely new application domains, particularly in the context of Autonomous Vehicles (AVs).

Since 2014, the continued evolution of deep neural networks and the progress in Graphics Processing Unit (GPU) technology have paved the way for faster and more efficient object detection in real-time images and videos. Modern AVs heavily rely on these improved object detectors for various crucial tasks, including perception, path planning, and other decision-making processes. **Figure 12** illustrates the taxonomy of object detectors.
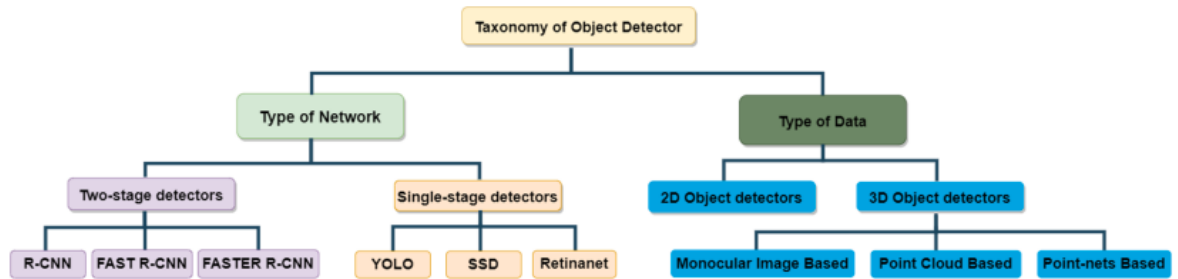
Figure 12: Taxonomy of object detectors (Balasubramaniam and Pasricha, 2022)

2.9   Object Classification

One of the widely pursued objectives in the field of computer vision involves the task of classification, which entails assigning a specific category to every image within a dataset. The ImageNet dataset, introduced by (Deng et al., 2009), has played a significant role in extensive exploration and research on classification. A vector is used to measure and represent the local characteristics of an image, summarizing these features in the form of a histogram that describes the overall image. This section will provide an overview of the evolution and improvements made over the years in image classification.

This section discusses the accomplishments in deep learning related to image classification, specifically in the context of the ImageNet challenge. It provides detailed information about well-known modules and architectures that are still widely employed for feature extraction today. It is important to note that this isn't an exhaustive catalogue of all the models developed between 2012 and 2018. Furthermore, the more recent advancements, particularly in Transformer architectures as proposed by (Dosovitskiy et al., 2020) fall outside the scope of this thesis.

2.9.1 AlexNet (2012).

A deep convolutional neural network known as AlexNet, created by Alex Krizhevsky and his team in 2012, emerged victorious in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This groundbreaking architecture featured numerous convolutional layers and served as a pioneering example of how deep learning could be harnessed for image classification. (Nitin Kushwaha, 2023)

2.9.2 VGGNet (2014).

The VGGNet design, created by the Visual Geometry Group at the University of Oxford, focused on creating deeper neural network architectures. Its straightforward and consistent structure set a standard for investigating network depth. (Deepchecks, 2021)

2.9.3 ResNet (2015).

Residual Networks, commonly known as ResNets, revolutionized deep neural networks by introducing the concept of residual connections, effectively tackling the issue of vanishing gradients. This breakthrough enabled the successful training of extremely deep networks, leading to the practical realization of ResNets with hundreds of layers, resulting in significant performance enhancements. They achieved remarkable success by securing the top position in the ILSVRC 2015 classification competition with a top-5 error rate of 3.57%, using an ensemble model. (Great Learning Team, 2020)

2.9.4 DenseNet (2017).

DenseNet introduced a novel approach to connectivity patterns by promoting dense connections, which encouraged the reutilization of features and facilitated the flow of gradients. This architectural innovation showcased enhanced training effectiveness and increased accuracy in the context of image classification tasks. (Paperswithcode.com, 2020)

2.10 Related Experimental Results

The following section is a summary of some results acquired from the literature to aid in choosing the model and approach to develop this project.To emphasize the importance of the 4DRT-based perception module the author of this paper, (Paperswithcode.com, 2022) introduced a basic neural network for 3D object detection (referred to as the baseline NN) that takes 4DRT as its input. Their experiments on the K-RADAR dataset reveal that the 4DRT-based baseline NN excels in the task of 3D object detection, particularly in challenging weather conditions, outperforming the LiDAR-based network.

In another study carried out by (Scheiner et al., 2021), the objective is to perform object detection on automotive RADAR point clouds to identify moving road users using two end-to-end object detectors (YOLOv3 and PointPillars). YOLOv3 performs the best with a mean Average Precision (mAP) of 53.96%, offering the potential for combining static and dynamic object detection in the future.

PointPillars, a point-cloud-based object detector, falls behind with a 36.89% mAP. Improving its performance to 45.82% by using an enhanced CNN backbone still falls short of YOLOv3-like results. While point-cloud-based detectors have potential, they are not yet on par with image-based variants. However, ongoing model development may bridge this performance gap, offering increased flexibility for various sensor types and improved speed in the future.

Another study comparing SSD and Faster R-CNN on RADAR Imagery done by (Stroescu et al., 2021), Concluded the training loss for SSD consistently decreases with each training epoch, the Faster R-CNN loss exhibits significantly lower values, leading to higher mean Average Precisions (mAPs). One plausible reason for the superior performance of Faster R-CNN can be learned from a study done by (Huang et al., 2016), which delves into the trade-off between speed and accuracy among various detectors on the COCO dataset. This research found that Faster R-CNN excels in accuracy, whereas SSD, though faster, struggles significantly when it comes to detecting small objects.

2.11 Conclusion

The literature review comprehensively explores the intricate landscape of autonomous vehicle perception technologies, focusing particularly on the integration and functionalities of various sensor systems such as LiDAR, radar, and computer vision aided by deep learning.

This exploration underscores the pivotal role of sensor fusion in enhancing the accuracy and reliability of autonomous systems in diverse driving conditions.

Significant attention is dedicated to the evolution and capabilities of radar technologies, especially 4D radar, which provides enhanced data on object detection and environmental understanding, crucial for autonomous navigation. The review critically assesses the comparative strengths and weaknesses of radar and LiDAR technologies, highlighting radar's robustness in adverse weather conditions and LiDAR's superior precision in object detection.

Deep learning models, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are discussed extensively as they represent the backbone of object detection and classification systems in autonomous vehicles. These models enhance the vehicle's ability to interpret complex scenes, contributing to safer driving decisions.

The chapter concludes that while substantial progress has been made in sensor technology and machine learning models, challenges remain, particularly in the fusion of data from different sensors to achieve seamless and reliable vehicle autonomy. Future research should focus on improving algorithms for data integration and real-time processing to enhance the adaptability and safety of autonomous vehicles.

This conclusion encapsulates the core findings and outlook as discussed in the literature review, setting a clear pathway for the subsequent chapters of the thesis.

Chapter 3 Analysis and Design

3.1 Application Overview

Object detection from 4D RADAR point clouds is a very challenging task for a final year thesis and it presents a massive set of unique challenges. These challenges stem from the nature of RADAR data, the complexity of the PV-RCNN architecture and the lack of industry knowledge in such a specific niche of software development for an undergraduate. This analysis aims to highlight these challenges to minimise the risk of running into too many blockers during the implementation phase.

3.2 Potential Blockers and Considerations

The following section discusses the potential blockers and considerations to be made before starting on the project.

3.2.1 Data Sparsity and Quality

RADAR point clouds are inherently sparser than those obtained from LiDAR, which can make it difficult to detect and classify objects with high accuracy. Furthermore, RADAR data can be affected by noise and clutter from the environment, such as reflections from ground surfaces or nearby objects, making it challenging to isolate and identify relevant features for object detection.

3.2.2 Dataset Acquisition

Accessing datasets with 4D RADAR data and ground truth annotations, essential for developing and testing object detection models like PV-RCNN, presents a significant challenge. The limited availability of such datasets is often due to their proprietary nature and the competitive advantage they confer within the automotive and tech sectors. Companies heavily invest in sensor technology and data collection, treating their datasets as key assets to maintain competitiveness. Consequently, the protective stance on data sharing impedes collaborative research and development, especially for smaller entities and academics, slowing progress in object detection and autonomous vehicle technologies.

3.2.3 Computational Complexity

The advanced object detection models tailored for 4D RADAR point clouds demand high-end computing power, often requiring multiple GPUs due to their computational complexity, which involves voxelization, feature extraction, and integrating point and voxel-level data. This need for substantial computational resources can result in prolonged training periods and slow development, potentially limiting the project's scale and scope. The iterative nature of machine learning, with its need for numerous experiments and parameter adjustments for optimal performance, exacerbates this challenge, especially for those without access to the computing capabilities of well-funded organizations.

3.2.4 Steep Learning Curve

The challenge of navigating a niche and technically complex task of object detection using 4D RADAR point clouds is particularly daunting for an undergraduate student, largely due to the steep learning curve and the lack of accessible industry knowledge. This specialized domain not only requires a deep understanding of advanced concepts in RADAR technology, computer vision, and machine/deep learning but also necessitates insights into the current state-of-the-art and industry practices that are often not available in academic curricula or publicly accessible resources. The proprietary nature of much of the research and development in this area further exacerbates the difficulty in obtaining relevant and up-to-date information. This means that beyond mastering the technical skills, there's an additional layer of challenge in simply understanding the context, applications, and potential limitations of this project.

3.3 Scope

The project's scope for tackling the task of object detection, tracking and classification will need to be deliberately concentrated on the detection and tracking of objects in point clouds, due to the challenges explained in section [3.2] and due to time constraints. This strategic limitation is critical to managing complexity, as it allows for a more manageable development process and provides a strong basis for future expansion into tracking and classification once detection is mastered.

3.4 Prerequisites

The following courses were done to gain a base knowledge for tackling this project:

1. Udemy. (2024). *Advanced Driver Assistance Systems (ADAS)*. [online] Available at: https://www.udemy.com/course/advanced-driver-assistance-systems/learn/lecture/24251478?start=0#overview
2. Udemy. (2024). *Automotive Radar*. [online] Available at: https://www.udemy.com/course/automotive-radar-basics-to-advance/learn/lecture/20249534?start=15#overview
3. Udemy. (2024). Python for Computer Vision with OpenCV and Deep Learning. [online] Available at: https://www.udemy.com/course/python-for-computer-vision-with-opencv-and-deep-learning/learn/lecture/12257624?start=0#overview
4. Udemy. (2024). *Deep Learning: Recurrent Neural Networks in Python*. [online] Available at: https://www.udemy.com/course/deep-learning-recurrent-neural-networks-in-python/learn/lecture/21514852?start=0#overview

3.5 Software Development Process

This section describes the development process undertaken by the author.

3.5.1 Agile

The author was grounded in Agile methodology, illustrated by **Figure 13**. This approach was particularly suitable given the complexity and the exploratory nature of designing such an advanced object detection and tracking model. Agile's iterative development cycles, known as sprints, allowed the author to break down the immense task into manageable segments, each delivering progress in increments. Embracing Agile's adaptive planning and development also provided the flexibility to respond to changes, which is crucial in a cutting-edge field where discoveries and knowledge alter the course of development. This methodology, with its emphasis on continuous learning and adjustment, was essential in navigating the challenges of this project.

Key aspects of Agile methodology suited to this project are:

- An iterative and incremental approach to project management

- Emphasis on adaptability and flexibility

- Breaking down projects into smaller, manageable tasks

- Setting short-term goals and adjusting as needed

- Incorporating regular feedback and open communication.
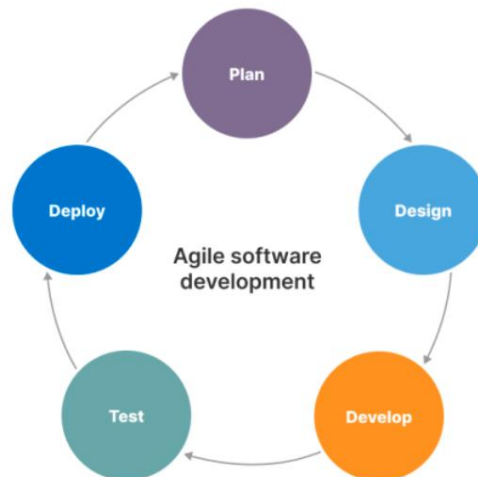

Figure 13: Agile Diagram (Aha, 2024)

3.5.2   Minimal viable product

Illustrated by **Figure 14** A Minimal Viable Product (MVP) is essentially a streamlined version of a product, designed to be functional enough for an initial release. It includes only the most essential features that define the product's core purpose, aiming to meet the needs of the first set of users, often referred to as early adopters. The primary objective of an MVP is to launch quickly and cost-effectively, allowing the development team to collect and analyse user feedback at an early stage. This feedback is invaluable as it guides subsequent development, ensuring that further enhancements and features are aligned with actual user needs and preferences. By focusing on the essentials, an MVP helps in validating product-market fit, minimizing initial investment, and reducing the risks associated with product development.
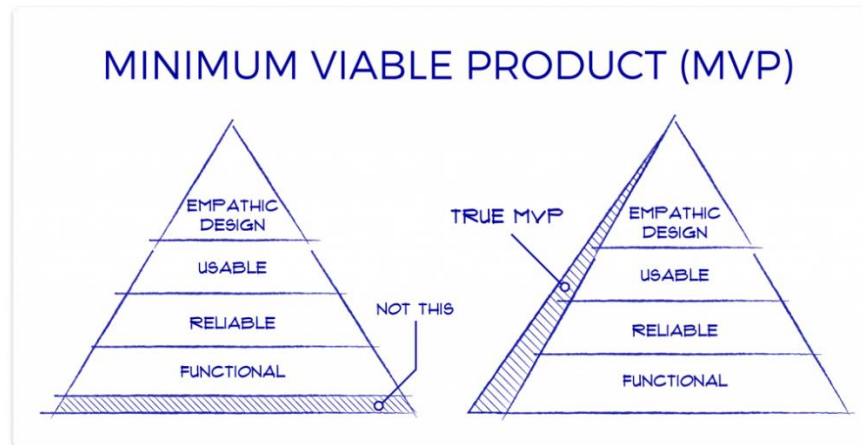
Figure 14: MVP Diagram

3.5.3   Project Management

The author leveraged ClickUp to implement agile methodology and manage the project by breaking down the workload into tasks and subtasks with urgency levels, and due dates and assigning the tasks to sprints over the development process. ClickUp is a versatile project management and productivity tool that enables teams or solo developers to organize tasks, collaborate on projects, and track progress in a unified platform, facilitating the use of agile for efficient project development. ClickUp is a free alternative tool to Jira, which is the software of choice used by the company where the author did his placement.

3.6   Tools and Framework Considered

This section discusses the considered tools and frameworks to undertake this project.

3.6.1   Python

What is Python?

Python is a powerful, high-level programming language that aims to be easy to read and understand. The term "high-level" refers to how closely related to human languages it is in comparison to other computer languages. Since the Python community is vibrant and active, learning this language will provide developers access to a wealth of useful tools that Python users have created since its inception.

Why Python?

