BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

PEOPLE DETECTION USING RADAR
DETEKCE OSOB S POUŽITÍM RADARU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR JAKUB BARTKO
AUTOR PRÁCE

SUPERVISOR Ing. LUKÁŠ MARŠÍK
VEDOUCÍ PRÁCE

BRNO 2022

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2021/2022

Bachelor's Thesis Specification

24905

Student:	**Bartko Jakub**
Programme:	Information Technology
Title:	**People Detection Using Radar**
Category:	Signal Processing

Assignment:

1. Study theory of convolutional neural networks and get familiar with radar sensor based on radar-on-chip platform.
2. Propose suitable data structure for representation of radar point cloud.
3. Collect and pre-process dataset for object recognition.
4. Select suitable method based on neural network and perform its training.
5. Design and implement application that performs classification of radar detections.
6. Carry out series of tests of implemented application and evaluate performance and reliability of recognition. Discuss possible future work as well.
7. Create a short video presenting your work, achieved goals and your results.

Recommended literature:

- M. Skolnik: Radar Handbook, 3rd edition, McGraw-Hill Professional, 2008
- M. A. Richards: Fundamentals of Radar Signal Processing, 1st edition, McGraw-Hill, 2005
- B. R. Mahafza: Radar Signal Analysis and Processing Using MATLAB, Chapman and Hall, 2008

Requirements for the first semester:

- Items 1 to 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Maršík Lukáš, Ing.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2021
Submission deadline:	May 11, 2022
Approval date:	November 1, 2021

Abstract

This thesis aims to research the applicability of deep learning methods on point clouds generated by millimeter-wave radars, as a solution for people detection, and 3D scene understanding in general. Radar is a system that uses radio waves to determine the distance, azimuth, and velocity of surrounding objects. For each point of detection, Cartesian coordinates can be calculated, to produce a set of points in 3D space called a point cloud. Deep neural network architectures designed to operate on sparse 3D point clouds can be trained for point-wise segmentation, object detection, classification, and tracking. This can be used to greatly advance the 3D scene understanding by machines. A model based on the state-of-the-art methods for object detection and classification on sparse point clouds was trained as a part of this thesis, for the purpose of people detection. To showcase the robustness of the trained model and the straightforwardness of its applicability to solve prominent real-world tasks of scene understanding, a people counting application was developed. The employed methods were thoroughly evaluated on a dataset created as a part of this thesis, consisting of over 19,500 labels on 3D radar point clouds.

Abstrakt

Cieľom tejto práce je výzkum použiteľnosti techník hlbokého učenia na mračnách bodov generovaných radarmi, ktoré operujú v pásme milimetrových vĺn; na účely porozumenia 3D scénam a rozpoznávania osôb. Radar je systém, ktorý využíva rádiové vlny na získavanie informácií o vzdialenosti a smere k objektu a o jeho rýchlosti. Na základe týchto dát je možné určiť pozíciu každého detekovaného bodu v priestore. Množina takýchto bodov sa nazýva mračno bodov. Hlboké neurónové siete navrhnuté na prácu s riedkymi mračnami bodov v 3D môžu byť natrénované na segmentáciu radarových mračien, detekciu a klasifikáciu objektov a sledovanie ich pohybu v priestore. Systémy, ktoré sú na nich postavené, môžu predstavovať veľký pokrok v oblasti interpretácie 3D scén autonómnymi strojmi. V rámci tejto práce bol model budujúci na súčasnom stave techniky natrénovaný na detekciu a klasifikáciu objektov a osôb, na základe radarových mračien bodov v 3D. Na ukážku robustnosti natrénovaného modelu a priamočiarosti jeho zapojenia na riešenie významných úloh z reálneho sveta, bola implementovaná aplikácia na počítanie a klasifikáciu ľudí. Presnosť využitých metód aj výsledného systému bola dôkladne vyhodnotená na súbore dát vytvorenom v rámci tejto práce z radarových mračien bodov, ktorý pozostáva z viac, než 19 500 anotovaných objektov.

Keywords

millimeter-wave radar, radar point cloud, 3D point cloud annotation, people detection, people counting, 3D scene understanding, PointNet++, PointRCNN

Kľúčové slová

radar operujúci v pásme milimetrových vĺn, radarové mračno bodov, anotácia mračien bodov v 3D, detekcia osôb, počítanie osôb, porozumenie 3D okoliu, architektúra PointNet++, architektúra PointRCNN

Reference

BARTKO, Jakub. *People Detection Using Radar.* Brno, 2022. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Lukáš Maršík

Rozšírený abstrakt

Narastajúca komplexnosť systémov a strojov nevyhnutných na správny chod každej modernej spoločnosti a množstvo oblastí ich využitia, ktoré pribúdajú každým dňom, predstavujú primárne sily, ktoré ženú rozvoj inteligentných systémov a ich využitie na porozumenie svojmu okoliu. Ľudské oko a mozog, ktorý sa za nim schováva, sú znamenito prispôsobené na pozorovanie svojho okolia a spracovávanie týchto vnemov, no len v posledných rokoch sa začala výkonnosť strojového vnímania dožadovať celospoločenskej pozornosti. Tieto pokroky boli v posledných desaťročiach poháňané prelomovými objavmi v oblastiach strojového učenia a počítačového videnia. Jedným z konečných cieľov týchto oblastí je vnímanie reálneho okolia, ktoré by konkurovalo ľudskému zraku. Inteligentné systémy, ktoré sa toto snažia implementovať, dnes nachádzajú využitie od monitorovania a analýzy dopravy, ľudského pohybu, stavu úrody aj rentgenových snímok, až po autonómne vozidlá a robotiku. Takéto systémy často využívajú viacero senzorov na snímanie svojho okolia. Odborníci sa rastúcim dôrazom na ich komplementaritu snažia vyriešiť nedostatky jednotlivých senzorov, ktoré sú spôsobené samotným spôsobom, akým snímajú svoje okolie. Jedným z popredných senzorov, využívaných inteligentnými systémami, je radarový systém, ktorý zavádza technológiu minulého storočia na riešenie úloh porozumenia 3D okoliu, riadenia autonómnych áut, monitorovania dopravy a pohybu chodcov a zdravotného stavu osôb. Radarové senzory už dnes dominujú v riešení úloh, ako detekcia prekážok, parkovacia asistencia, monitorovanie mŕtveho uhla, detekcia pohybu, autonómne riadenie poľnohospodárskych strojov, kontrola kvality výrobkov a neinvazívne monitorovanie zdravotného stavu osôb.

Najmodernejšie pokroky v strojovom učení na riedkych mračnách bodov v 3D priestore priniesli nové možnosti porozumenia 3D okoliu. Moderné radarové zariadenia operujúce v pásme milimetrových vĺn, ktoré sú vďaka miniatúrnej veľkosti svojich kompomentov integrované priamo na čipe, predstavujú významnú možnosť energeticky úsporného snímania okolia v 3D, ktoré nie je znehodnotené nedostatkom svetla, extrémnym počasím, ani rušením ostatnými senzormi, no svojou presnosťou a informačným prínosom svojich dát konkuruje aj najmodernejším kamerovým inteligentným systémom. Cieľom tejto práce je preskúmať metódy využívajúce mračnách bodov v 3D, ktoré sú produkované radarovými senzormi operujúcimi v pásme milimetrových vĺn. Tieto metódy sú potom využité na vytvorenie systému na detekciu a klasifikáciu osôb na základe radarových detekcií. Na tieto účely je natrénovaný model na sade dát, ktorá je zostavená z radarovej nahrávky z cyklotrasy, s využitím referenčného videa, na anotáciu týchto dát. Výkonnosť a presnosť tohoto modelu sú následne dôkladne vyhodnotené a analyzované. Tento model je ďalej využitý na implementáciu systému na počítanie ľudí. Jeho výsledná presnosť je skoro na nerozoznanie od manuálneho počítania chodcov a cyklistov z referenčnej video nahrávky. Jednoduchosť metód, ktoré tento systém využíva, umožňuje jeho prispôsobenie a nasadenie na počítanie ľudí, monitorovanie dopravy, monitorovanie obsadenosti miestností, automatické otváranie dverí a ďalšie, v reálnom čase a s nízkymi energetickými požiadavkami.