Python's appeal to developers stems from its versatility across numerous fields like data science, AI, web and desktop app development, statistics, math, and scientific studies. The abundance of open-source libraries and the continuous growth of easy-to-use tools contribute to its popularity. The language is known for being beginner-friendly, which helps its already large development community grow even faster. Moreover, the vast Python community ensures that support, advice, and solutions are readily available for anyone encountering challenges with their projects.

What does Python's simplicity mean?

The best way to highlight Python's simplicity and user-friendliness is by contrasting how it and other programming languages, such as Java and C++, tackle the same basic task. Consider the quintessential programming task of displaying "Hello World!" in the terminal as an example.

C++	Java	Python

```1. #include <iostream>```
```2. int main()```
```3. {```
```4.```
```std::cout<<"Hello,world!\n";```
```5.   return 0;```
```6. }```
	```1. class HelloWorldApp {```
	```2. public static void```
	```main(String[] args) {```
	```3. System.out.println("Hello```
	```World!");```
	```4. }```
	```5. }```
		```1. print("Hello```
		```World")```
		```2.```

3.6.2 Jupiter Notebook

Jupyter, short for Julia, Python, and R, began with these languages but now supports many more. The Jupyter Notebook is a free, open-source web application that allows for the sharing and collaborative editing of programming work. It enables programmers to create "notebooks," interactive documents that blend code, commentary, multimedia, and visuals, facilitating code execution directly in a web browser and proving valuable for educational demonstrations. The following is a list of reasons as to why someone would use notebooks:

1. Interactive Environment

Jupyter allows for real-time code execution, making it ideal for exploratory data analysis and iterative changes, enabling users to test hypotheses and adjust data instantly.

2. Supports Multiple Languages

While it started with Python, Jupyter supports multiple programming languages like R, Julia, and Scala, making it adaptable for various projects.

3. Combines Code, Text, and Media

It integrates code, narrative text, and multimedia in one document, allowing for interactive reports and presentations that are both informative and engaging.

4. Ease of Sharing

Notebooks can be easily shared, capturing code, outputs, and notes in a single file, which simplifies collaboration and helps in reproducing results.

5. Rich Ecosystem

Jupyter is supported by a vast community that contributes plugins and extensions, enhancing its functionality for tailored workflow needs.

In conclusion, Jupyter Notebook is a versatile tool that caters to a wide range of applications from data analysis and scientific research to education and report generation. Its ability to combine explanation, live code, and visual output into one cohesive document makes it uniquely positioned to enhance productivity and foster an environment of open, reproducible scientific inquiry.

3.7   Dataset

Several datasets with 4D radar have been released and applied in recent years. However, despite the lack of publicly available datasets, an analysis was conducted on the available datasets. From the analysis, the author created the following summary:

Astyx, an early-released dataset, provides rich data for 3D object detection but is limited by its small size of 546 frames and 3000 object annotations, lacking special scenarios and urban data. RADIal offers a medium-scale dataset with urban streets and highways but lacks 3D bounding boxes and tracking IDs and does not cover adverse weather conditions. View-of-Delft addresses the object tracking problems present in the other datasets with 8,693 frames and 120,000 annotated objects but has a short detection range and lacks 4D radar information for long-range mode. TJ4DRaDSet includes various driving scenarios but lacks data in middle and short-range modes and scenarios with adverse weather conditions. K-Radar provides rich driving scenarios with adverse weather conditions but lacks 4D radar point clouds in long-range mode.

View-of-delft was chosen for this experiment setup as it "consists of more than 123000 3D bounding box annotations, including more than 26000 pedestrian, 10000 cyclist and 26000 car labels." (GitHub, 2024) Which is great for an object detection problem.

The View-of-delft researchers have an open-source GitHub repository containing useful guides to their dataset along with visualisation and evaluation scripts they used when conducting their research, this repository would come in very useful when exploring the dataset and getting familiar with the process of doing object detection with 4D radar point clouds.

3.7.1   View-Of-Delft Sensor Setup

Illustrated by Figure 15, the RADAR sensor utilized is a ZF FRGen21 3+1D RADAR, operating at approximately 13 Hz, and is mounted behind the vehicle's front bumper. The provided RADAR point clouds undergo ego-motion compensation to account for any motion between RADAR and camera data capture, ensuring consistency when overlaying both datasets. (GitHub, 2024)

Figure 15: Prius Sensor Setup VOD (GitHub, 2024)

3.7.2 Dataset Frame Information

These RADAR point clouds are stored in bin files, with each file containing a set of points represented as an Nx7 array, where N is the number of points, and 7 denotes the number of features: *[x, y, z, RCS, v_r, v_r_compensated, time]*. Here, *"v_r"* signifies the relative radial velocity, *"v_r_compensated"* indicates the absolute radial velocity compensated for ego-motion, and "*time*" denotes the point's time ID, indicating its originating scan (GitHub, 2024).

3.7.3 Dataset Label Information

Sensor fusion in autonomous systems integrates data from multiple sensors like LiDAR and radar to enhance accuracy and robustness, utilizing LiDAR's high-resolution for precise object labelling which is applied to radar data for additional insights such as velocity. Aligning sensor data in both space and time is crucial, often using LiDAR as the benchmark for its detailed spatial data. LiDAR's clear 3D point clouds simplify labelling tasks, with labels easily transferred to radar data for consistency. The dataset's structure prioritizes LiDAR, reflecting its initial design focus and convenience in labelling, leading to radar data being adjunct and reliant on LiDAR for annotations.

For frame number 1047, the first line of the label information could be interpreted as follows:

- **Object Type:** rider.
- **Truncation**: 1 (possibly indicating high confidence or manual verification)
- **Occlusion**: 0 (fully visible)
- **Observation Angle**: 1.716500830699201
- **Bounding Box**: 979.41486 789.5281 1018.06165 866.89154
- **3D Object Dimensions**: 1.503325462332693 (height), 0.7167884312694952 (width), 0.6358283468841199 (length)
- **3D Object Location**: 0.7805723338707173 (x), 4.960184749066411 (y), 31.026849236059597 (z)
- **Rotation Y**: -4.541531818868102
- **Score**: 1 (high confidence or manually verified)

3.8   Data Visualization and Exploration

Before any work can be carried out, the radar scans need to be visualised in a point cloud for easier interpretation and comparison of this visualisation to the output of the camera. K3D was used by VOD, however, the author chose Open3D because it is preferred for its extensive feature set, active community support, cross-platform compatibility, seamless Python integration, and ongoing development. These factors make it a versatile choice for various 3D data processing tasks compared to K3D. Once the point cloud is visualised its good practice to get the annotations from the dataset and attempt to draw the 3D bounding boxes inside of the point cloud to be able to interpret the different objects (car, pedestrian, cyclist etc.) present in the scene.

It would also be beneficial to calculate the number of points within a specific scene and any output any other relevant information.

3.9 Model Architecture Considered

Due to the scarcity of open-source projects focused on 4D RADAR object detection, which can be attributed to the novelty of this technology. The PV-RCNN architecture from the OpenPCDet library was considered as a foundational guide and reference point. OpenPCDet is primarily a toolbox designed for 3D object detection using lidar technology. By choosing this established framework, the author aims to adapt and extend its capabilities to handle 4D RADAR data effectively.
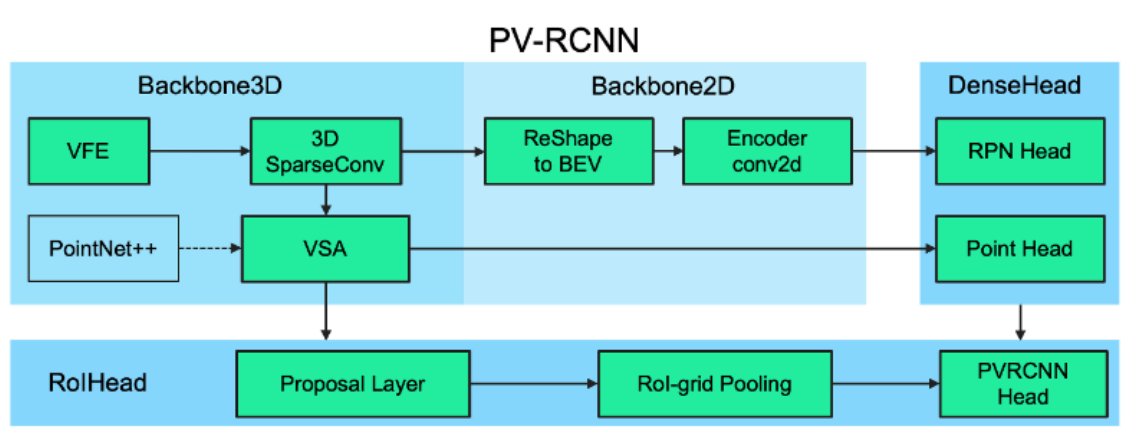
Figure 16: OpenPCDet PV-RCNN Architecture Diagram (GitHub, 2024)

The PV-RCNN architecture combines 3D voxel CNNs and PointNet for precise 3D object detection from point clouds. It leverages voxel CNNs for efficient feature learning and high-quality proposal generation while incorporating Point Net's ability to capture detailed contextual information. The framework introduces voxel-to-key point scene encoding to condense scene features into key points and point-to-grid RoI feature abstraction for refining proposal confidence and location, effectively integrating the strengths of both network types for enhanced performance.

3.9.1 PV-RCNN Components Overview

In this section, the author deconstructs the architecture into its primary components, as illustrated in **Figure 16**, to simplify it for the reader's comprehension. The primary components of the PV-RCNN are as follows:

1. **VFE (Voxel Feature Encoding):** This is the initial stage where raw point clouds are converted into a structured voxel representation. The VFE learns to encode the raw point cloud data into a fixed-size feature representation for each voxel, which can capture the essential information while reducing the sparsity of point clouds.

2. **Backbone3D**

   3D SparseConv: Once the point clouds are voxelized, the 3D Sparse Convolutional Network processes these voxels to generate high-dimensional sparse feature volumes. It is designed to efficiently handle the sparsity in voxelized point clouds by only computing features at occupied voxels.

   VSA (Voxel Set Abstraction): The VSA module is responsible for summarizing the voxelized features into a smaller set of key points. This abstraction process leverages the strength of PointNet++ to learn a more discriminative feature representation from the unordered point set within each voxel.

3. **Backbone2D**

   Reshape to BEV (Bird's Eye View): The high-dimensional features from the 3D backbone are reshaped into a 2D representation corresponding to the bird's eye view of the scene. This process essentially projects the 3D features onto a 2D plane.

   Encoder conv2d: This 2D feature map is then processed by a 2D convolutional encoder which further refines the features and captures spatial relationships in the bird's eye view.

4. **DenseHead**

   RPN Head (Region Proposal Network): The RPN Head uses the refined 2D feature maps to generate region proposals. These proposals are candidate regions where objects might be present.

   Point Head: Parallel to the RPN Head, the Point Head processes the point features from the VSA to generate keypoint features that describe the objects in the scene.

5. **RoI Head**

   Proposal Layer: This layer takes the region proposals from the RPN Head and the key point features from the Point Head to generate accurate 3D bounding box proposals.

RoI-grid Pooling: The RoI-grid Pooling module aggregates the key point features onto a structured grid within each 3D RoI using set abstraction operations. This step is crucial for capturing the local context around each proposal.

6. **PVRCNN Head:** Finally, the PV-RCNN Head combines the features from the RoI-grid Pooling module with the proposals from the RoI Head to refine the proposals and predict the final bounding boxes along with the object classification.

3.10 Training and Inference Details

The PV-RCNN framework from OpenPCDet undergoes training from the ground up in a seamless end-to-end process utilizing the ADAM optimization algorithm. The researchers utilized the KITTI dataset to train the entire network using a batch size of 24 and a learning rate of 0.01 across 80 epochs. This training is performed on 8 GTX 1080 Ti graphics processing units and is completed in approximately 5 hours.

Given that the author of this thesis, training setup consists of a laptop with a single Nvidia GTX 1660 Ti GPU, the author may need to consider downsizing the dataset and be prepared for the training duration to extend over several days. This is due to the lower computational power compared to the original setup with 8 GTX 1080 Ti GPUs. Alternatively, to maintain the scale of the dataset and potentially accelerate the training process, the author could leverage a cloud computing service that offers more powerful GPU resources. This would allow me to conduct the training more efficiently, albeit at a potential increase in cost. The author will also need to experiment with hyperparameters such as the epochs.

3.11 New Architecture Considered

Given time constraints, the myriad of challenges encountered and the release of a new library RaTrack on March 13th a decision was made to switch to the architecture from OpenPCDet's PV-RCNN to RaTrack. This pivot not only addressed the immediate needs but also set a new course with the intention of contributing to and enhancing the RaTrack repository. The aim now is to actively contribute to the RaTrack project and build on top of it, as well as experiment with model training configurations.

RaTrack introduces a novel approach illustrated by **Figure 17**, that emphasizes motion segmentation and clustering over the conventional tracking-by-detection model. The system utilizes a point-wise motion estimation module to enrich RADAR data with motion vectors, improving the detection and tracking of moving objects. The method sidesteps the need for specific object type identification and 3D bounding boxes, which are challenging to determine accurately from RADAR data. (Pan et al., 2023)
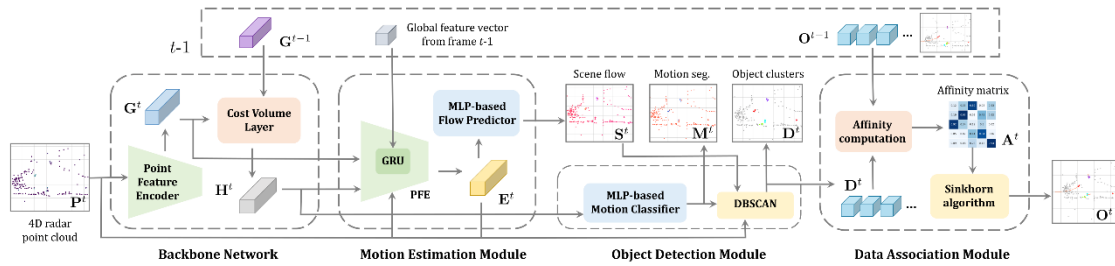
Figure 17: RaTrack Pipeline (GitHub, 2024)

Breaking down RaTrack modules in detail:

Backbone Network:

- Point Feature Encoder (PFE): Encodes local-global features from the 4D RADAR point cloud.

- Cost Volume Layer: Correlates features across consecutive frames to capture inter-frame motion information.

Motion Estimation Module:

- Uses the backbone's output to estimate point-wise scene flow, which describes the motion of each point from the current frame to the previous one.

- GRU (Gated Recurrent Unit): Integrates temporal information to enrich the motion estimation.

Object Detection Module:

- MLP-based Motion Classifier: Classifies points as either moving or static based on the backbone's features.

- Motion Segmentation: Segregates moving points from static points.

- Clustering (DBSCAN algorithm): Clusters moving points to form detected moving objects.

Data Association Module:

- Affinity Computation: Computes an affinity matrix to represent the similarity between detected objects and previously tracked objects.

- Sinkhorn Algorithm: A differentiable approach used for optimizing bipartite matching between frames, thus maintaining object identities over time.

3.12 Evaluation Metrics

In the context of 4D RADAR for object detection and tracking, "scene flow" and "segmentation" are two distinct aspects of the overall perception task, each focusing on different attributes of the environment and serving different purposes.

Scene Flow Evaluation:

Scene flow refers to the 3D motion field of points within a scene, indicating how each point moves from one frame to the next in 3D space. In the case of 4D RADAR, which captures both spatial and temporal dimensions (the fourth dimension being time), scene flow would involve analysing the movement of detected objects over time.

When evaluating scene flow, you're assessing the model's ability to accurately predict the motion of each object or point across successive frames. This is crucial for understanding the dynamics of the scene, predicting future states, and making decisions based on object trajectories.

Segmentation Evaluation:

Segmentation, on the other hand, involves dividing the RADAR's spatial data into segments corresponding to different objects or areas of interest. In object detection and tracking, segmentation would help in identifying the boundaries and extent of objects, separating them from the background or other objects.

Evaluation of segmentation focuses on how well the model can classify each point in the RADAR data. It's about spatial precision at a single time point, rather than movement over time.

What the metrics tell us:

In practice, for 4D RADAR object detection and tracking:

Good scene flow prediction allows the system to anticipate where objects will move, which is vital for collision avoidance systems and autonomous navigation. Accurate segmentation is essential for identifying what the objects are and where they are located precisely, which is necessary for object recognition, counting, and tracking stationary objects. The results for scene flow and segmentation are discussed in [Chapter 5].

3.13 Conclusion

Chapter 3 concludes by emphasizing the need for a structured and well-equipped approach to tackle the inherent challenges of working with 4D RADAR data in autonomous vehicle technology. It outlines the planned methodologies, tools, and frameworks to be used in overcoming these challenges, setting a foundational blueprint for the practical phases of implementation and testing described in subsequent chapters. The chapter effectively sets the stage for addressing the technical demands and educational gaps through a systematic development approach.

Chapter 4 Solutions

4.1 Introduction

The chapter's primary objective is to give the reader a thorough grasp of the project's implementation procedure and the technologies employed. This chapter will cover the hardware and software utilised and the factors in its selection. It describes the project's setup and highlights the components of technical interest to the project, along with the challenges faced and how they were addressed.

4.2 Source Control

Git is the source control technology used, and the source is maintained in a GitHub repository @ https://github.com/nicktmv/four-d-radar-thesis

4.3 Installation

To install and run the project.

1. Clone the repository.