Výstupom tejto práce je anotačný nástroj využiteľný na efektívnu a presnú anotáciu riedkych radarových mračien bodov v 3D podľa referenčného videa; sada dát vytvorená týmto procesom; model natrénovaný na detekciu a klasifikáciu osôb nasnímaných radarovým senzorom a aplikácia na počítanie ľudí a sledovanie ich pohybu.

People Detection Using Radar

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Lukáš Maršík. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Jakub Bartko
May 17, 2022

Contents

Chapter 1

Introduction

Due to the growing complexity of systems and machines indispensable to the function of modern human society and their use becoming more widespread by the day, increasingly more attention is being allocated to the intelligence behind the interactions of such systems and their surroundings. The human eye and the brain behind it might have grown accustomed to sensing and processing great amounts of highly complex inputs, but only in the most recent decades have we seen the real power of machine perception begin to manifest. These advances have been greatly fueled by the breakthroughs of recent years in the field of machine learning. Among others, a human-like perception of one's surroundings might be considered one of the ultimate goals of this field, with applications ranging from traffic, crops, or x-ray analysis, to people counting, autonomous driving, and robotics. Systems dedicated to scene understanding are often comprised of numerous sensors, but while the great advancements in camera-based computer vision have rendered image processing the go-to method for scene understanding, only recently have alternative methods of sensing come to be considered a competitive solution. The growing emphasis on their complementarity might be attributed to the fact, that each type of sensor, each physical medium used for sensing, and each analytical method the sensed data allows for, introduces a unique set of challenges posed by the environment, processing of data and the cost of manufacturing. Such challenges are often overcome by employing multiple different types of sensors, not limited by the same set of technical obstacles. Among the most prominent of them is the **ra**dio **d**etection **a**nd **r**anging system — *a radar*. It reintroduces technology over a century old to applications in scene understanding, automotive or pedestrian detection, tracking, and classifying of objects. The very recent advancements in machine learning on sparse 3D point clouds have shun light on new approaches toward machine understanding of 3D scenes.

The contributions of this thesis include the in-depth study of the state of the art of sensors and methods used for 3D scene understanding, with a particular focus on the state-of-the-art millimeter-wave frequency-modulated continuous-wave radars based on the radar-on-chip platform. It is followed by the adaptation of an open-source tool for efficient annotation of sparse 3D radar point clouds using referential video, and the adaptation of the state-of-the-art deep learning methods for object detection on lidar point clouds for people detection on sparse radar point clouds. As a part of this thesis, a dataset of 3D radar point cloud annotations was produced. Published datasets of this nature are either very few or do not match the general requirements for people detection set out in this thesis. The model trained as a part of this thesis and the subsequent tracking algorithm render the final products of this thesis straightforwardly deployable for real-world tasks, such as people counting, occupancy monitoring, trajectory prediction, and collision avoidance. This is also showcased by the application developed using the methods detailed in this thesis, together with the robustness of the trained model and the subsequent tracking algorithm.

This thesis aims to research the methods for scene understanding on sets of points in 3D space detected by radar systems, with a particular focus on people detection. The initial part of this thesis — in Chapter 2, is focused on briefly summarizing the current state of systems used for scene understanding. Next, a thorough introduction to the components and principles of radar and point cloud data is presented in Chapter 3. It is followed by Chapter 4 on deep learning on point clouds, introducing the challenges, goals, and an overview of deep learning approaches relevant to this thesis. Further, this chapter introduces the two architectures, on which the system used in this thesis is based. The data processing and annotation, and the tool adapted for effective 3D point cloud annotation, are presented in Chapter 5, along with the analysis of the final dataset. This dataset is then used for model training, which is discussed in Chapter 6, together with the model's implementation and configuration details, and a detailed evaluation of its performance. The deployment of the model, the people counting application, evaluation of its performance, and the tracking algorithm it employs, are presented in Chapter 7. The results, applicability, and future work are finally discussed in Chapter 8.

Chapter 2

Summary of the Current State

This chapter presents an overview of current options in sensors most broadly used for scene understanding. Here, the underlying sensing mechanisms, both the advantages and shortcomings, specific use cases, and methods of data processing of each sensor, are listed. A general understanding of the underlying principles and both positives and negatives of employing a specific sensor to gather information on its surroundings is certainly of use when designing and evaluating a system for these purposes of scene understanding. In particular, the camera, lidar, and radar sensors and the systems that incorporate them, are presented here. A detailed introduction to radar systems and their principles follows in Chapter 3.

2.1 Tools and Methods for Scene Understanding

The three sensors discussed here and systems that incorporate them are the backbone of many intelligent systems, namely in the fields of autonomous driving, robotics, traffic analysis, people counting, health monitoring, Internet of Things (IoT), and many others. Scene understanding is roughly the process of perceiving and analyzing a snap shot of a real world scene with the goal of its interpretation. This is usually done through a network of complementary sensors used for observation of the surrounding environment. Each of the sensors presented below makes use of waves from the electromagnetic spectrum reflecting off of objects also in the direction of the sensor, which in turn captures and processes them to acquire information on the objects. A comparison of the sensing technologies most commonly used for scene understanding, with human perception for analogy, is illustrated in Table 2.1.

Camera Systems

Most computer vision systems of today employ cameras operating in the spectrum of visible light, used with great success in autonomous driving, robotics, and traffic analysis. A camera operates by illuminating an array of cells sensitive to light, with light reflecting off of surrounding surfaces. This generates a certain level of electrical charge, which can be converted to a digital value, that is finally used for conversion into digital image [28]. To further process the retrieved images and to extract information useful for decision-making elements of intelligent systems — neural networks, e.g., Convolutional Neural Networks (CNNs), are most often used. CNNs have become the most widely used deep learning method for images [6]. The CNN architecture is inspired by the visual system structure of

Table 2.1: Sensor comparison of performance on scene understanding tasks [28].

Performance aspect	**Human**	**Sensor**		
		Radar	*Lidar*	*Camera*
Object detection	Good	Good	Good	Fair
Object classification	Good	Poor	Fair	Good
Distance estimation	Fair	Good	Good	Fair
Edge detection	Good	Poor	Good	Good
Lane tracking	Good	Poor	Poor	Good
Visibility range	Good	Good	Fair	Fair
Poor weather performance	Fair	Good	Fair	Poor
Low illumination performance	Poor	Good	Good	Fair

the human brain. They are proven the best learning algorithms for understanding information available from images and are regarded as the ideal model for the various image-related tasks like classification, detection, and image retrieval [26].

In favorable light conditions, cameras are great at recognizing an object's color (used e.g., to recognize traffic lights, road signs and lines). This makes them very suitable for collision and lane departure warning systems, traffic sign recognition, parking assistance and blind-spot monitoring [28]. Cameras are widely used for people detection, people counting and occupancy monitoring. Camera-based vision systems tend to suffer from adverse weather and light conditions. Rain, snow, dust, direct sunlight and lack of light severely affect the accuracy and the recognition capabilities of these systems. The recognition aspect of using cameras for monitoring also introduces the prospect of being misused for the intrusion of privacy, e.g., recording private property or collecting data on the behavior of individual citizens. Although cameras achieve high resolutions even at greater distances, the increased cost limits the multi-camera monitoring systems to much lower resolutions. Thus, the detection range and measurements of distance and speed are often limited to the tens of meters [21].

Lidar Systems

Lidar sensor also retrieves images of its surroundings by making use of the light rays reflected off of surfaces. Unlike cameras, the lidar systems also comprise the source of the light rays to be reflected, and the images produced are in 3D space, and in comparatively high resolution. Lidar sensor works by emitting pulse modulated light. It illuminates a target with an optical pulse, reflection of which is measured and analyzed to detect points of reflection, as well as their distance, angle and intensity of reflection [10]. Lidars are seeing a growing popularity namely for autonomous applications, with the adoption of new technologies aimed to reduce their cost and comparatively large size. They are also used for applications in 3D aerial and geographic mapping, and building, interior and product modeling.

Lidar excels at the spatial resolution, meaning that it can deliver extremely high-resolution characterization of objects in 3D, without substantial processing efforts, as is the case with cameras. While both lidar and radar sensors provide an excellent field of view (FoV), a single mechanically rotated lidar sensor can provide a 360-degree FoV, matching the state-of-the-art advanced driver-assistance systems of 8 or more cameras for autonomous driving, without the substantial effort necessary for their synchronization and calibration. Both radar and lidar systems are robust against inference from other sensors.

A significant shortcoming of lidar is its cost, far surpassing that of camera and radar systems, but that has also been steadily declining over the recent years [10, 29]. Due to the physical medium for sensing the environment being light, lidars also suffer the environmental and weather accuracy deficiencies of cameras — due to rain, snow, fog and dust, all be it to a comparatively lesser extent than in the case of a camera, as the light rays a lidar generates are outside of the visible spectrum. On the other hand, lidar faces nearly no obstructions at night, unlike camera systems, which would require an additional infrared camera to operate, and lidar sensors produce much more populated point clouds than radars. This provides greater detail of the surroundings on one hand, but on the other, the lidar point clouds pose far greater processing demands.

Lidars produce data in the point cloud format — further discussed in Section 3.3. A point cloud is a set of points in 3D space, each representing a 3D coordinate — the point's position in space, and additional data (speed, angle, intensity, normal, etc.). Deep neural networks are becoming the go-to approach for object detection on 3D lidar point clouds (also discussed in Chapter 4). Other traditional methods include k-means and DBSCAN clustering, that might be applied recursively to learn the scene details at different scales.

Radar Systems

Radar sensors based on the radar-on-chip platform are steadily gaining a foothold among the most prominent methods for scene understanding, object detection, classification, and tracking. As of today, they are the dominant technology for the tasks of obstacle detection, parking assistance, blind-spot monitoring, motion sensors, autonomous driving in agriculture, quality control, and health monitoring. The continual development of semiconductor technologies and improvements of frequency modulated continuous-wave radars (see Section 3.2) in the millimeter-wave range have paved a way for its widespread adoption namely in robotics and autonomous driving. The small components of a millimeter-wave radar have allowed for the miniature size and low cost of modern-day radars, that are commonly integrated into a single chip.

Radars are used to implement versatile, highly accurate, and robust systems at a low cost for reliable sensing and comparatively undemanding data processing, for the task of scene understanding. They transmit electromagnetic wave signals by transmitting antennas, which are reflected by objects in their path. Reflected waves are captured by receiving antennas and further analyzed to gain data on the range, velocity, and angle of the objects [8]. For a more detailed description of radar systems, see Chapter 3.

The output of a radar sensor can be in form of the received signal used for further signal processing and analysis, but it is becoming standard for modern radar-on-chip systems to produce point clouds, comprised of spherical or Cartesian coordinates, Doppler velocity, and a time stamp. This format is further discussed in Section 3.3. It also allows for the adoption of deep learning methods used on lidar point clouds, also discussed in Chapter 4.

Radar, unlike its counterparts, remains unaffected by extreme weather and poor lighting conditions. Automotive and industrial radar sensors are cheap to produce and consume little power for both sensing and signal processing. They can detect objects with high velocity difference and at a great distance, and the state-of-the-art millimeter-wave radars are capable of detecting movements as small as a fraction of a millimeter [8]. Short-, medium-, and long-range radars perform very well at their designed distances with decent resolution, yet the point clouds they produce are far sparser than lidar point clouds.

Methods for object detection on radar data include the processes of generating point clouds from sensed reflections, clustering, and tracking (e.g., using the Kalman filter, also discussed in Section 7.2). A robust implementation of this method is presented in [18]. This method, however, relies on good results of its clustering stage, that might also diminish the accuracy of further scene understanding tasks, such as object classification. Another large group of object detection methods builds on neural network models designed to operate on point clouds. These methods are already the go-to choice for object detection and classification on lidar point clouds. Deep neural networks for object detection on radar and lidar point clouds are further discussed in Section 4.1. A more detailed description of radar system elements, principles, applications, and methods for scene understanding follows in Chapters 3 and 4.

Chapter 3

Radar Technology and Data Representation

Radar is an electromagnetic sensor for the detection and location of objects reflecting energy [25]. It radiates electromagnetic energy to its surroundings, detecting the reflections from any nearby objects. The time of signal's travel, its frequency shift, magnitude and further signal processing is used to derive information on the object's distance, velocity and angle relative to the radar, as well as other characteristics of the object. This chapter provides a brief overview of radar applications, introduction to the principles of this technology and its outputs, with a closer look at point clouds.

3.1 Modern-Day Applications of Radar

Although radar owes a great portion of its renown and numerous early developments to its military applications, its use has since spread throughout countless civilian fields, ever-detecting over the air, sea, land, vacuum and underground. Below are listed some of the most prominent applications of radar systems, as presented in [1, 25].

Air Travel

Air Traffic Control (ATC) radars are used in the vicinity of airports for the purposes of guiding aircraft for landing in adverse weather conditions. Other radar systems are employed for aircraft navigation, avoiding uncomfortable weather, collision avoidance, and guiding aircraft on the ground on larger airports. Airports also employ weather observation radars.

Marine Radars

Radar systems installed in ports and mounted on ships are of similar use as the ones on airports and aircraft. In addition to the collision avoidance and navigation in poor weather conditions, they are used for measuring the depth of sea and ice and are also used to monitor the dynamics, decay, and growth of ice sheets.

Space

Radars are commonly employed for spacecraft docking and landing. Ground-based radars are also used for detecting and tracking satellites and spacecraft. Radar sensing has been famously used for the exploration of the surface of the planet Venus to penetrate its thick layer of clouds.

Remote Environment Sensing

Radars provide some of the most reliable and widely used methods for weather observation and forecast, used to measure wind speed and direction, detect wind conditions dangerous for aircraft, and measure the mean sea level. Radars are commonly used to study formations of clouds to determine their type for a weather forecast. Special weather avoidance radars have been designed for installment within the nose of both small and large aircraft.

Military

The first radars in use were developed because of the necessity of defense against bomber aircraft attacks. Military use of radar has been the driving force behind a great portion of its major developments, including those for civilian purposes. Radar has been employed in a wide range of air, naval and ground defense tasks for long-range surveillance, tracking and recognition, weapon and fire control, and damage assessment. It is used for the detection, tracking, and navigation of ballistic missiles, mortars, artillery, and unmanned aircraft.

Other applications include ground-penetrating radars used for locating buried objects, bodies, and artifacts by police and archaeologists, observing the migrations of birds and insects, seepage of hazardous gasses, and one of the most common forms to encounter — radar speed guns used by the law enforcement.

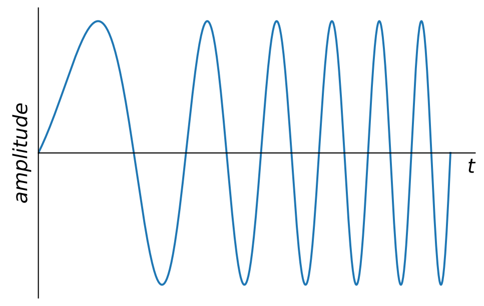
Millimeter-Wave radars

The millimeter-wave radars are a special class of radar technology used in this thesis to produce recordings used to create a dataset of point cloud annotations. The miniature size of its components allows for this type of radar to be easily deployed in vehicles, robots, and elevated places for traffic monitoring and people counting. The automotive applications include obstacle detection, object tracking, collision avoidance, lane departure warning, parking assistance, and blind-spot monitoring. In modern car systems, vital sign monitoring systems are developed using millimeter-wave radars. Other medical applications include heartbeat, breathing, and sleep monitoring. They are also used as motion sensors and for room occupancy monitoring. Meteorological applications mentioned above also often employ this type of radar.