```
git clone https://github.com/nicktmv/four-d-radar-thesis.git
```

2. Install the Python dependencies.

```
pip install -r requirements.txt
```

3. To obtain the full VOD dataset used please navigate to: https://github.com/tudelft-iv/view-of-delft-dataset under the heading "Access" are instructions on how to get access to the dataset, this involves filling out a form from which you will receive a download link.

4. Refer to section [4.5.2.1] for a full breakdown of the setup and installation of the RaTrack library and source code.

4.4 OpenPCDet Implementation

OpenPCDet is a freely available library designed for 3D object detection using LiDAR. Developed by the OpenPCDet Development Team and based on PyTorch, it aims to be flexible and efficient. The library accommodates multiple cutting-edge 3D detection models and features a modular architecture that enables researchers and developers to create their custom models.

4.4.1 Tools Used

This section outlines the author's choice of software tools, frameworks, and libraries used to carry out the research and project implementation.

4.4.1.1 Integrated Development Environment (IDE)

The author chose PyCharm as it is often the preferred IDE for implementing complex projects like implanting a PV-RCNN architecture for 4D RADAR due to its comprehensive Python-centric features. It offers an integrated environment tailored for Python development, including smart code navigation, advanced debugging, and refactoring tools, which significantly enhance productivity and code quality. PyCharm's support for scientific libraries and frameworks, such as TensorFlow and PyTorch, streamlines the machine learning workflow, from data exploration to model training and evaluation. The IDE's virtual environment management simplifies dependency handling, ensuring project consistency. Moreover, PyCharm's powerful visualization capabilities, coupled with its interactive Python console, facilitate the examination and interpretation of 4D RADAR data, making it an invaluable tool for researchers and developers working on advanced RADAR object detection systems.

The author effectively utilized Jupyter Notebooks to visualize results, providing an interactive platform that allows for a dynamic display of data and findings. This integration not only enhanced the clarity and interpretability of the outcomes but also facilitated a more engaging and accessible way for others to explore the analytical processes and insights derived from the data.

4.4.1.2   Ubuntu

After an initial attempt was made to set up the OpenPCDet repository on a Windows system, which involved resolving library dependencies issues and running any needed setups scripts. The author found an external resource that stated this repository is only supported on Linux (Alifya Febriana, 2023), despite there being no mention of this in the OpenPCDet repository. A Linux environment was set up on an Ubuntu Virtual Machine.

4.4.1.3   Libraries

All libraries are defined inside of requirement.txt files and can be installed running:

```
1. pip install -r requirements. txt
```

An outline of the main libraries needed:

Previously mentioned in section [3.8] **Open3d** was chosen for visualising the point cloud.

Numpy: A fundamental package for scientific computing with Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Numba: An open-source JIT compiler that translates a subset of Python and NumPy code into fast machine code, significantly accelerating execution.

PyTorch (2.0.1+cu117): A deep learning library that provides a flexible and powerful array library, Tensor, with GPU acceleration and automatic differentiation capabilities for building and training neural networks.

TensorboardX: An extension to TensorBoard, providing visualization and tooling needed for machine learning experimentation, such as tracking and visualizing metrics, and model graphs.

PyYAML: A YAML parser and emitter for Python, enabling easy reading and writing of YAML files for configuration or data serialization.
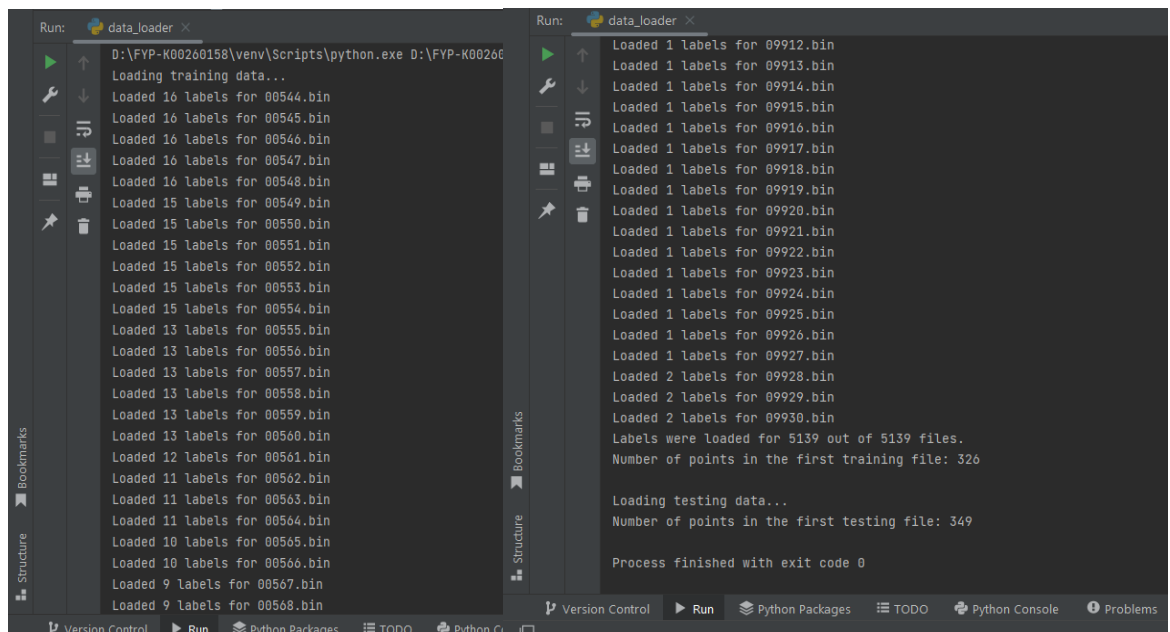
Scikit-image: A collection of algorithms for image processing in Python, providing tools for image manipulation and analysis.

Torchvision: A package consisting of popular datasets, model architectures, and common image transformations for computer vision.

OpenCV-Python: A Python wrapper for OpenCV, offering access to a wide range of image processing and computer vision functions.

4.4.2 Data Loader & Split

To facilitate model training and validation on the dataset which is 13.6 GB in total, it is partitioned into training and testing subsets. This partitioning is guided by predefined indices listed in the 'train.txt' and 'test.txt' files, ensuring a consistent and reproducible split across different experimental runs. This approach adheres to best practices in machine learning by preventing data leakage and ensuring the model's performance is evaluated on unseen data.

Figure 18: Data Loader output.

Illustrated by **Figure 18**, An output for the number of labels loaded for each .bin file in the training set was displayed. Then the loader gets the total number of the labels loaded for the training set, along with the number of points in the first point cloud. Lastly, the loader attempts to load in the testing set and prints out the number of the points in the first .bin file.

During the data loading process an assumption is made *"Assuming 7 features (which is correct since the point cloud is an N7 array) x 4 bytes each"*.

This assumption that each feature is 4 bytes is confirmed in check_data_structure.py located in four_d_radar/utils/check_data_structure.py by the following code:

```
1. # Ensure the file exists
2.    if not file_path.exists():
3.        print(f"File not found: {file_path}")
4.        return
5.
6.    try:
7.        with open(file_path, 'rb') as file:
8.            data = file.read()  # Read the entire file
9.
10.        # Calculate the number of points and the remainder
11.        num_points = len(data) // DATA_POINT_SIZE
12.        remainder = len(data) % DATA_POINT_SIZE
13.
14.        # Print results
15.        print(f"Total data length in bytes: {len(data)}")
16.        print(f"Number of points calculated: {num_points}")
17.        print(f"Remainder = {remainder} (should be 0 if assumption is correct)")
18.
19.        # Check a few points to see if the assumption holds
20.        for i in range(min(num_points, NUM_POINTS_TO_CHECK)):
21.            point_data = data[i * DATA_POINT_SIZE:(i + 1) * DATA_POINT_SIZE]  #
Extract the data for one point
22.            point = struct.unpack(f'{NUM_FEATURES}f', point_data)  # Unpack the point
data
23.            print(f"Point {i + 1}: {point}")
24.
25.    except Exception as e:
26.        print(f"An error occurred: {e}")
27.
```

Verified by using a frame 00544, the assumption is true as the remainder should ideally be 0 if the assumption is that each point is exactly 28 bytes, which is correct, illustrated by **Figure 19**.

Figure 19: check_data_structure.py output.
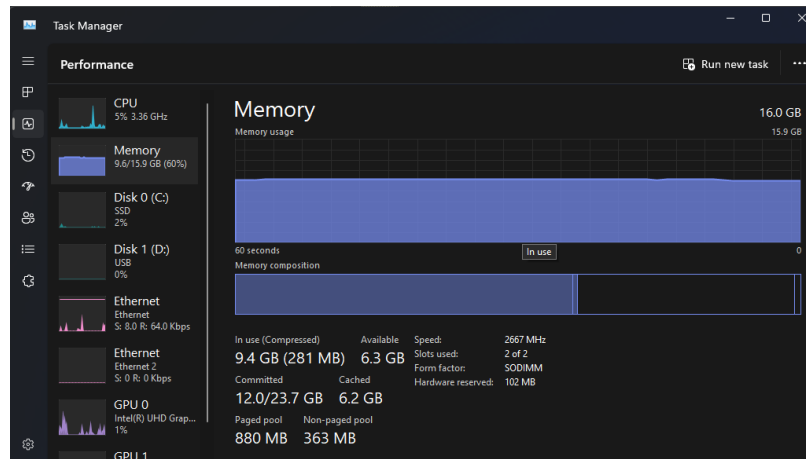
Figure 20: Task Manager screenshot.

Loading and visualizing the large-scale point cloud data significantly consumes memory, as shown in **Figure 20**, a screenshot displaying 60% usage of the available 16 GB RAM. This high memory usage stems from the need to store vast amounts of data and its labels in RAM for real-time processing and visualization. The substantial memory footprint results from the data's volume and complexity, as well as the memory overhead from the data structures employed.

4.4.3 Point Cloud Visualization in Open3d
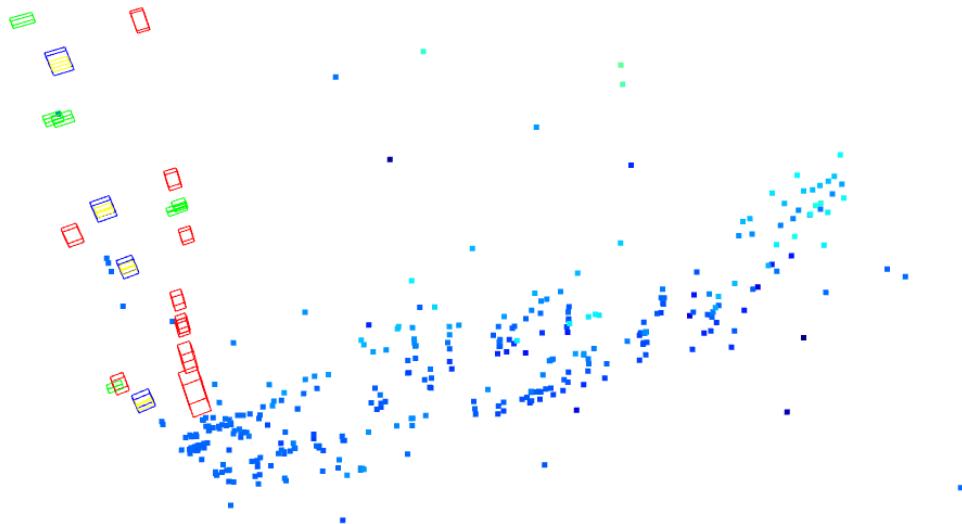
Figure 21: Visualization of radar point cloud for Frame 01047

Illustrated by **Figure 21** the label information is being interpreted incorrectly and used to create the bounding boxes in the 3D visualization.
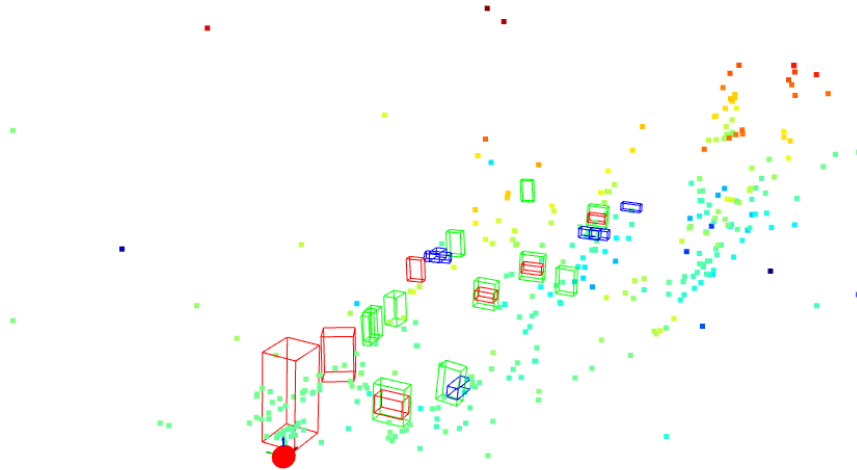
Figure 22: Visualization of radar point cloud for Frame 01047

Illustrated by **Figure 22** is the somewhat accurate label representation using the VOD example_set. Although this is a visual representation of the RADAR point cloud the label annotation comes from lidar.

The visualization code was expanded to utilize OpenCV to display the point cloud side by side with the images captured from the camera mounted on the car. **Figure 23** illustrates the output when the code is executed. It allows the user to press the "space key" to move through the frames, this helped with interpreting the different scenes within the dataset, despite point cloud appearing in the wrong orientation.
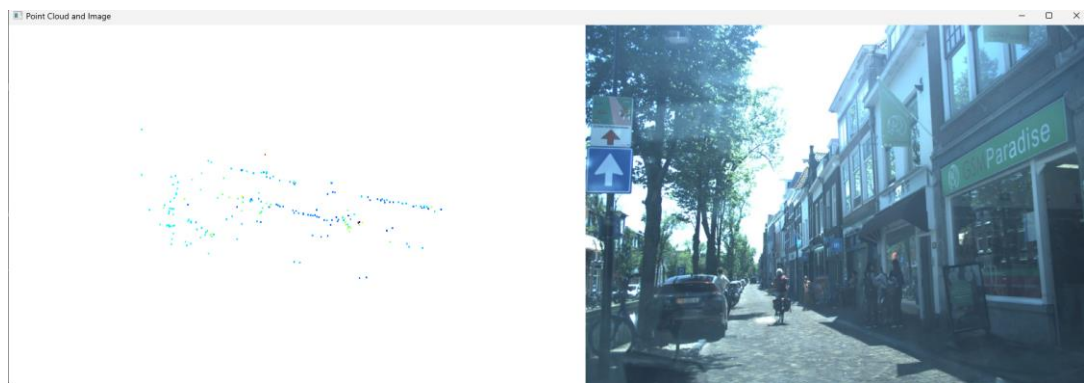
Figure 23: all_frames_and_images.py output.