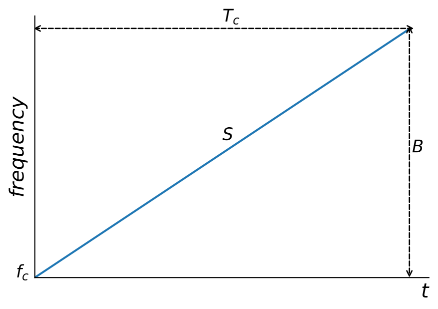
3.2 Millimeter-Wave, Frequency-Modulated Continuous-Wave Radar

The radar device used to produce the recordings for the final dataset discussed in Chapter 5, is a millimeter-wave radar sensor that uses the *frequency-modulated continuous-wave (FMCW)* technology. The following introduction of this technology is inspired by the Texas Instrument's article on the millimeter-wave radar sensors [8].

Millimeter-wave radar, as the name suggests, transmits waves with a wavelength in the millimeter range. Among the advantages of using such a comparatively short wavelength are the small size of system components and high accuracy — up to a fraction of a millimeter. FMCW radars transmit signals, where frequency increases linearly with time, as shown in Fig. 3.1. The „chirp“ signal is characterized by its start frequency (f_c), bandwidth (B), duration (T_c), and the slope of the chirp (S), which captures the rate of change of frequency.



(a) Amplitude as a function of time



(b) Frequency as a function of time

Figure 3.1: Chirp signals.

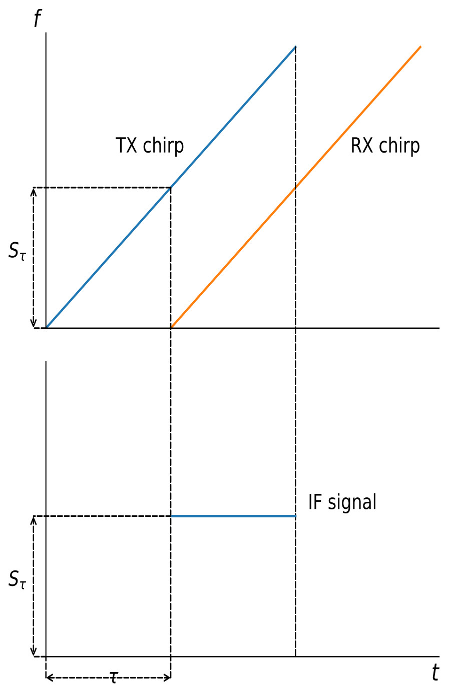
The FMCW radar operates along these steps:

- A syntethizer generates a chirp.
- The chirp is transmitted by a transmit antenna.
- The chirp reflected by an object is captured by the receive antenna.
- A mixer combines the transmitted (TX) and received (RX) signals to produce an intermediate frequency (IF) signal

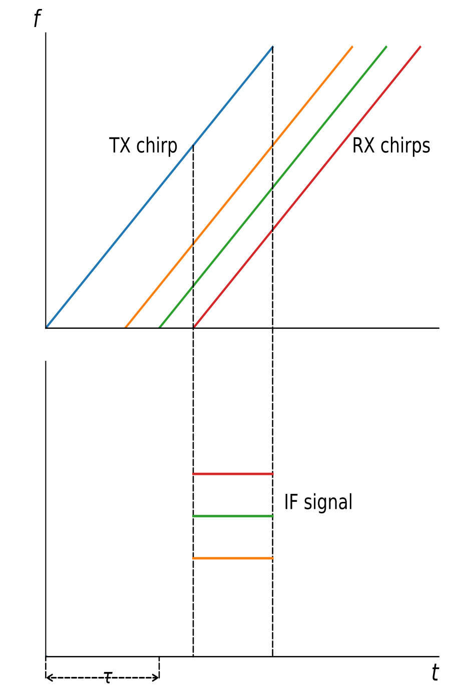
Range measurement

The IF signal produced by the mixer has an instantaneous frequency equal to the difference of the instantaneous frequencies of the two input signals. Analogically, the phase of the output signal is equal to the difference of the phases of the two input signals. For a single object detected, the TX and RX chirps, and the IF signal are shown as functions of time in Fig. 3.2a. The RX chirp is the same as the TX chirp, delayed by the time delay τ. The frequency of the output IF signal is the difference of the TX and RX signals, on the time interval where the two signals overlap. As the difference between the two signals is constant, the IF signal consists of a single frequency. The initial phase of the IF signal is the difference between the phases of TX and RX chirps at the time instant that corresponds to the start of the RX and IF signals.

In the case that multiple objects are detected, the RX antenna receives different RX chirps from different objects, as shown in Fig. 3.2b. Here, the different IF signals of constant frequencies need to be separated using a Fourier transform. The output of this process is a frequency spectrum with separate peaks, corresponding to the different objects at specific distances from the sensor.



(a) Single object detected.



(b) Multiple objects detected.

Figure 3.2: IF signals.

Velocity measurement

To measure velocity of a single object, FMCW radar transmits two chirps separated by a time delay T_c. Each of the reflected chirps is then processed using the *fast Fourier transform (FFT)* to detect the range of the corresponding object. The corresponding range-FFTs will have peaks in the same location, but with different phase. The difference of these two phases represents the motion of the detected object.

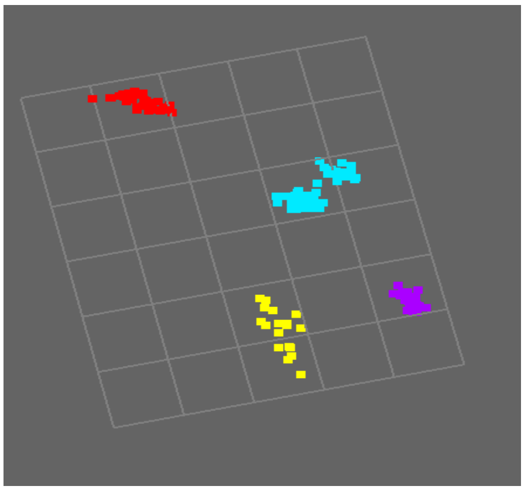
However, this method fails to account for multiple moving objects with different velocities measured at the same distance from the radar, because due to their same distance, their RX chirps will have identical IF frequencies. This way, the range-FFT would have only a single peak. In order to measure the speed in this case, more than two equally spaced chirps must be transmitted. Using range-FFT to process the N transmitted equally spaced chirps produces N identically located peaks, each with a different phase that incorporates the phase contributions from all of the detected objects (at the same distance from the sensor, but with different velocities). Finally, the Doppler-FFT is performed on the N phasors to resolve these objects.

Angle estimation

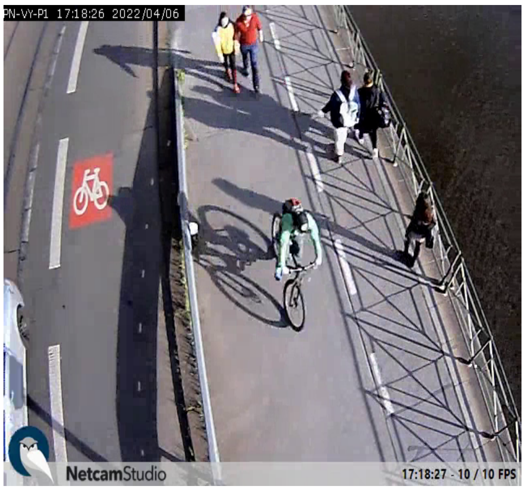
To estimate the *angle of arrival* (**AoA**) of the RX signal, FMCW radars build on the observation that a small change in the distance to an object results in a phase change at the peak of the range-FFT or Dopler-FFT. Using at least two RX antennas, the differential distance from the object to each antenna results in a phase change at the FFT peak, which enables the estimation of the AoA. The accuracy of the AoA estimation degrades as the angle increases.