```
1. class RadarPointCloudVisualizer:
2.     def __init__(self, data, labels, image_path):
3.         self.data = data
4.         self.labels = labels
5.         self.image_path = image_path
6.         self.index = 0
7.
8.     def load_point_cloud_image(self, index, zoom=0.5):
9.         # Convert point cloud to image using Open3D
10.        points = self.data[index][:, :3]
11.        point_cloud = o3d.geometry.PointCloud()
12.        point_cloud.points = o3d.utility.Vector3dVector(points)
13.
14.        vis = o3d.visualization.Visualizer()
15.        vis.create_window(visible=False)
16.        vis.add_geometry(point_cloud)
17.
18.        # Set the zoom level of the view control.
19.        ctr = vis.get_view_control()
20.        ctr.set_zoom(zoom)
21.
22.        vis.poll_events()
23.        vis.update_renderer()
24.        image = vis.capture_screen_float_buffer(False)
25.        vis.destroy_window()
26.
27.        image = np.asarray(image)
28.        image = (image * 255).astype(np.uint8)
29.        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)  # Convert from RGB to BGR
format
30.        return image
```

Figure 24: all_frames_and_images.py code snippet.

After a comparison was made of the labels coming from the .txt files and comparing them to the labels in the. JSON format the author noticed differences in the coordinate system, or the units of measurement used between the two files. JSON objects have explicit centre and quaternion fields for position and rotation, which do not match how the .txt file represents these values. After doing some more digging it became apparent that certain transformations are required to get the labels which were done on the lidar point clouds mapped correctly to the RADAR point clouds.

4.4.4   Sensor Calibration

The VOD Dataset contains a "calib" directory for RADAR and LiDAR which contain .txt files corresponding to each frame. At the time of making the visualisation scripts, the author was not aware of the importance of these calibration files, however after more research and analysis was done. More was now understood about these files. The code block down below shows the structure for **radar/calib/00000.txt** as an example.

```
1. P0: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
2. P1: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
3. P2: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
4. P3: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
5. R0_rect: 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0
6. Tr_velo_to_cam: -0.013857 -0.9997468 0.01772762 0.05283124 0.10934269 -0.01913807 -
0.99381983 0.98100483 0.99390751 -0.01183297 0.1095802 1.44445002
7. Tr_imu_to_velo:
```

P0, P1, P2, P3: These are the projection matrices for different cameras or sensors. Each matrix Pn has 12 elements arranged as a 3x4 matrix, which transforms 3D coordinates into 2D coordinates in the image plane, along with a scaling factor.

```
1. P0: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
2. P1: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
3. P2: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
4. P3: 1495.468642 0.0 961.272442 0.0 0.0 1495.468642 624.89592 0.0 0.0 0.0 1.0 0.0
5. R0_rect: 1.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0
6. Tr_velo_to_cam: -0.007980200000000000 -0.999854100000000000 0.015104900000000000
0.151000000000000000 0.118497000000000000 -0.015944500000000000 -0.992826400000000000 -
0.461000000000000000 0.992922400000000000 -0.006133100000000000 0.118606900000000000 -
0.915000000000000000
7. Tr_imu_to_velo:
```

All the matrices P0 through P3 here are identical.

R0_rect: This is the rectifying rotation matrix, used to align the coordinate systems of different sensors (e.g., from raw sensor coordinates to aligned coordinates). It's typically a 3x3 identity matrix.

Transformation Matrices (Tr_velo_to_cam): here, differences emerge, reflecting how each sensor is physically positioned and oriented relative to the camera:

These matrices are crucial as they transform coordinates from the lidar/velodyne (velo) or radar/velodyne system to the camera coordinate system. Each matrix includes rotations and translations which adapt the sensor data to be visually accurate when overlaid onto images from the camera. The differences in values reflect the unique physical setups and orientations of the RADAR versus the lidar relative to the camera.

4.5 RaTrack

As mentioned in section [3.11], a choice was made to abandon trying to integrate the OpenPCDet PV-RCNN architecture and the focus of the project shifted to trying to get RaTrack's implementation to work.

4.5.1 RaTrack Code Standard Review

Upon analysing the RaTrack source code the following conclusions were established.

Currently, the project lacks a style guide, contribution guidelines, or any code standards or linting. When utilizing analysis tools like Sonar, several problems become immediately apparent. For instance, illustrated by **Figure 25**, there is an unnecessarily complex method, requiring four parameters when only three are essential. Additionally, resources are allocated to create a **val_loader** object, which ultimately remains unused.
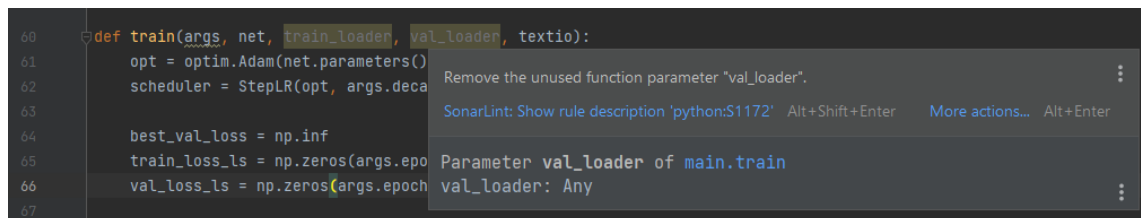
Figure 25: RaTrack/main.py

Illustrated by **Figure 26**, placing an import statement, such as a Python import directive, at line 279 or deep within the code can lead to additional disk access, which can slow down the execution of the program. This occurs because, typically, import statements are expected to be at the beginning of a script or module. When imports are declared upfront, Python loads the required modules into memory at the start, making their functions and classes available throughout the execution without further need for disk access.
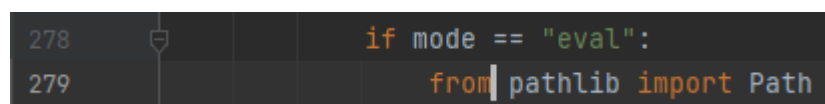
Figure 26: RaTrack/main.py

As depicted in **Figure 27**, a screenshot taken from the file main.py, the tool Sonar Lint is used to highlight the numerous warnings scattered throughout the code. This visual representation serves to emphasize the potential issues and areas for improvement, offering a clear and direct insight into the code's quality.
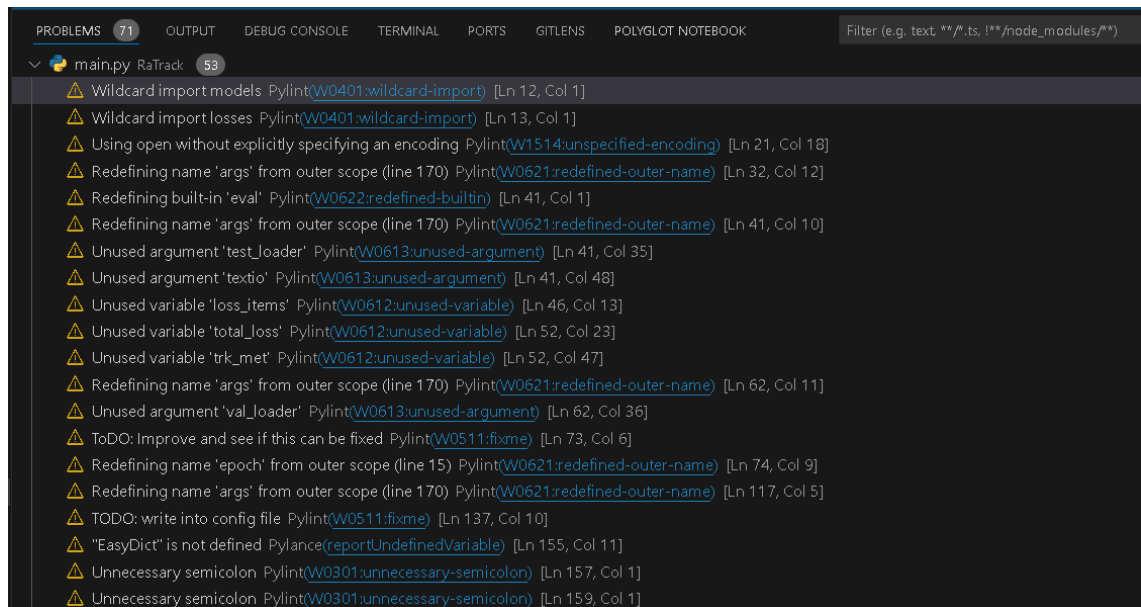
Figure 27: RaTrack/main.py warnings.

The screenshots provided in this section merely touch on the instances of poor practice and substandard code quality.

4.5.2   RaTrack Contributions

Before making any modifications to the source code, the author took proactive steps to improve the installation process and user experience. Recognizing the benefits of a more streamlined setup, the author decided to compile the source code of various dependant libraries into Python wheels. This approach was motivated by the realization that providing pre-built wheels could drastically simplify the setup process for users by removing the need for manual compilation and making dependency management more straightforward. By undertaking this compilation voluntarily, the author not only sought to benefit the broader community but also to foster a collaborative environment that prioritizes code maintainability.

4.5.2.1 Getting required repositories

One design flaw in the RaTrack project stems from its practice of duplicating code, which has led to significant complications in version control, as well as in tracking and crediting authors. Initially, each original repository was forked. For instance, the AB3DMOT repository was made as a subfolder inside of RaTrack. However, AB3DMOT depends on another repository Xinshuo_PyToolbox both by the same GitHub user xinshuoweng.

After the forking of RaTrack, the 'xinshuo_py_toolbox' folder was removed. Subsequently, the 'xinshuo_py_toolbox' repository was integrated as a Git submodule to streamline updates and maintain a clear lineage of code changes.

To improve the maintainability of the code, the compiled software is encapsulated into wheel packages and placed in the 'libs' folder. After consulting with the original developers, there is an intention to release these wheels on PyPi. Once they are available on PyPi, users will be able to easily install them using pip, which will streamline both the distribution and installation processes.

4.5.2.2 Building the Wheels

In building Xinshuo Python Toolbox, the following steps were carried out:

1. Fix minor compilation bugs in the source code.
2. Resolve library dependency issues.
3. A setup.py file was created. The setup.py file is a configuration script for setup tools that defines package details, dependencies, and build instructions, enabling you to build and distribute Python packages, such as wheels.
4. pip reqs were used to build a new requirements.txt file containing all required libraries for the source code.
5. Updating the .gitingore file.

```
1. Successfully built xinshuo_py_toolbox-1.tar.gz and xinshuo_py_toolbox-1.0.0-py3-none-
any.whl
```

Building AB3DMOT source code into a Python wheel, steps 1-5 were repeated.

```
1. Successfully built ab3dmot-1.tar.gz and ab3dmot-1.0.0-py3-none-any.whl
2. PS D:\four-d-radar-thesis\AB3DMOT>
```

The RaTrack library incorporates a specialized C++/Python package known as pointnet2. This package was explicitly designed to harness the computational power of Nvidia GPU cards, optimizing performance for tasks that require significant processing power. Therefore, pointnet2 does not come as a pre-compiled, readily distributable package; it must be compiled and installed manually by the user. This requirement inherently restricts the types of computers that can operate the library, as only those with compatible hardware can compile the software.

Additionally, users must download the appropriate CUDA libraries that are compatible with their operating system and the specific version required by RaTrack. The use of pointnet2 within the RaTrack framework extends to various operations, including running various calculations, model training and evaluation scripts via PyTorch libraries that are part of pointnet2.

To function correctly within RaTrack, CUDA version 11.8 is specifically required, otherwise, you will get a similar error as shown below:

```
1. PS D:\four-d-radar-thesis> python main.py --config configs_eval.yaml
C:\Users\nick2\AppData\Local\Programs\Python\Python311\python.exe: can't open file
'D:\\four-d-radar-thesis\\main.py': [Errno 2] No such file or directory File "D:\four-d-
radar-thesis\RaTrack\main.py", line 169, in <module> main(args.config) File "D:\four-d-
radar-thesis\RaTrack\main.py", line 146, in main eval(args, net, train_loader,
test_loader, textio) File "D:\four-d-radar-thesis\RaTrack\main.py", line 41, in eval
net.eval() ^^^^^^^^ AttributeError: 'NoneType' object has no attribute 'eval' PS D:\four-
d-radar-thesis\RaTrack>
```

cuda_11.8.0_522.06_windows were installed to solve this error, then the following command was executed in RaTrack/lib:

```
1. python setup.py install
```

Now the RaTrack library and its submodules are available as libraries to plug in and use for development, note this process took a considerable amount of time.

4.5.2.3   Source Code Contributions

SonarLint, an IDE extension that provides on-the-fly feedback to developers about bugs, vulnerabilities, and code quality issues, was utilized to enhance the RaTrack source code. By integrating SonarLint into the development environment, the author was able to identify and resolve numerous code inefficiencies and potential errors promptly, significantly improving the readability and maintainability of the code.

Outlined in section [4.5.1] unused parameters were removed as well as in the original **eval** method contained **test_loader** and **textio** parameters that were never used and hence were removed to free up unnecessary resource allocation.

```python
1. def eval(args, net, train_loader, test_loader, textio): # Original Code
1. def eval_model(args, net, train_loader): # Refactored Code
```

The **save_eval_results** and **save_epoch_training_results** methods was developed to record the segmentation and scene flow results during model training and evaluation, illustrated by Figure 30.