3.3 Radar Data and Point Clouds

Using the measurement processes introduced in the previous section, data on object's distance to sensor, velocity and angle of arrival (AoA) can be derived. This in itself allows for many use cases, also presented in Section 3.1. Using the object's distance and AoA, coordinates in the Cartesian coordinate system can be calculated, resulting in a set of coordinates (x, y, z) per each point of detection. Depending on the resolution of a radar sensor, multiple reflections from an object can be detected. By also incorporating other additional information provided by the sensor, a rich set of points in 3D space can be constructed. Such a set of points is commonly referred to as a *point cloud*.



(a) Visualisation of radar point cloud clusters.



(b) Reference camera image.

Figure 3.3: Radar point cloud example. The images are from the annotation stage, described in Chapter 5.

This format of data is most commonly generated by radar and lidar sensors. The 3D data they produce leave out the necessity to introduce complex algorithms to learn the depth of a captured image, unlike with cameras. Although the point clouds are far more sparse than e.g., images, depending on the configuration of the sensor, they can be used to learn even fine geometric features, which is far more challenging on 2D images. It is also more straight forward to separate individual objects in a 3D scene, as they are separated naturally and do not overlap. Using a radar sensor to produce the point cloud, static objects that might not be of interest can be easily filtered out. The shape of larger objects might be very efficiently estimated using common clustering algorithms, such as DBSCAN. The geometric features of the detected objects can also be used for object classification. Methods developed for this task are further discussed in Chapter 4.

Point clouds are used to a great extent for semantic scene understanding, with applications in autonomous driving, robotics, remote environment sensing, but also 3D modeling of buildings and products, and others. In this thesis, point clouds are used to represent the radar detections of objects and people, as shown in Fig. 3.3. Using this data format, such detections were be annotated to create a dataset later used to train a deep neural network to detect pedestrians and cyclists.

Chapter 4

Deep Learning on Point Clouds

The great developments in the field of computer vision have driven the deep learning techniques to an extraordinary level and rendered them the go-to method for most vision-related tasks on 2D images. These methods, however, are not directly applicable to the irregular and unordered point cloud data. That is due to the fact, that the typical neural network architectures, namely Convolutional Neural Networks (CNNs), require highly regular input data formats [2, 13]. This chapter provides an overview of the partial goals in 3D scene understanding, approaches to object detection on point clouds, and an introduction to the two architectures, on which the model trained as a part of this thesis, is built. The underlying goals of 3D scene understanding regarding people detection are presented below.

Semantic segmentation is the process of a point-wise classification. This means that for each point in a point cloud, a class label is predicted. An example of an output of this process would be a set of points in a scene categorized as belonging to a building, a tree, a vehicle, a pedestrian, or an undesired background point.

Instance segmentation clusters the points of a scene into object instances [13]. The output of this process would be a set of clusters predicted to be individual object instances. It is usually followed by a classification algorithm to infer a set of objects and their predicted labels.

Bounding box estimation is the process of estimating boundaries of individual objects. As in [23] it might be preceded by a semantic segmentation algorithm, and followed by class prediction — analogically to object detection on images. This would also result in output similar to a combination of semantic and instance segmentation, where each point would be assigned a class and instance label. Here, this would be accomplished by assigning the points to the respective bounding boxes with a simple inside-or-outside test.

Object tracking is the process of determining or predicting the position of an object throughout the time frame where it is detected. This is usually accomplished by inferring the associations of detected objects across multiple frames. This way, the movement of pedestrians, cyclists, vehicles, etc. might be monitored and predicted. The output of this process might be the track an observed object has traveled and the direction it is currently headed.

The combination of these processes would allow for multiple scene understanding tasks. For the purposes of this thesis, their suitable combination could be used for a robust detection of people, predicting their classes (e.g., a pedestrian or a cyclist), estimating their respective bounding boxes, and monitoring their movement. For the purposes of this thesis, this is considered the joint task of people detection.

4.1 Approaches to Object Detection

Object detection in the context of 3D scene understanding can also be described as the joined task of semantic and instance segmentation of a scene. The output of this process would be a set of clusters representing the scene objects labeled by an appropriate class label. Example of such output is show in Fig. 4.1. In this section, five distinct approaches toward object detection will be compared, as presented in [22].

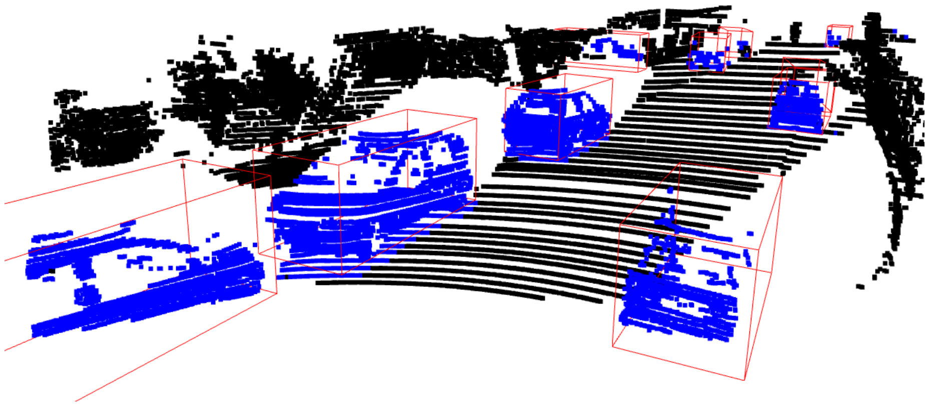


Figure 4.1: Object detection on lidar point clouds. The points were segmented (shown as blue and black points) using a PointNet++ backbone, discussed in Section 4.2, and the bounding boxes estimated using a PointRCNN model, discussed in Section 4.3. The visualized point cloud is part of the KITTI 3D Object Detection Evaluation 2017 dataset [5].

Image Object Detection Network

This approach aims to utilize one of the most prominent methods for object detection on 2D images — a convolutional neural network (*CNN*) image object detector. This introduces the necessity to arrange the data in an image-like structure, usually accomplished by grid mapping. The grid maps might incorporate the number of radar points per cell, point amplitudes, and maximum and minimum Doppler values per each cell [4]. The map representing the density of radar points might be used to populate the neighbors of cells where the number of points exceeds an empirically set threshold.

YOLOv5 model [16] poses a good compromise between accurate object detection and computational efficiency. This model divides images into grids, with each cell responsible for object detection within itself. It is a deep, fully-convolutional network for one-stage object detection. The input image is down-sampled by multiple factors to produce features at different scales. At each scale, a detection head[1] is used with an additional convolutional

[1]**Object detection head** is an element used for object prediction on features extracted by a backbone network.

layers to process the features with objectness, classification and localization loss. The losses are used to predict whether an object is present at a given location, to predict the class label, and to regress the position of the bounding box and its size, respectively.

This approach, along with others aiming to transform the point cloud into pixel or voxel representation beforehand, has been considered by some competitive works to be unnecessarily voluminous and that might introduce artifacts that can obscure the natural invariances of the data [2, 13]. Additionally, 3D convolution used to learn the features of 3D voxels is both memory and computation inefficient [23].