```python
1. def save_epoch_training_results(
2.     epoch_number: int, seg_met: dict, flow_met: dict
3. ) -> None:
4.     # Setting Epoch Number
5.     seg_met["epoch"] = epoch_number
6.     flow_met["epoch"] = epoch_number
7.
8.     # Setting Timestamp
9.     timestamp = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
10.    seg_met["timestamp"] = timestamp
11.    flow_met["timestamp"] = timestamp
12.
13.    # Setting Software Version
14.    seg_met["sw-version"] = __version__
15.    flow_met["sw-version"] = __version__
16.
17.    # Define the order of columns explicitly for segmentation metrics
18.    seg_fieldnames = ["epoch", "acc", "miou", "sen", "timestamp", "sw-version"]
19.
20.    # Define the order of columns explicitly for flow metrics
21.    flow_fieldnames = [
22.        "epoch",
23.        "rne",
24.        "50-50 rne",
25.        "mov_rne",
26.        "stat_rne",
27.        "sas",
28.        "ras",
29.        "epe",
30.        "timestamp",
31.        "sw-version",
32.    ]
33. #...
```

save_epoch_training_results/mine.py

The results would then be stored in CSV files, and then called by **save_json_list_to_csv** refer to **Figure 28**. This addition was crucial because the results get displayed in the terminal after the model has been trained or evaluated and then lost. Storing these results proves extremely valuable, particularly as discussed in section [5.4], for various visualisations.

```
1. def save_json_list_to_csv(
2.     json_list: list[dict], filename: str, mode: str = "a", fieldnames: list = None
3. ) -> None:
4.     """Save data to a CSV file.
5.
6.     Args:
7.         json_list (list): List of dictionaries containing data to be saved.
8.         filename (str): Name of the CSV file to save.
9.         mode (str): Mode to open the CSV file. Default is 'a' (append).
10.        fieldnames (list): Explicit list of field names for the CSV file.
11.    """
12.    if not json_list:
13.        logging.warning("No data to save.")
14.        return
15.
16.    os.makedirs(os.path.dirname(filename), exist_ok=True)
17.
18.    # Use the provided fieldnames or deduce them from the JSON list.
19.    if fieldnames is None:
20.        fieldnames = list(set().union(*(json_dict.keys() for json_dict in json_list)))
21.
22.    if not os.path.exists(filename):
23.        mode = "w"
24.
25.    with open(filename, mode=mode, newline="", encoding="utf-8") as file:
26.        dict_writer = DictWriter(file, fieldnames=fieldnames)
27.        if mode == "w":
28.            dict_writer.writeheader()
29.        dict_writer.writerows(json_list)
30.
```

Figure 28: RaTrack/mine.py

Important note before running the evaluation script, inside of the following method in RaTrack/main_utils.py:

```
1. def epoch(
2.     args, net, train_loader, ep_num=None, opt=None, mode="train", display_images=False
3. ):
```

The method has been modified to include a Boolean flag display_images, if set to True it will also run the real time visualisation and can slow down the evaluation a lot, if you do not wish to run the visualisation as the model is being evaluated set the flag to False.

The evaluation script located inside of RaTrack is run with the following command:

```
1. python main.py --config .\configs_eval-sw-test.yaml
```

This will evaluate the pretrained model specified inside the .yaml file and generate the following:

1. Inside of RaTrack/artifacts/eval: eval-flow-metrics.csv and eval-segmentation-metrics.csv are used to write different metrics for segmentation and scene flow evaluation. Each time the evaluation script is run it will add a new row to the according CSV file with the results, note these results will also be printed to the terminal upon completion of a model evaluation.

2. Inside of RaTrack/artifacts a new folder with a timestamped name for example "2024-04-21--11-10" is created in the format Y-m-d-H-M-S. Inside are 2 folders results and results_vis. Results contains a delft_"number" folder which contain TXT files, each TXT file contains the raw points of the predictions for each corresponding original raw radar point cloud data .bin file located in data/view_of_delft_PUBLIC/radar/training/velodyne, The results_vis contains PNG images of the radar point cloud with the prediction results illustrated by **Figure 29**, numbered to their corresponding original raw radar point cloud data .bin file located in data/view_of_delft_PUBLIC/radar/training/velodyne, these are used for the real time visualization of object detection and tracking as the model is being evaluated.

3. A terminal output with training loss information along with the segmentation and scene flow results, refer to **Figure 30**.
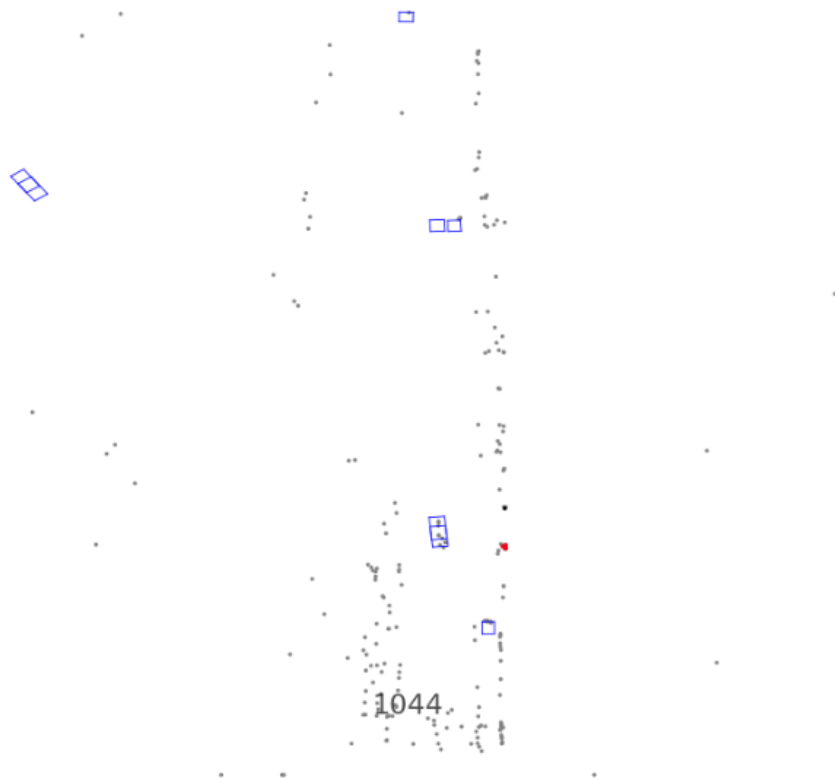
Figure 29: Output preview of "seq445.png"

Figure 30: Terminal output for model evaluation.

Inside of RaTrack/mine.py various methods were created for visualising the object detections in real time during model evaluation. This Python code defines a function display_point_with_image_frame that loads and displays a combined view of a point cloud image and another image captured from camera. Then, it combines this image side-by-side with the camera image using the combine_images function, illustrated in **Figure 34** which also includes a customizable border between them.
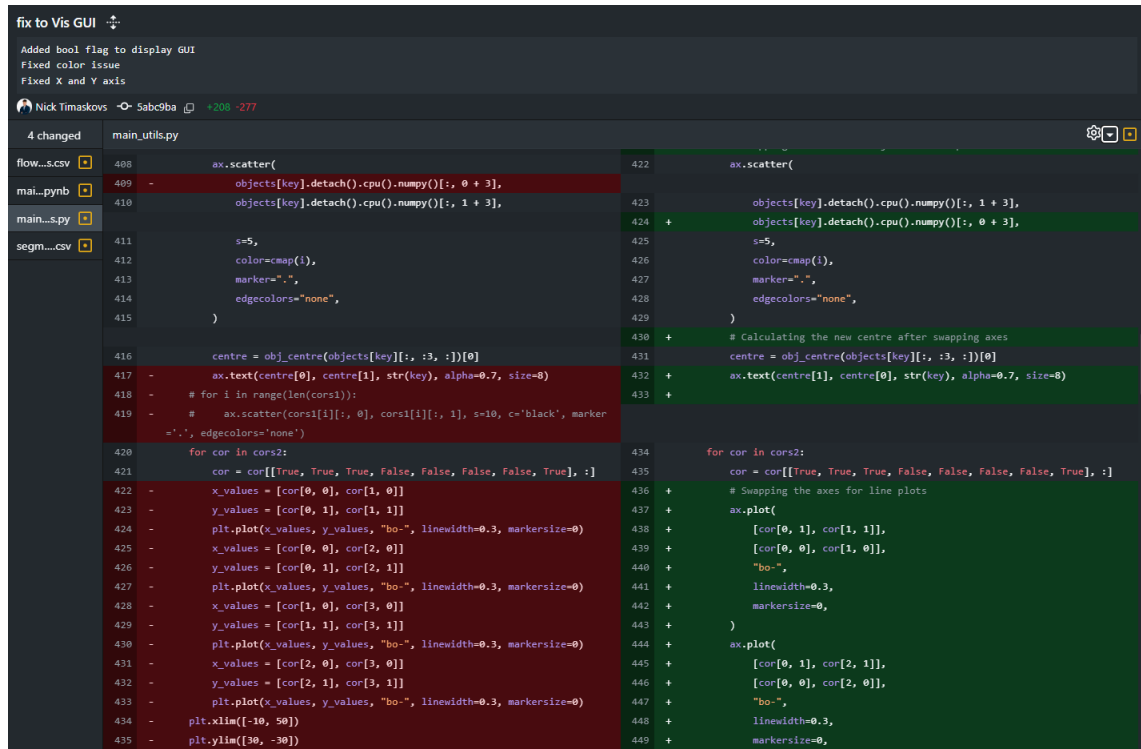
```python
1. def combine_images(
2.     img1, img2, border_color=(255, 255, 255), border_width=10, max_width=1920
3. ):
4.     # Check if images have 4 channels and convert to 3 if necessary
5.     if img1.shape[2] == 4:
6.         img1 = img1[:, :, :3]
7.     if img2.shape[2] == 4:
8.         img2 = img2[:, :, :3]
9.
10.    # Resize images to match in height and adjust width to not exceed max_width
11.    h1, w1 = img1.shape[:2]
12.    h2, w2 = img2.shape[:2]
13.
14.    # Calculate scale to fit both images and border within max_width
15.    scale_factor = min((max_width - border_width) / (w1 + w2), 1)
16.
17.    # Resize both images with the same scale factor to maintain aspect ratio
18.    img1 = cv2.resize(img1, (int(w1 * scale_factor), int(h1 * scale_factor)))
19.    img2 = cv2.resize(img2, (int(w2 * scale_factor), int(h2 * scale_factor)))
20.
21.    # Adjust height of the images to match by adding a black border to the smaller
image
22.    if img1.shape[0] > img2.shape[0]:
23.        delta_h = img1.shape[0] - img2.shape[0]
24.        top, bottom = delta_h // 2, delta_h - (delta_h // 2)
25.        img2 = cv2.copyMakeBorder(
26.            img2, top, bottom, 0, 0, cv2.BORDER_CONSTANT, value=[0, 0, 0]
27.        )
28.    elif img2.shape[0] > img1.shape[0]:
29.        delta_h = img2.shape[0] - img1.shape[0]
30.        top, bottom = delta_h // 2, delta_h - (delta_h // 2)
31. #...
```

Figure 31: combine_images method in RaTrack/mine.py.

The helper function combine_images is used to align and combine two images. It ensures both images are resized to match in height and their total width does not exceed the specified maximum. The author recognized the necessity to swap the x and y axes to ensure that the orientation of the point cloud image aligns correctly with that of the camera image, refer to **figure 35**. This adjustment was essential for coherent data presentation and analysis. Additionally, variable inclinations were modified to improve the overall integration and functionality of the imaging systems.

In the source code, the authors of RaTrack included comments indicating that they had experimented with similar adjustments. This highlights the common challenge faced by developers working with such complex data.

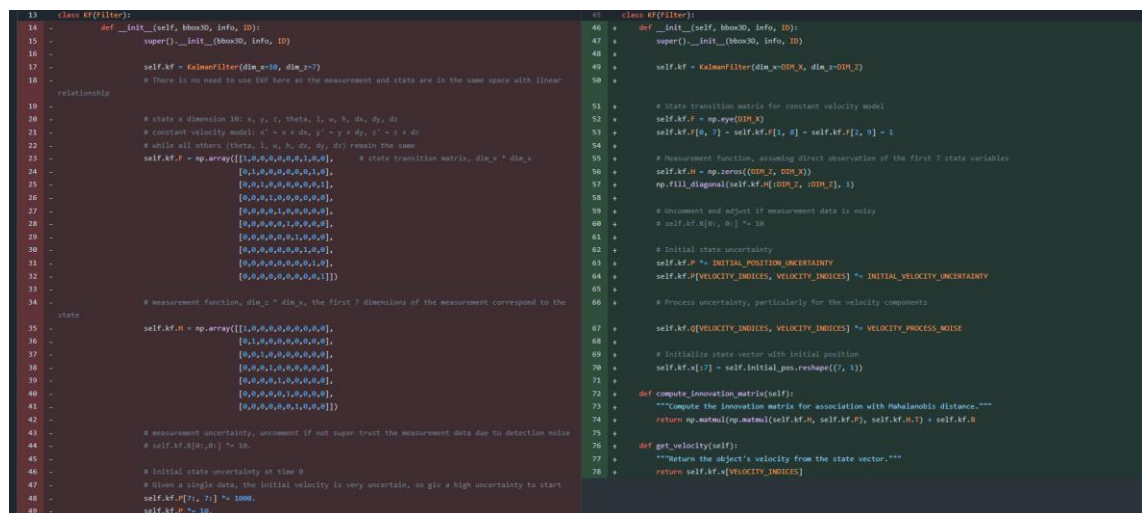**Figure 32:** Fixing orientation of point cloud in RaTrack/main_utils.py

For a detailed view of the code change please refer to the following commit link:

[https://github.com/nicktmv/RaTrack/commit/5abc9ba6ba04cd04a0b76ee42772e416f7a985ad?diff=split&w=1#diff-f65709569da8f501b45cd2327556f93219a5fe7a43afe407fc131f12bc854e32]

AB3DMOT_libs/kalman_filter.py was revised and several enhancements for better clarity, maintainability, and configuration were made, refer to **Figure 29**. Constants are now used to clearly define dimensions and uncertainties, improving the readability and ease of adjustment. The simplification of the state transition and measurement matrices using Numpy functions makes the model's behaviour more intuitive. Documentation has been expanded for better understanding and collaboration. The refactoring ensures the code is modular and scalable, making future updates or model extensions simpler to integrate. Overall, these changes make the code not only easier to understand but also more robust and ready for further development.

For a detailed insight on the changes made, please refer to commit linked here:

[https://github.com/nicktmv/AB3DMOT/compare/28288d48fc13ca688d50f5add39de40e3e240143...87e1857b09c2e3914346ae29a1a14d23c3913742#diff-07770ffd9d85f50f27349fe68ace0d623f9e74789d1b7a8d20a04eb5c2c52f0f]

Figure 33: AB3DMOT_libs/kalman_filter.py Before & After

4.5.3 Miscellaneous Work

A simple script was created inside of a Jupyter notebook named **results_gif.ipynb** illustrated in **Figure 30**

Located at: four-d-radar-thesis\four_d_radar\results_gif.ipynb.

The script merges a sequence of all the PNG files that were generated following the execution of an evaluation script, which depict the outcomes of 4D RADAR object detection and tracking. For reference of the output PNG images refer to **Figure 23**.

```python
 1. os.makedirs(output_folder, exist_ok=True)
 2.
 3. # Define the output file name
 4. output_filename = '4d-radar-track-predictions.gif'
 5. output_path = os.path.join(output_folder, output_filename)
 6.
 7. # Check if the file already exists
 8. if os.path.exists(output_path):
 9.     print(f"GIF file '{output_filename}' already exists. No new file created.")
10. else:
11.     # Retrieve all image files in the directory
12.     image_files = [os.path.join(image_folder, f) for f in os.listdir(image_folder) if
f.endswith('.png')]
13.     # Sort the files by extracting numbers from filenames and converting them to
integers
14.     image_files.sort(key=lambda x: int(re.search(r'\d+',
os.path.basename(x)).group()))
15.
16.     # Create a list to hold the images
17.     images = []
18.
19.     # Open each file, convert to the same mode and append to the list
20.     for file in image_files:
21.         img = Image.open(file)
22.         images.append(img.convert('P', palette=Image.ADAPTIVE))
23.
24.     # Save the images as a GIF
25.     images[0].save(output_path, save_all=True, append_images=images[1:],
optimize=False, duration=200, loop=0)
26.     print(f"GIF created successfully at '{output_path}'!")
27.
```

Figure 34: code snippet for results_gif.ipynb

The created GIF is placed inside "four-d-radar-thesis\docs\images"

4.5.4 Training the model.

According to the research carried out by (Pan et al., n.d.), *"The training keeps for 16 epochs with an initial learning rate of 0.001. This allows for fast learning of accurate moving object detection before data association. We then train the whole network end-to-end for an additional 8 epochs with an initial learning rate of 0.0008."* The authors of RaTrack used a total of 24 epochs to train the model.

It took approximately 35s for 1% of an epoch, given this information we can create the following equation:

$$Time\ per\ epoch = 35\ seconds\ per\ 1\% \times 100\% = 3500\ seconds\ per\ epoch$$

$$Total\ time\ (in\ seconds) = 3500\ seconds\ per\ epoch \times 24\ epoch = 84000\ seconds$$

$$Total\ time\ (in\ hours) = \frac{8400\ seconds}{3600\ seconds\ in\ an\ hour} \approx 23\ hours$$

Considering the capabilities of the available machine processor and the project deadlines, it was decided to initially train the model for 8 epochs. Subsequently, the model would then be trained for 10 and 24 epochs, allowing for a comparative analysis between the training durations.

The author trained the model by running the following command:

```
1. cd Ratrack
2. python main.py
```

The model training parameters are defined inside of "configs.yaml".

```
 1. exp_name: track4d_radar
 2. model: track4d_radar
 3.
 4. ## training
 5. num_points: 256
 6. batch_size: 1
 7. val_batch_size: 1
 8. epochs: 8
 9. lr: 0.001
10. decay_epochs: 1
11. decay_rate: 0.97
12. pretrain_epochs: 16
13.
14. ## device
15. cuda_device: '0'
16. no_cuda: False
17. seed: 1234
18. num_workers: 4
19.
```

Figure 35: Code snippet of config.yaml

The following terminal output was displayed after 8 epochs where run:

```
1. ==epoch: 7, learning rate: 0.000808==
2.
100%|███████████████████████████████████████████████████
████████████████████████| 5139/5139 [39:12<00:00,  2.18it/s, Loss=0.162,
SceneFlowLoss=0.503, TrackingLoss=0.755, SegLoss=0.162]
3. segmentation:  {'acc': 0.9350254951044601, 'miou': 0.6210612418488741, 'sen':
0.8979069273541074}
4. scene flow:  {'rne': 0.2756754927497989, '50-50 rne': nan, 'mov_rne': 0.0, 'stat_rne':
nan, 'sas': 0.9884141889721247, 'ras': 0.9992770739035745, 'epe': 0.6500250041047136}
5. Loss: 0.153130
6. SceneFlowLoss: 0.650025
7. SegLoss: 0.153130
8. TrackingLoss: 0.604181
9. mean train loss: 0.153130
10. best val loss till now: 0.153130
11. FINISH
```

Figure 36: model trained on 8 epochs output.

The model was then trained on 24 epochs resulting in the following output:

```
1.100%|████████████████████████     5139/5139 [49:45<00:00,  1.72it/s, Loss=0.155,
SceneFlowLoss=0.0807, TrackingLoss=0.00234, SegLoss=0.114]
2. segmentation:  {'acc': 0.9248832600880852, 'miou': 0.6067429993465803, 'sen':
0.8802882110384151}
3. scene flow:  {'rne': 0.09319050577322738, '50-50 rne': nan, 'mov_rne': 0.0,
'stat_rne': nan, 'sas': 0.9995545196051737, 'ras': 0.9999953641432782, 'epe':
0.2205050654612007}
4. Loss: 0.332229
5. SceneFlowLoss: 0.220505
6. SegLoss: 0.172420
7. TrackingLoss: 0.099113
8. mean train loss: 0.332229
9. FINISH
```

Figure 37: model trained on 24 epochs output.

The results and the different metrics shown in the above terminal output are explained in greater detail in [Chapter 5].

4.5.5 Conclusion

To conclude the implementation chapter, it's important to acknowledge that significant advancements on the RaTrack system were constrained by the rigorous demands of the development process. Each proposed modification required extensive validation, primarily through model training, to ensure that the system's performance remained stable and reliable. This necessity for thorough testing before confirming any changes means that the pace of development was necessarily cautious. The careful approach was crucial in maintaining the integrity of the system and avoiding discrepancies that could undermine its effectiveness. As such, while the progress in this phase might appear limited, the meticulous attention to detail ensures that any enhancements to the system are both robust and beneficial.

Chapter 5 Testing and Results

5.1 Introduction

Pending….

5.1.1 What is Unit Testing, and why is it important?

Unit testing is a software development technique that involves testing individual components or units of a software application to verify their functionality. In this approach, each unit is tested in isolation, with dependencies typically mocked or stubbed to focus solely on the unit's behaviour. The significance of unit testing lies in its ability to detect errors early in the development cycle, reducing the complexity and cost associated with later-stage bug fixes. By isolating and testing units separately, developers can swiftly identify and resolve issues that might be elusive in broader testing strategies. Furthermore, unit testing plays a crucial role in maintaining code integrity during updates, preventing new modifications from disrupting existing functionality. Integrating unit tests into a continuous integration workflow ensures thorough vetting of changes before merging them into the main code repository. Additionally, unit testing promotes the development of clean, maintainable code by encouraging modular design, which simplifies understanding and maintenance. Ultimately, unit testing is a vital practice in software development, enhancing the quality, stability, and reliability of software products, while fostering best coding practices and preventing regression.

5.1.2 Testing

Unit testing an the existing RaTrack source code would be exceptionally challenging due to the complexity of the code and the advanced technology involved. The intricate interdependencies within the system and the high-level technical knowledge required to understand the nuances of the radar signal processing and neural network implementation add significant hurdles. This complexity often leads to a testing environment that is difficult to set up and maintain. However, had unit testing been integrated from the beginning of the development process, it would have enforced better coding standards, promoted cleaner and more modular code, and facilitated easier bug tracking and fixes. Implementing unit tests early on would not only have made the code more robust and maintainable but also would have simplified later stages of testing by ensuring that individual components function correctly before they are integrated into the larger system.

5.2   Model Training Results Analysis

This chapter discusses the training results of various models, across different configuration setups. The results which are stored in CSV files are referenced throughout this section can be found in four-d-radar-thesis/docs/results. There are also a Jupyter Notebook provided at the following location:

four-d-radar-thesis\four_d_radar\results_visualization.ipynb

These Notebook contain the visualisations of results showcased in this section.

5.2.1   Model Trained on 8 Epochs

The following are graphs plotting total loss, scene flow loss and segmentation loss over an 8 epochs, note run 1 had pretrained epochs set to 2 while run 2 had them set to 16.

Figure **38:** Graph 1

Figure 39: Graph 2

Figure 40: Graphs 3

5.2.2 Model Trained on 10 Epochs

The following are graphs plotting total loss, scene flow loss and segmentation loss over an 8 epochs, note run 1 had pretrained epochs set to 16 while run 2 had them set to 2.
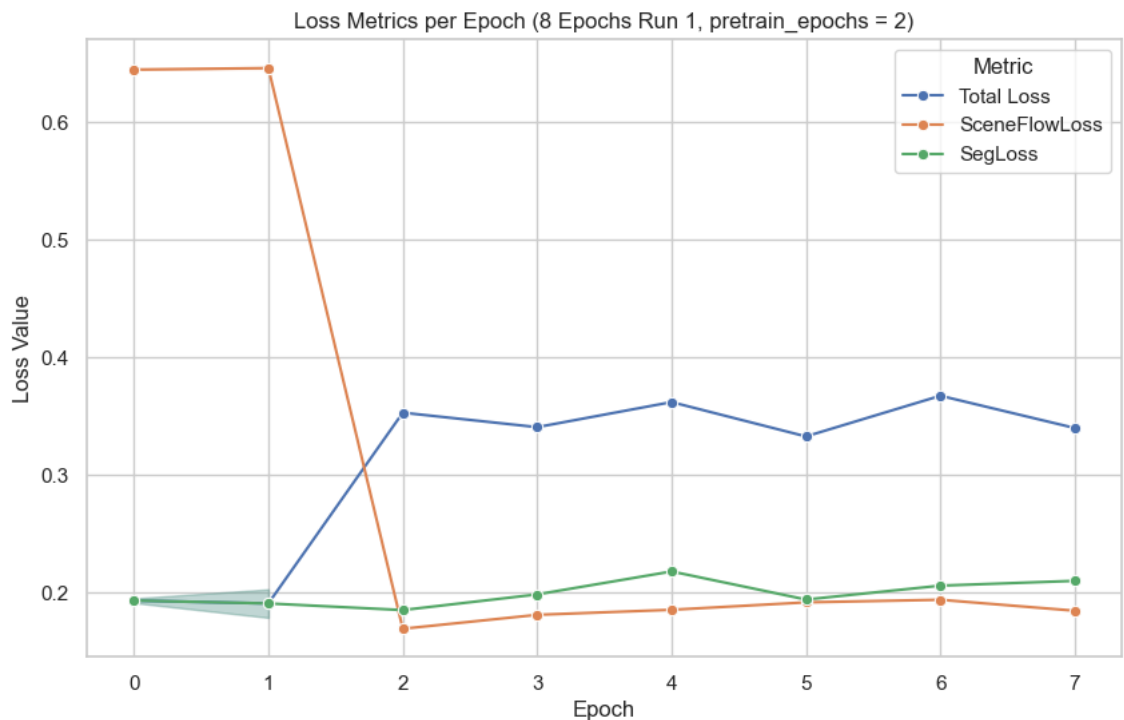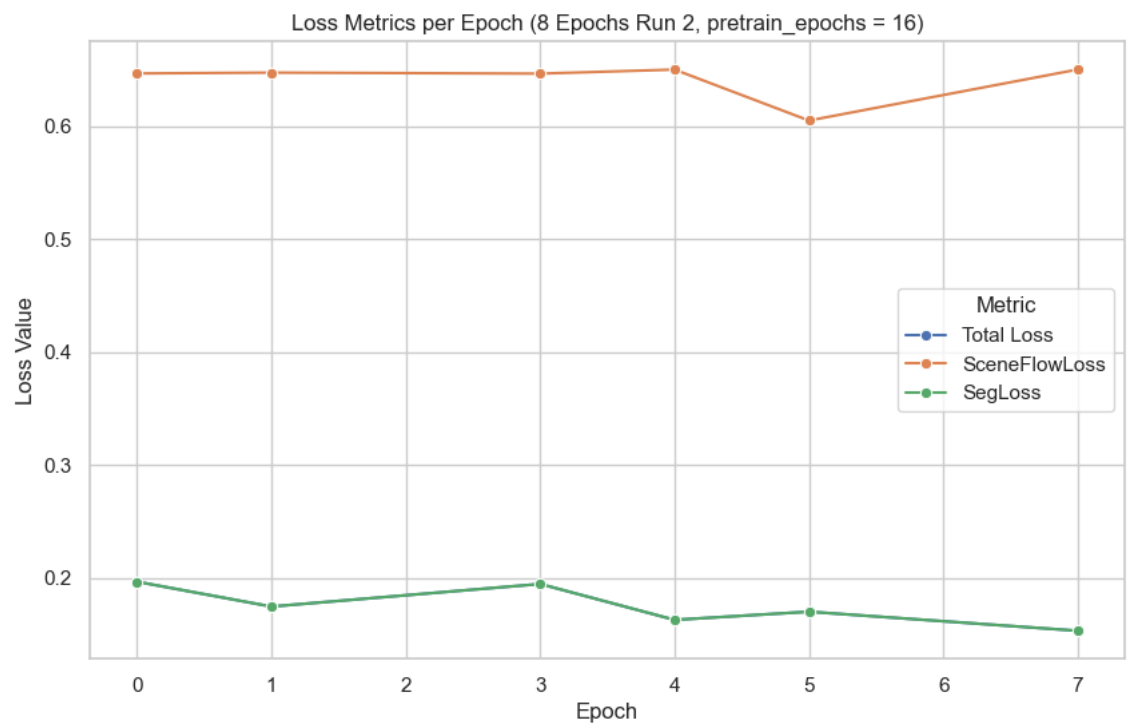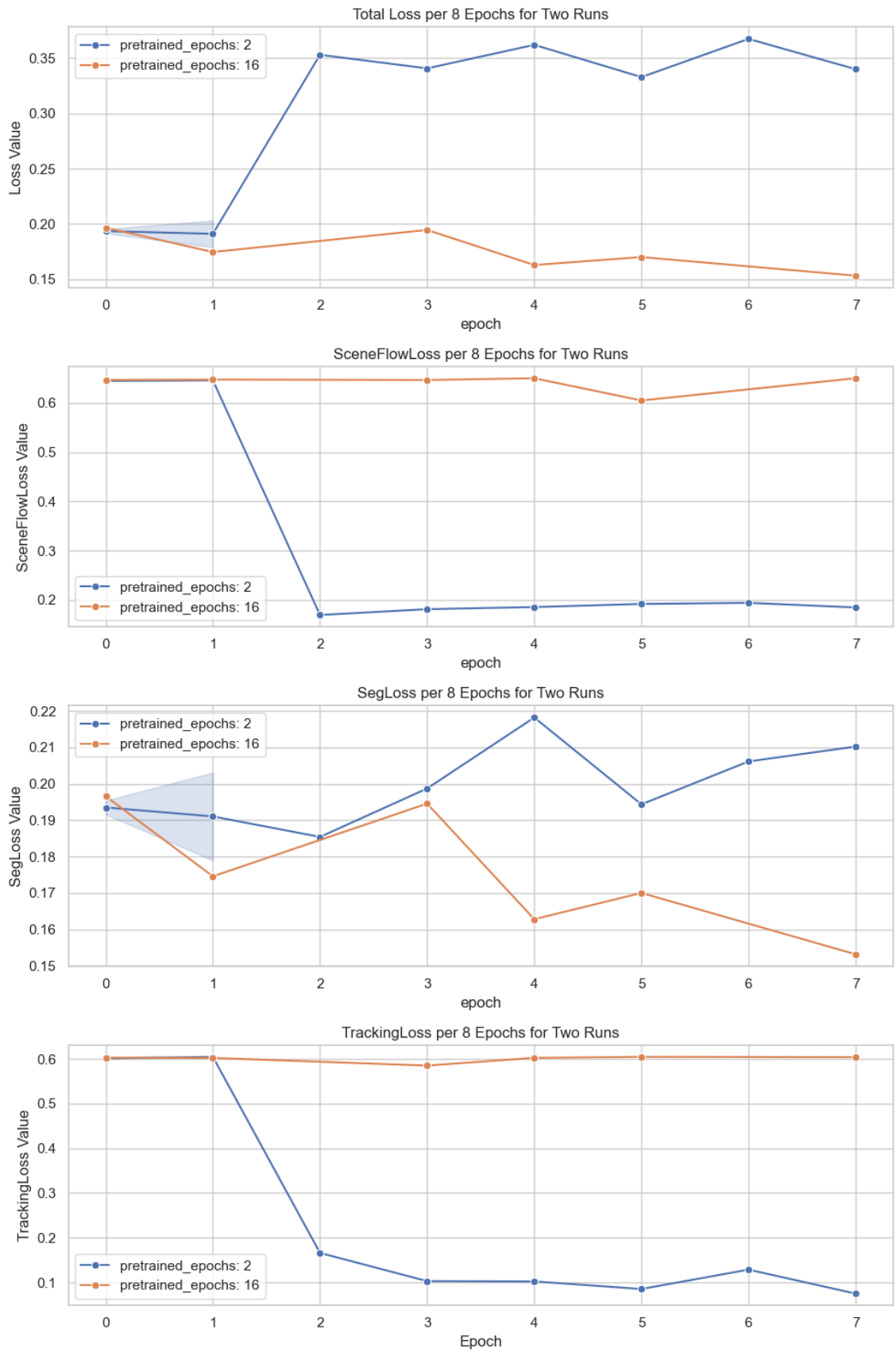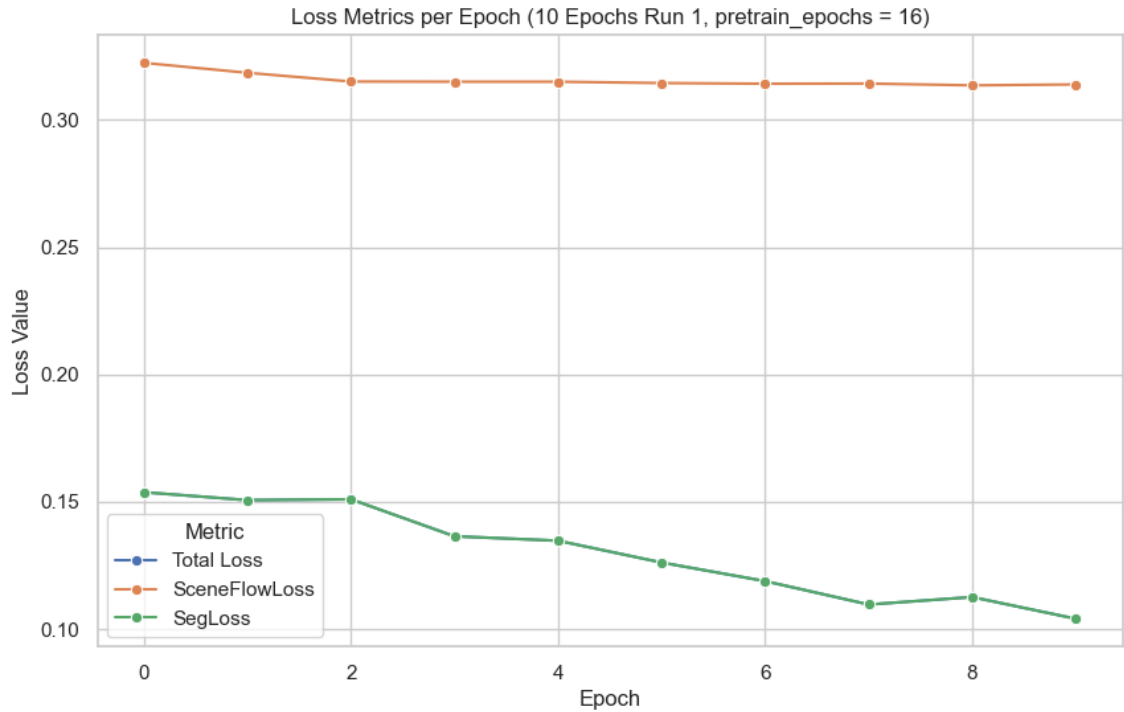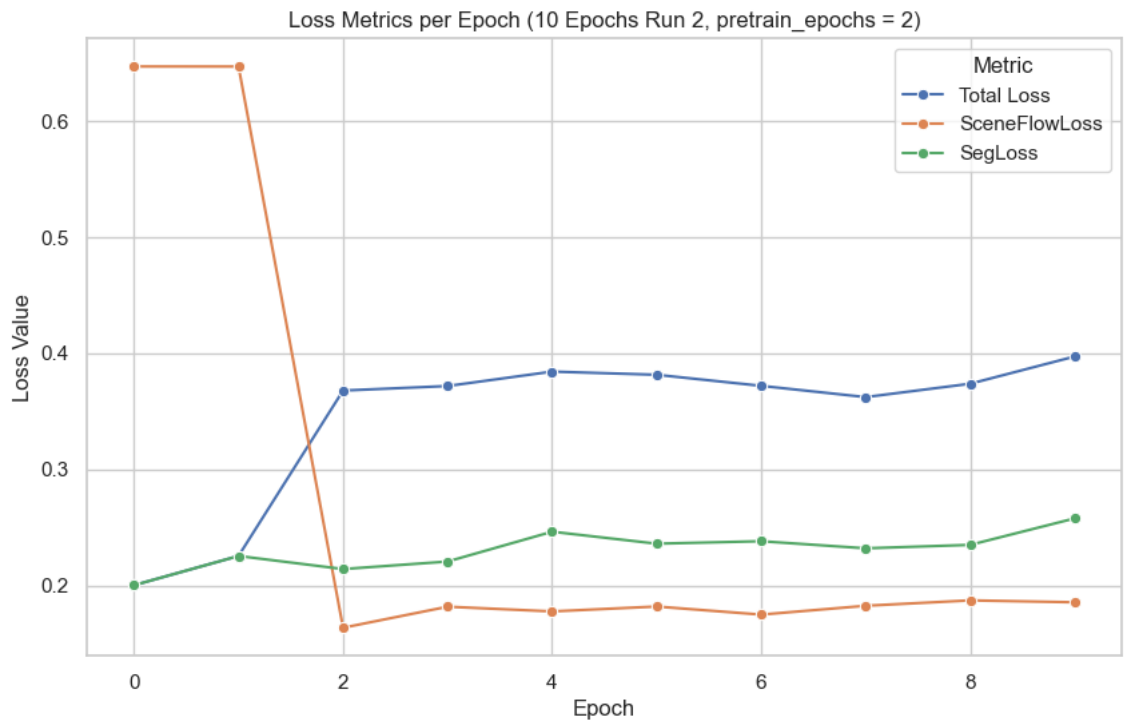

Figure 41: Graph 4

Figure 42: Graph 5

Talk about similar trend emerging why???

The following is a detailed breakdown of what these metrics represent, along with the values obtained from **Figure 22** and what they mean:

acc (Accuracy): The proportion of correct predictions out of the total number of cases examined. The model has an accuracy of approximately 68.53%.

miou (Mean Intersection over Union): A common metric for semantic segmentation tasks that measures the average overlap between the predicted segmentation and the ground truth across all classes. The model has a miou of approximately 37.99%.

sen (Sensitivity): Also known as recall or true positive rate, it measures the proportion of actual positives that are correctly identified. Your model has a sensitivity of approximately 78.69%.

rne (Relative Navigation Error): This appears to be a custom or specific metric for evaluating the performance the 4D radar tracking model created by the researchers of the RaTrack repository. This could be assessing the accuracy of the model in terms of relative positioning or movement estimation. The rne is approximately 14.78%.

50-50 rne: This might be a variant of rne evaluated under certain conditions or thresholds. It's showing as `nan`, which suggests that the condition may not have been met or there was an issue during evaluation.

mov_rne (Moving Relative Navigation Error): This is showing as 0.0, which suggests that for moving objects, the model is performing perfectly, but this is quite unusual and further investigation will be required.

stat_rne (Static Relative Navigation Error): Like the 50-50 rne, this is also `nan`, which could indicate missing or undefined values in the dataset for static objects.

sas (Static Accuracy Score): It's very high, suggesting that the model's predictions for static objects are nearly perfect with a score of approximately 99.72%.

ras (Radar Accuracy Score): Like sas, it's a very high score, showing that the model's overall radar prediction accuracy is nearly perfect at approximately 99.98%.

epe (End-Point Error): Commonly used in optical flow estimation, this measures the average Euclidean distance (error) between the predicted end point and the ground truth. The model has an epe of approximately 0.35.

5.3.1 Segmentation Comparison

This table represents the comparison between run 1 and run 2 when the model was trained on 8 epochs.

Run 1: pretrain_epochs = 2

Run 2: pretrain_epochs = 16

Metric	Run 1	Run 2	Better Run
Accuracy	0.915624	0.929237	Run 2
MIoU	0.588501	0.610672	Run 2
Sensitivity	0.869579	0.884062	Run 2

5.3.2 Scene Flow Comparison

This table represents the comparison between run 1 and run 2 when the model was trained on 8 epochs.

Run 1: pretrain_epochs = 2

Run 2: pretrain_epochs = 16

Metric	Run 1	Run 2	Better Run
RNE	0.126901	0.274913	Run 1
SAS	0.996817	0.988114	Run 1
EPE	0.299801	0.648234	Run 1

5.4 Comparing 10 vs 8 Epochs

This table represents the segmentation and scene flow results comparison between a model trained on 8 epochs vs a model trained on 10 epochs, unfortunately the results for 10 epochs when pretrain_epochs = 16 were corrupted during training.

10 Epochs: pretrain_epochs = 2

8 Epochs: pretrain_epochs = 16

Metric	10 Epoch	8 Epoch	Better Model
Accuracy	0.902215	0.929237	Epoch 8
MIoU	0.569491	0.610672	Epoch 8
Sensitivity	0.849747	0.884062	Epoch 8

Metric	10 Epoch	8 Epoch	Better Model
RNE	0.115484	0.274913	Epoch 10
SAS	0.997415	0.988114	Epoch 10
EPE	0.272903	0.648234	Epoch 10

5.5 Discussion of Findings

In main_utils.py on line 232, This the following flag:

```
1. pretrain = True if ep_num < args.pretrain_epochs else False
```

is passed into a method 4d track_4d_loss inside of loss.py located in RaTrack/losses/.py