Clustering and Recurrent Neural Network Classifier

An intuitive approach to solving the problems identified by Qi *et. al.* in [2], mentioned in the beginning of this chapter, would be to follow one of the solutions proposed by the authors — split the scene into separate objects and focus on single-object identification afterward, employing a recurrent neural network classifier. In this approach, the input point cloud is first segmented using a two-stage clustering algorithm based on **DBSCAN**. ***D**ensity-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise (DBSCAN)* is one of the most widely used clustering algorithms. It uses two parameters — neighborhood size and minimum number of points to form a cluster. It visits a random unvisited point, and clusters the points in its neighborhood and the neighborhood of the points in its cluster, in case they exceed the minimum number of points parameter. The algorithm can be further improved by removing all points below some absolute velocity threshold. This allows for a wider neighborhood parameter for DBSCAN without including an excessive amount of clutter. The standard DBSCAN algorithm is further modified to account for lesser densities of points at greater distances by implementing the minimum-amount-of-neighbors parameter as range-dependent.

The clustering is followed by a classifier that decides whether each cluster resembles an object, or is labeled as the background (or clutter) class. Each cluster is then sampled using a sliding window. For each of those samples, a large vector of handpicked feature candidates (simple statistical features — mean or maximum Doppler values, but also more complex ones, like the area of a convex hull around the cluster) is extracted. A set of binary classifiers (based on *long short-term memory*[2] cells.) is then used to classify the clusters using the extracted feature vectors.

Semantic Segmentation Network and Clustering

This approach aims to improve on the limitations of the clustering stage of the previous approach. In the previous approach, the clustering parameters are empirically chosen, and the good clustering results the subsequent stages are dependent on, are often not achieved. This is improved on by introducing a classification process before clustering, also referred to as semantic segmentation for point-wise class prediction. The class predictions provide the clustering algorithm with additional information vital to the precision of the later stages.

The backbone model utilized for this task is most commonly the state-of-the-art PointNet or the improved PointNet++ architecture. This architecture uses a symmetry function for unordered point cloud input to aggregate the information from each point into a global feature vector of the input point cloud. It is fed back to per point features, allowing per point predictions of the network utilizing both local geometry and global semantics [2, 14].

[2]**Long short-term memory (LSTM)** is an artificial recurrent neural network architecture [7].

The two-stage clustering method of the previous approach is adapted to incorporate the class labels in both the filtering and the clustering stages. In addition to filtering by velocity threshold, points predicted to be background or noise are omitted. The DBSCAN algorithm is extended by a class distance parameter to account for class differences by weighting in the distance between individual pairs of classes. A clustering run is performed for each of the object classes. The final class label for the cluster is selected by a voting scheme, weighting in the most frequent label prediction within the cluster.

Point-Cloud-Based Object Detection Network

Networks utilizing this approach use an intermediate pseudo image representation on which features are learned. One of the most prominent architectures among them is PointPillars [12]. It consists of three stages: a network for feature encoding, a simple 2D CNN, and a detection head for bounding box estimation.

The encoder produces a sparse pseudo image by dividing the point cloud into grids, producing additional data on the pillar an individual point belongs to. Limitations on the amount of both pillars and points are imposed, making the non-empty pillars sparse and each with few points within them. From there, a miniature PointNet [2] is used to infer a multidimensional feature space. Features are then extracted by a CNN backbone at three different scales, up-sampled and passed to the detection head, similarly to the image approach and using the same set of losses together with a size loss and an angular loss. This solution has been further improved by replacing the CNN backbone with the Darknet-53 backbone from YOLOv3, reported as the PointPillars++ architecture.

Semantic Segmentation as Proposals for Region-Based Convolutional Neural Network

This approach also utilizes a PointNet backbone to segment the scene into foreground and background points. The learned features of foreground points are simultaneously used to generate 3D box proposals. Their center, size, and rotation are then estimated. A second stage might then be employed to refine the locations and orientations of the box proposals by pooling the points and their learned features according to each box proposal. This means that each point belonging to a specific box proposal is then used for its refining [23]. Using the learned features of a bounding box proposal, a robust object classification can be undertaken 4.3.

This approach is further discussed in the following sections 4.2 and 4.3, presenting the architectures of the backbone network, and the network used for bounding box proposal and refinement, respectively.

4.2 PointNet++ Architecture

The PointNet++ [14] neural network architecture builds on the success of the PointNet [2] architecture, which served as the pioneer of deep learning on unstructured point sets. It is further utilized — both in the original paper [23] and this thesis — as the backbone network for point-wise scene segmentation and learning point features, which are subsequently used by the stages of a PointRCNN neural network for 3D bounding box proposing and refinement, in this thesis finally used for people detection. The following introduction of both the PointNet and PointNet++ architectures is based on the original papers [2, 14].

The PointNet paper was the first to explore the feasibility of designing a deep learning network operating on raw 3D point clouds, without prior transformation to highly regular formats, more widely used for the tasks of scene understanding. That is unlike the previous works, where e.g., projecting the point cloud to a 2D view with subsequent utilization of a 2D CNN in [3, 19], or dividing the point cloud into regular 3D voxels and utilization of a 3D CNN [32], were used to avoid the problems introduced by the irregularity of point clouds. The authors of [2] emphasize three main properties of the point cloud format:

- The points are *unordered*;
- *the neighbouring points form meaningful subsets* of the scene, features of which are vital to the structure of the scene;
- and that the sets of points should be *invariant to tranformations*, like rotation and translation.

The basic idea of PointNet is to learn the spatial features of individual points and then aggregate the point features into a global point cloud signature. This way, however, the model fails to capture the local structures, unlike with CNNs, that are capable of capturing local features at multiple scales, thanks to the grid-like structure of their inputs. To this end, PointNet++ employs the PointNet model to extract local features from increasingly larger neighbors of points, capturing even fine geometric structures. This is done by partitioning the point cloud into overlapping local regions, based on the point distance, as seen in Fig. 4.2 *(set abstractions)*. PointNet model is then recursively applied to learn the local features, until the features of the whole point cloud are obtained.

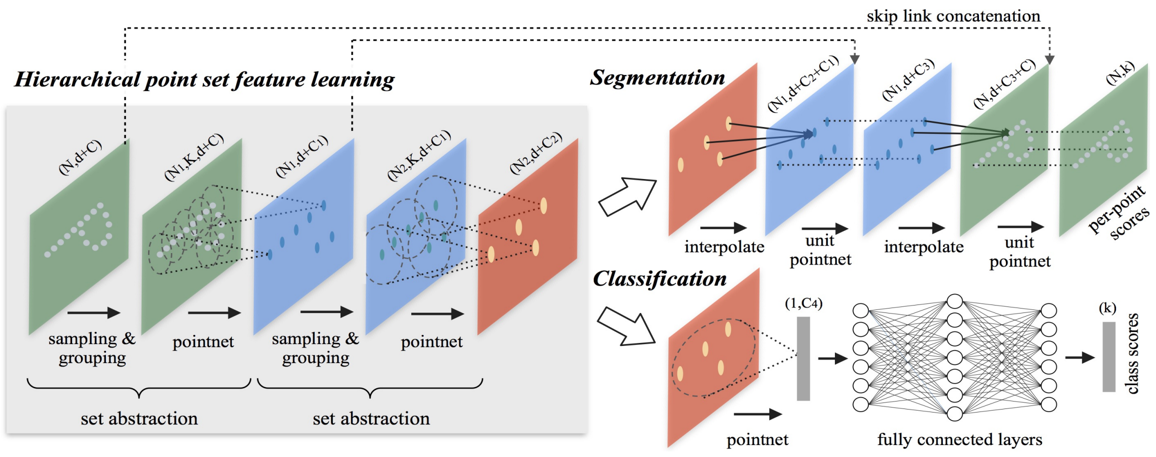


Figure 4.2: PointNet++ Architecture [14].

Unlike PointNet, which uses a single max-pooling operation to aggregate the features of the whole point cloud, the PointNet++ architecture introduces a hierarchical approach to feature learning, that would progressively abstract increasingly larger regions, as seen in Fig. 4.2 *(Hierarchical point set feature learning)*. The hierarchical structure consists of multiple **set abstraction levels**. Each such level is composed of a **sampling**, a **grouping** and a **PointNet** layer, as seen in Fig. 4.2. They operate as follows:

- The **sampling layer** uses *farthest point sampling*[3] to define the centroids of local regions.
- The **grouping layer** then constructs local regions around the centroids with variable number of points.
- Then a **mini-PointNet layer** is used to abstract the centroid and its local neighborhood.

Finally, to complete the task of point-wise segmentation, multiple feature interpolation levels are used to obtain point features for all the original points (see Fig. 4.2, part *Segmentation*). For interpolation, the inverse distance weighted average based on k-nearest neighbors is used. The interpolated features are also concatenated with skip linked point features from the set abstraction level. These features are then passed through a „unit PointNet", that is also used to calculate the per-point scores using *multilayer perceptrons*, once the features are propagated all the way back to the original set of points.

4.3 PointRCNN Architecture

The PointRCNN [23] framework is designed for 3D object detection and classification from raw point clouds. It aims to address the space for development provided by the success of the PointNet++ architecture in the task of scene segmentation, and to fill the void left in 3D object detection task created by the state-of-the-art methods, mentioned in the section 4.1, that seek to utilise the power behind state-of-the-art methods for 2D object detection, by transforming the point clouds beforehand. The framework is composed of two sub-networks that employ a scene segmentation backbone to segment the input point cloud, generate a small number of high-quality 3D bounding box proposals, that are then refined and used for class prediction. In this thesis, it is used for people detection and their bounding box estimation. The following introduction to the architecture of the PointRCNN framework is based on the original paper [23].

3D Bounding Box Proposal Generation

The first of the two sub-networks builds on the observations, that the objects in 3D scenes are separated naturally without overlapping, and that the ground truth bounding boxes can be used to obtain complete segmentation masks for each object's points — unlike with images. To generate only a small number of box proposals for the second — refinement stage, the authors of [23] utilize a backbone network to learn the point cloud features, then jointly segment the point cloud into the background and foreground points, and generate 3D box proposals. To this end, both the original paper [23] and this thesis employs the PointNet++. The following procedure is also visualized in the Fig. 4.3 section a).

The bounding boxes are only regressed from the foreground points, but all point features are used to generate the box proposals. A bounding box can be defined by its center location, size, and rotation — for our use case, the orientation from the bird's view will suffice. For estimating the center of an object and its rotation, the authors of [23] proposed employing bin-based regression losses, proven in [23] to localize the center locations more

[3]**Farthest Point Sampling** - is a sampling algorithm that iteratively samples the set of points farthest from the set of already points. Its advantage, when compared with other sampling algorithms is in better coverage over the input set of points.

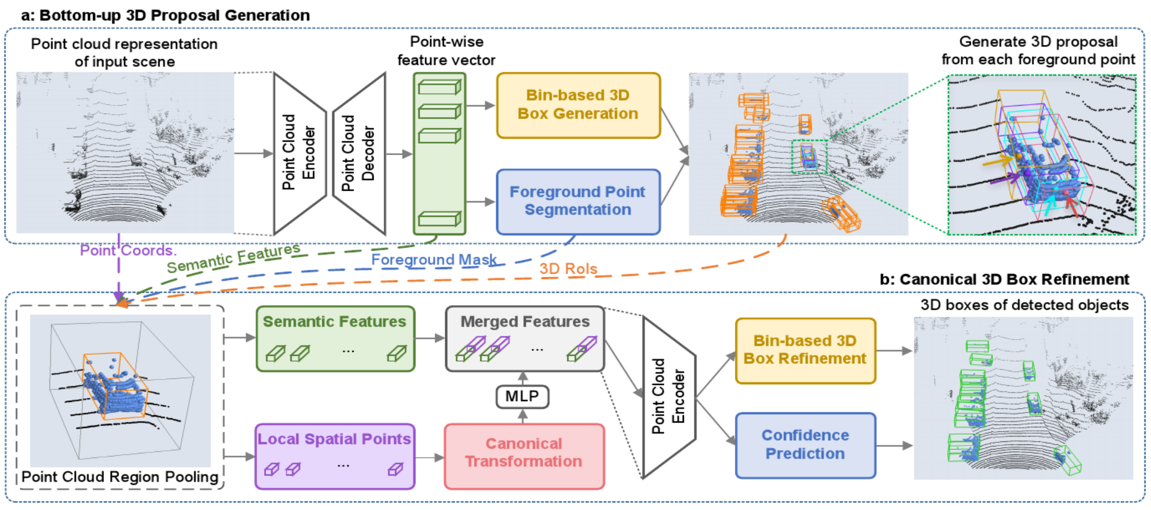

a: Bottom-up 3D Proposal Generation
Point cloud representation of input scene
Point-wise feature vector
Point Cloud Encoder
Point Cloud Decoder
Bin-based 3D Box Generation
Foreground Point Segmentation
Generate 3D proposal from each foreground point
Point Coords.
Semantic Features
Foreground Mask
3D RoIs
b: Canonical 3D Box Refinement
3D boxes of detected objects
Semantic Features
Merged Features
MLP
Local Spatial Points
Canonical Transformation
Point Cloud Region Pooling
Point Cloud Encoder
Bin-based 3D Box Refinement
Confidence Prediction

Figure 4.3: PointRCNN Architecture [23].

accurately and robustly, than by utilizing a smooth $L1$ loss[4]. The original paper [23] proposes to use a smooth $L1$ loss for the z-axis, as the vertical range, in which most of the objects are located, is very small. In this thesis, model with bin-based regression losses for all of the axes was utilized in the initial stages, but no significant difference was observed. The surrounding area of each foreground point is split into a set number of discrete bins. The search area for each axis is set by the parameter $\mathcal{S}$, that is divided into bins of uniform length δ. Then the equation for the localization targets can be formulated as the equation 4.1

$$\begin{aligned}\underset{u\in\{x,y,z\}}{\mathrm{bin}_{\mathrm{u}}^{(\mathrm{p})}} &= \left\lfloor \frac{u^p - u^{(p)} + \mathcal{S}}{\delta} \right\rfloor \\ \underset{u\in\{x,y,z\}}{\mathrm{res}_{\mathrm{u}}^{(\mathrm{p})}} &= \frac{1}{\mathcal{C}}\left(u^p - u^{(p)} + \mathcal{S} - \left(\mathrm{bin}_u^{(p)} \cdot \delta + \frac{\delta}{2}\right)\right)\end{aligned} \tag{4.1}$$

where $u^{(p)}$ represents the coordinates of a foreground point, u^p represents the center coordinates of the respective object, $\mathrm{bin}_u^{(p)}$ represents the ground-truth bin assignments for the respective axes, $\mathrm{res}_u^{(p)}$ represents the ground-truth residual for further location refinement within the assigned bin, and $\mathcal{C}$ is the bin length for normalization. For orientation θ of the target, the bin classification target $\mathrm{bin}_\theta^{(p)}$ and residual regression target $\mathrm{res}_\theta^{(p)}$ are calculated analogically to the center predictions, splitting the orientation 2π into n bins beforehand. The object size is directly regressed by calculating the respective residuals with respect to the average object size of each class in the entire training split. For the inference of the bin-based parameters (x, y, z, θ), the bin center with with the highest predicted confidence is chosen and the predicted residual is added to obtain the refined parameters. For the directly regressed parameters — the object size, the predicted residual is added to the initial values. To limit the number of proposals and avoid their redundancy,

[4]$L1$ **loss** is a loss function used to minimize the error of prediction, using the sum of the differences between the ground truth and the predicted values.

non-maximum suppression[5] based on the *Intersection over Union (IoU)* of the bounding box proposals from the bird's view, is used.

3D Bounding Box Refinement

To refine the bounding box proposals from the previous stage, first the points and their respective features are pooled accordingly to each box proposal. This is done by enlarging each box proposal to incorporate the additional information from its context, and performing an inside-or-outside test for each point. If a point is inside a box proposal, it is kept for the refinement of the proposal. Additionally to the point's coordinates, its predicted segmentation mask (background/foreground), learned feature representation and its distance to the sensor are included in its features. The proposals with no inside points are also eliminated.