```
1.    total_loss = 0.5 * sf_loss + 0.5 * trk_loss + 1 * seg_loss
2.    if pretrain:
3.        total_loss = 0 + 0 + seg_loss
```

To summarise what this snippet of code does, until an epoch number is greater than pretrain which is mapped to pretrain_epochs inside of the config.yaml file.

The total loss includes half of the sf (scene flow) loss and half the trk (tracking) loss.

Only after the epoch number exceeds or equals pretrain_epochs (defined in config.yaml) is total loss calculated as seg loss alone. This explains the differences noticed between the training results observed in 10_epoch_training_results_run2 (where pre trained epochs where set to 1) and 10_epoch_training_results (where pre trained epochs where set to 16). This also explain the sharp dip:
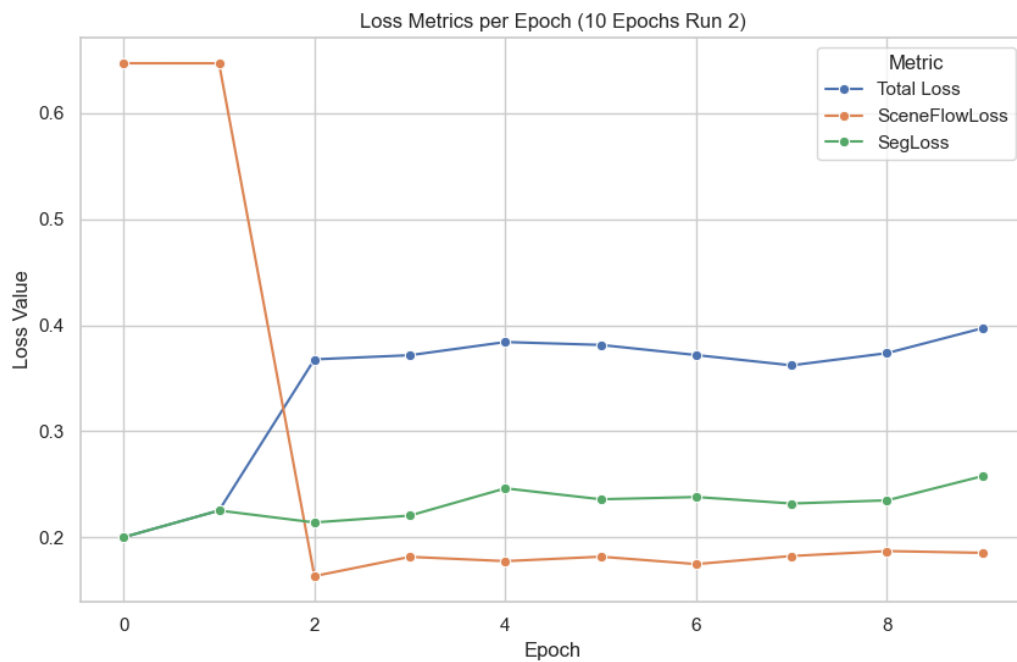
Figure 43: Graph temp.

Chapter 6 Conclusion:

Pending…

6.1 Future Work

The 4D radar object detection and tracking pipeline, could be modified to integrate a classification module to enhance the pipeline's functionality. The current pipeline, which focuses on segmentation and scene flow, could be augmented by a classification module to identify, and categorize detected objects based on their radar signatures. This addition would allow for more sophisticated scene understanding and decision-making capabilities in applications like autonomous driving or surveillance. The classification results could also be fed back into the tracking system to improve the tracking accuracy by using class-specific motion models or size constraints, thereby refining the overall effectiveness of the object detection, and tracking process.

A keen interest is expressed in continuing to contribute to the RaTrack repository and its dependent submodules. The aim would be to further develop and refine the functionalities by implementing the latest research findings and community feedback. This ongoing commitment will focus on enhancing the robustness, accuracy, and efficiency of the modules, ensuring that the repository remains at the forefront of advancements in radar-based tracking technologies. The author is also dedicated to collaborating with other contributors to foster a vibrant community around the RaTrack project, encouraging open-source collaboration and knowledge sharing.

Furthermore, enhancing the RaTrack framework is seen as an opportunity to bridge the gap between the fields of mathematics, artificial intelligence, and software development. This involves implementing better software practices such as Object-Oriented Programming (OOP), adherence to single responsibility principles, improved naming conventions, and the introduction of a style guide. Additionally, the use of linters, creating more structured documentation, and addressing ownership issues aim to reduce language barriers and misunderstandings caused by unclear naming conventions, ultimately facilitating smoother interactions and clearer communication among diverse technical teams.

Chapter 7 References

Zhou, Y., Liu, L., Zhao, H., López-Benítez, M., Yu, L. and Yue, Y. (2022). Towards Deep Radar Perception for Autonomous Driving: Datasets, Methods, and Challenges. *Sensors*, [online] 22(11), pp.4208–4208. doi:https://doi.org/10.3390/s22114208.

2.0 A Vision for Safety AUTOMATED DRIVING SYSTEMS. (n.d.). Available at: https://www.nhtsa.gov/sites/nhtsa.gov/files/documents/13069a-ads2.0_090617_v9a_tag.pdf.

Anwesh Marwade (2020). *Kalman Filtering: An Intuitive Guide Based on Bayesian Approach*. [online] Medium. Available at: https://towardsdatascience.com/kalman-filtering-an-intuitive-guide-based-on-bayesian-approach-49c78b843ac7

AR, A. (2021). *LiDAR vs RADAR: Detection, Tracking, and Imaging - For 4D Sensing, AI, AR, AV... - Medium*. [online] Medium. Available at: https://4sense.medium.com/lidar-vs-radar-detection-tracking-and-imaging-ca528c0e9aae [Accessed 24 Oct. 2023].

AUTOCRYPT. (2023). *The State of Level 3 Autonomous Driving in 2023 | AUTOCRYPT*. [online] Available at: https://autocrypt.io/the-state-of-level-3-autonomous-driving-in-2023/

Alex Becker (www.kalmanfilter.net (2017). *Online Kalman Filter Tutorial*. [online] Kalmanfilter.net. Available at: https://www.kalmanfilter.net/default.aspx#:~:text=The%20Kalman%20Filter%20is%20a%20widely%20used%20estimation%20algorithm%20that,state%20based%20on%20past%20estimations.

Walenta, K., Genser, S. and Selim Solmaz (2024). Bayesian Gaussian Mixture Models for Enhanced Radar Sensor Modeling: A Data-Driven Approach towards Sensor Simulation for ADAS/AD Development. *Sensors*, [online] 24(7), pp.2177–2177. doi:https://doi.org/10.3390/s24072177.

Robotics Knowledgebase. (2019). *Guassian Process and Gaussian Mixture Model*. [online] Available at: https://roboticsknowledgebase.com/wiki/math/gaussian-process-gaussian-mixture-model/

Avijeet Biswal (2020). *Power of Recurrent Neural Networks (RNN): Revolutionizing AI*. [online] Simplilearn.com. Available at: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

baeldung (2022). *The Viola-Jones Algorithm | Baeldung on Computer Science*. [online] Baeldung on Computer Science. Available at: https://www.baeldung.com/cs/viola-jones-algorithm

Balasubramaniam, A. and Pasricha, S. (2022). *Object Detection in Autonomous Vehicles: Status and Open Challenges*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/357953408_Object_Detection_in_Autonomous_Vehicles_Status_and_Open_Challenges#pf2 [Accessed 25 Oct. 2023].

Balasubramaniam, A. and Pasricha, S. (n.d.). *Object Detection in Autonomous Vehicles: Status and Open Challenges*. [online] Available at: https://arxiv.org/ftp/arxiv/papers/2201/2201.07706.pdf#:~:text=Object%20detection%20consists%20of%20two

Barnard, M. (2016). *Tesla & Google Disagree About LIDAR - Which Is Right? - CleanTechnica*. [online] CleanTechnica. Available at: https://cleantechnica.com/2016/07/29/tesla-google-disagree-lidar-right/ [Accessed 10 Oct. 2023].

Bloom, C. (2020). *Introduction to Radar*. [online] Arrow.com. Available at: https://www.arrow.com/en/research-and-events/articles/introduction-to-radar [Accessed 23 Oct. 2023].

Borah, C. (2020). *Evolution of Object Detection - Analytics Vidhya - Medium*. [online] Medium. Available at: https://medium.com/analytics-vidhya/evolution-of-object-detection-582259d2aa9b [Accessed 25 Oct. 2023].

Cohen, J. (2020). *How RADARs work*. [online] Welcome to The Library! Available at: https://www.thinkautonomous.ai/blog/how-radars-work/ [Accessed 4 Oct. 2023].

Cohen, J. (2023). *4D LiDARs vs 4D RADARs — Why the LiDAR vs RADAR comparison is more relevant today than ever*. [online] Welcome to The Library! Available at: https://www.thinkautonomous.ai/blog/fmcw-lidars-vs-imaging-

radars/#:~:text=4D%20RADARs%20work%20using%20MIMO,have%20a%20pretty%
20bad%20resolution. [Accessed 10 Oct. 2023].

Davies, E.R. (2022). The dramatically changing face of computer vision. *Elsevier eBooks*, [online] pp.1–91. doi:https://doi.org/10.1016/b978-0-12-822109-9.00010-2.

Doppler effect | Definition, Example, & Facts | Britannica. (2023). In: *Encyclopædia Britannica*. [online] Available at: https://www.britannica.com/science/Doppler-effect [Accessed 4 Oct. 2023].

Everythingrf.com. (2021). *What are 4D Radars? - everything RF*. [online] Available at: https://www.everythingrf.com/community/what-are-4d-radars [Accessed 10 Oct. 2023].

Gandhi, R. (2018). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. [online] Medium. Available at: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

GeeksforGeeks. (2018). *Introduction to Recurrent Neural Network*. [online] Available at: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ [Accessed 1 Nov. 2023].

Gibiansky, A. (2014). *Convolutional Neural Networks - Andrew Gibiansky*. [online] Gibiansky.com. Available at: https://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/#:~:text=If%20we%20use%20an%20m,)(j%2Bb).

Gillis, A.S., Burns, E. and Brush, K. (2023). *deep learning*. [online] Enterprise AI. Available at: https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network

Girshick, R., Donahue, J., Darrell, T. and Malik, J. (2016). Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1), pp.142–158. doi:https://doi.org/10.1109/tpami.2015.2437384.

GitHub. (2023). *view-of-delft-dataset/figures/example_frame_2.png at main · tudelft-iv/view-of-delft-dataset*. [online] Available at: https://github.com/tudelft-iv/view-of-delft-dataset/blob/main/figures/example_frame_2.png [Accessed 24 Oct. 2023].

Hearst, M.A., Dumais, S.T., Osman, M., Platt, J. and Bernhard Schölkopf (1998). Support vector machines. *IEEE Intelligent Systems & Their Applications*, [online] 13(4), pp.18–28. doi:https://doi.org/10.1109/5254.708428.

Hesai Webmaster (2023). *What You Need to Know About Lidar: The Strengths and Limitations of Camera, Radar, and Lidar. | HESAI*. [online] HESAI. Available at: https://www.hesaitech.com/what-you-need-to-know-about-lidar-the-strengths-and-limitations-of-camera-radar-and-lidar/#:~:text=Compared%20to%20the%20traditional%203D,over%20100%2C000%20points%20per%20frame. [Accessed 24 Oct. 2023].

Jessica Van Brummelen, O'Brien, M., Gruyer, D. and Homayoun Najjaran (2018). Autonomous vehicle perception: The technology of today and tomorrow. *Transportation Research Part C-emerging Technologies*, [online] 89, pp.384–406. doi:https://doi.org/10.1016/j.trc.2018.02.012.

Levity.ai. (2023). *Deep Learning vs. Machine Learning – What's The Difference?* [online] Available at: https://levity.ai/blog/difference-machine-learning-deep-learning

Mathworks.com. (2023). *What Is SLAM (Simultaneous Localization and Mapping) – MATLAB & Simulink*. [online] Available at: https://www.mathworks.com/discovery/slam.html#:~:text=SLAM%20(simultaneous%20localization%20and%20mapping)%20is%20a%20method%20used%20for,to%20map%20out%20unknown%20environments. [Accessed 10 Oct. 2023].

Navtech Radar. (2023). *FMCW Radar*. [online] Available at: https://navtechradar.com/explore/fmcw-radar/#:~:text=Frequency%20Modulated%20Continuous%20Wave%20(FMCW,by%20the%20radar%20and%20compared. [Accessed 10 Oct. 2023].

NHTSA. (2020). *NHTSA*. [online] Available at: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety [Accessed 23 Oct. 2023].

Ouaknine, A. (2022). *Deep learning for radar data exploitation of autonomous vehicle*. [online] arXiv.org. Available at: https://arxiv.org/abs/2203.08038

ResearchGate. (2019). *Figure 1. Schematic diagram of a basic convolutional neural network...* [online] Available at: https://www.researchgate.net/figure/Schematic-

diagram-of-a-basic-convolutional-neural-network-CNN-architecture-
26_fig1_336805909

Synopsys.com. (2023). *What is an Autonomous Car? – How Self-Driving Cars Work | Synopsys*. [online] Available at: https://www.synopsys.com/automotive/what-is-autonomous-car.html#:~:text=Definition,in%20the%20vehicle%20at%20all. [Accessed 10 Oct. 2023].

Tyagi, M. (2021). *HOG (Histogram of Oriented Gradients): An Overview - Towards Data Science*. [online] Medium. Available at: https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f [Accessed 25 Oct. 2023].

Udemy. (2023). *Automotive Radar*. [online] Available at: https://www.udemy.com/course/automotive-radar-basics-to-advance/ [Accessed 10 Oct. 2023].

Xx, N., Xx, X. and Xxxx (n.d.). Millimeter Wave Sensing: A Review of Application Pipelines and Building Blocks. *IEEE SENSORS JOURNAL*, [online] (1). Available at: https://arxiv.org/pdf/2012.13664.pdf.

Yang, B., Guo, R., Liang, M., Casas, S. and Urtasun, R. (n.d.). *RadarNet: Exploiting Radar for Robust Perception of Dynamic Objects*. [online] Available at: https://arxiv.org/pdf/2007.14366.pdf.

Zhou, T., Yang, M., Jiang, K., Wong, H.T. and Yang, D. (2020). MMW Radar-Based Technologies in Autonomous Driving: A Review. *Sensors*, [online] 20(24), pp.7283–7283. doi:https://doi.org/10.3390/s20247283.

Zhou, Y. and Yue, Y. (2022). *Radar Signal Processing Fundamentals*. [online] Encyclopedia.pub. Available at: https://encyclopedia.pub/entry/23781 [Accessed 23 Oct. 2023].

Joel Markus Vaz and Balaji, S. (2021). Convolutional neural networks (CNNs): concepts and applications in pharmacogenomics. *Molecular Diversity*, [online] 25(3), pp.1569–1584. doi:https://doi.org/10.1007/s11030-021-10225-3.

Deng, J., Dong, W., Socher, R., Li, L., Li, K. and Li, F. (2009). ImageNet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [online] doi:https://doi.org/10.1109/cvpr.2009.5206848.

Hironobu Fujiyoshi, Hirakawa, T. and Yamashita, T. (2019). Deep learning-based image recognition for autonomous driving. *IATSS Research*, [online] 43(4), pp.244–252. doi:https://doi.org/10.1016/j.iatssr.2019.11.008.

Balasubramaniam, A. and Pasricha, S. (n.d.). *Object Detection in Autonomous Vehicles: Status and Open Challenges*. [online] Available at: https://arxiv.org/ftp/arxiv/papers/2201/2201.07706.pdf#:~:text=Object%20detection%20consists%20of%20two.

Nitin Kushwaha (2023). *A Brief History of the Evolution of Image Classification*. [online] Medium. Available at: https://python.plainenglish.io/a-brief-history-of-the-evolution-of-image-classification-402c63baf50

Deepchecks. (2021). *What is VGGNet | Deepchecks*. [online] Available at: https://deepchecks.com/glossary/vggnet/#:~:text=Its%20object%20recognition%20method%20developed,dataset%20by%20a%20wide%20margin.

Great Learning Team (2020). *Introduction to Resnet or Residual Network*. [online] Great Learning Blog: Free Resources what Matters to shape your Career! Available at: https://www.mygreatlearning.com/blog/resnet/#:~:text=ResNet%2C%20short%20for%20Residual%20Network,Residual%20Learning%20for%20Image%20Recognition%E2%80%9D.

Paperswithcode.com. (2020). *Papers with Code - DenseNet Explained*. [online] Available at: https://paperswithcode.com/method/densenet#:~:text=A%20DenseNet%20is%20a%20type,sizes)%20directly%20with%20each%20other.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N. (2020). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. [online] arXiv.org. Available at: https://arxiv.org/abs/2010.11929 .

GitHub. (2024). tudelft-iv/view-of-delft-dataset: This repository shares the documentation and development kit of the View of Delft automotive dataset. [online] Available at: https://github.com/tudelft-iv/view-of-delft-dataset?tab=readme-ov-file#introduction

kili-website. (2023). *Mean Average Precision (mAP): A Complete Guide*. [online] Available at: https://kili-technology.com/data-labeling/machine-learning/mean-average-precision-map-a-complete-guide

GitHub. (2024). *OpenPCDet/docs/multiple_models_demo.png at master · open-mmlab/OpenPCDet*. [online] Available at: https://github.com/open-mmlab/OpenPCDet/blob/master/docs/multiple_models_demo.png

Guan, S., Wan, C. and Jiang, Y. (2023). *AV PV-RCNN: Improving 3D Object Detection with Adaptive Deformation and VectorPool Aggregation*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/371321106_AV_PV-RCNN_Improving_3D_Object_Detection_with_Adaptive_Deformation_and_VectorPool_Aggregation

Aha (2024). Agile Software Development - Agile Methodology Explained. [online] Www.aha.io. Available at: https://www.aha.io/roadmapping/guide/agile/agile-software-development

Pan, Z., Ding, F., Zhong, H. and Lu, C.X. (2023). *RaTrack: Moving Object Detection and Tracking with 4D Radar Point Cloud*. [online] arXiv.org. Available at: https://arxiv.org/abs/2309.09737 [Accessed 20 Apr. 2024].

GitHub. (2024). *RaTrack/doc/ratrack_pipeline.png at main · LJacksonPan/RaTrack*. [online] Available at: https://github.com/LJacksonPan/RaTrack/blob/main/doc/ratrack_pipeline.png

Pan, Z., Ding, F., Zhong, H. and Lu, C. (n.d.). *RaTrack: Moving Object Detection and Tracking with 4D Radar Point Cloud*. [online] Available at: https://arxiv.org/pdf/2309.09737.pdf.

Alifya Febriana (2023). *Setting Up 3D Open Source OpenPCDet with Anaconda: A Step-by-Step Guide*. [online] Medium. Available at: https://medium.com/@alifyafebriana/setting-up-3d-open-source-openpcdet-with-anaconda-a-step-by-step-guide-66126107215 [Accessed 25 Apr. 2024].