Then the pooled points are fed into the second stage sub-network for refining the bounding box locations and foreground object confidence. This is done by firstly transforming the pooled points to the canonical coordinate system of its corresponding box proposal. Using such coordinate system enables the box refinement stage to learn better local spatial features for each proposal [23]. For each proposal, its inside points and their associated features are then fed to multiple fully-connected layers to encode their local features to the same dimension of the global features. Both the local and the global features are then concatenated and fed to a PointNet++ network to obtain a discriminative feature vector for the following confidence classification and box refinement [23]. Finally, the oriented NMS with IoU from the bird's view is again applied, now with a lower threshold to remove the overlapping bounding boxes.

[5]**Non-Maximum Supression (NMS)** is an algorithm used to select the top result from a set of overlapping entities using a simple criterium, such as the Intersection over Union.

Chapter 5

Dataset and Annotation

This chapter details the creation of the dataset used for the subsequent model training (further referred to as *the dataset*), the tool adapted for this purpose and the analysis of the dataset. The entire dataset is based on an approximately 1 hour 50 minutes long recording, that was recorded on a bike route in Prague, Czech Republic approximately between 15:35 and 17:25, on 6th of April, 2022; using Texas Instruments' radar module IWR6843ISK[1], show in Fig. 5.1. Its millimeter-wave radar sensor operates in range from 60 to 64 GHz. The module encorporates 4 receive and 3 transmit antennas with 120° azimuth field of view and 30° elevation field of view. The environment of the bike route is illustrated in Fig. 5.2. The configuration of the radar sensor was tuned with a compromise between high resolution and detecting faster cyclists in mind. Together with the recordings, source codes for the *millimeter-wave Data Analyser* application were provided by my supervisor, Ing. Lukáš Maršík. Using the provided `mmwTLVparser`[2] module to process the binary radar packets of the recording files, I implemented the script `rec_parser.py` for recording parsing and point cloud augmentation. The output point clouds were then annotated, using the video recording for reference.

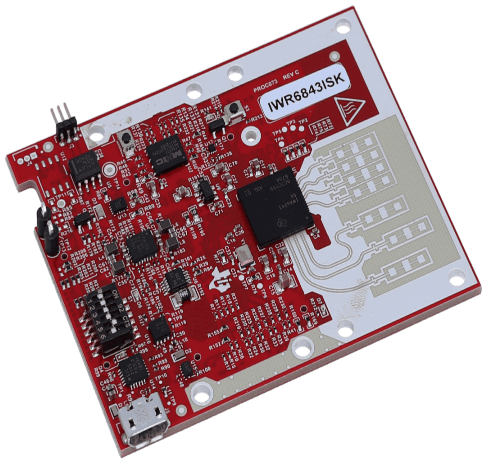


Figure 5.1: The IWR6843ISK radar module [17].

[1]https://www.ti.com/tool/IWR6843ISK

[2]**mmwTLVparser** is a module of the millimeter-wave Data Analyser application that implements the parsing of the radar packets.

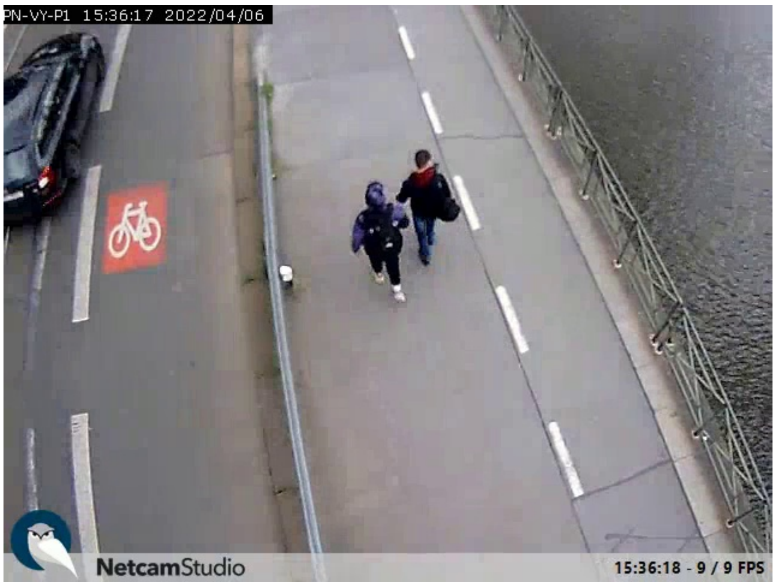


Figure 5.2: Bike route recording. Pedestrians and cyclists use the right two lanes, while the left two lanes are used by cars, and occasionally by cyclists. The radar sensor is positioned towards the right two lanes. The cars and bikers in the left two lanes are captured very sparsely, and are omitted from the dataset.

5.1 Data Processing and Augmentation

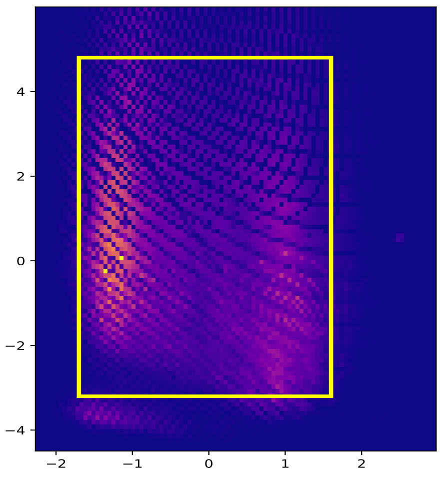
The `rec_parser.py` script implements the class `RecordingParser`, that is used to read the binary files produced by the radar sensor and loop over the packets they contain. Each packet is processed by the `mmwTLVparser`, according to the recording configuration specified by the provided configuration file. For each packet, the timestamp, the points and the packet's length are retrieved.

Table 5.1: Analysis of the recording used for creating the final dataset.

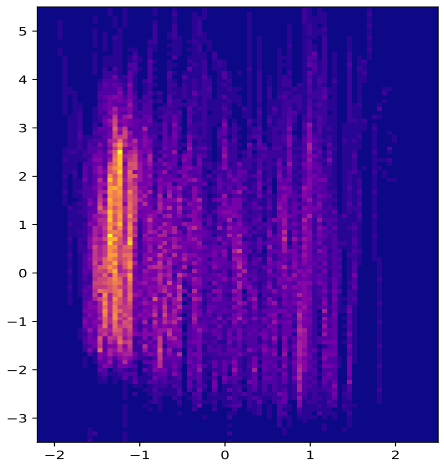
Recording Length	1h 50min
Frames	36,441
Total Points	1,544,078
Total Samples	19,629

Average Points	42
Median Points	27
Avg. Points per Person	58.3
Resampling Goal	**128**

The points from the neighboring frames are optionally grouped (the group size is set to 3 by default) using a sliding window algorithm, to increase the density of the point clouds. Further optional transformation includes rotating the point clouds and flipping them along an axis, to align them with the respective camera recording for a more intuitive annotation. For this particular dataset, the points have been filtered by the area, where only pedestrians and cyclists are reliably detected by the radar sensor, omitting the neighboring road (shown on the left side of Fig. 5.2) and other background points. The filtering area of the detected points is shown in Fig. 5.3a. The two vertical stripes in brighter yellow roughly correspond to the edges of the bike route, show on the right side of Fig. 5.2. Each point cloud is then resampled to a fixed number of points. I chose the resampling goal as 128, based on the average number of points per frame and average points per ground truth box, as seen in Tab. 5.1. The resampling is done as follows: (1) if the number of points in point



(a) Point heatmap and the filtering area. The points within the yellow rectangle are kept for further processing — approx. 83.5 % of the total points.



(b) Target heatmap showing most common tracks.

Figure 5.3: Point Cloud and Target Heatmaps.

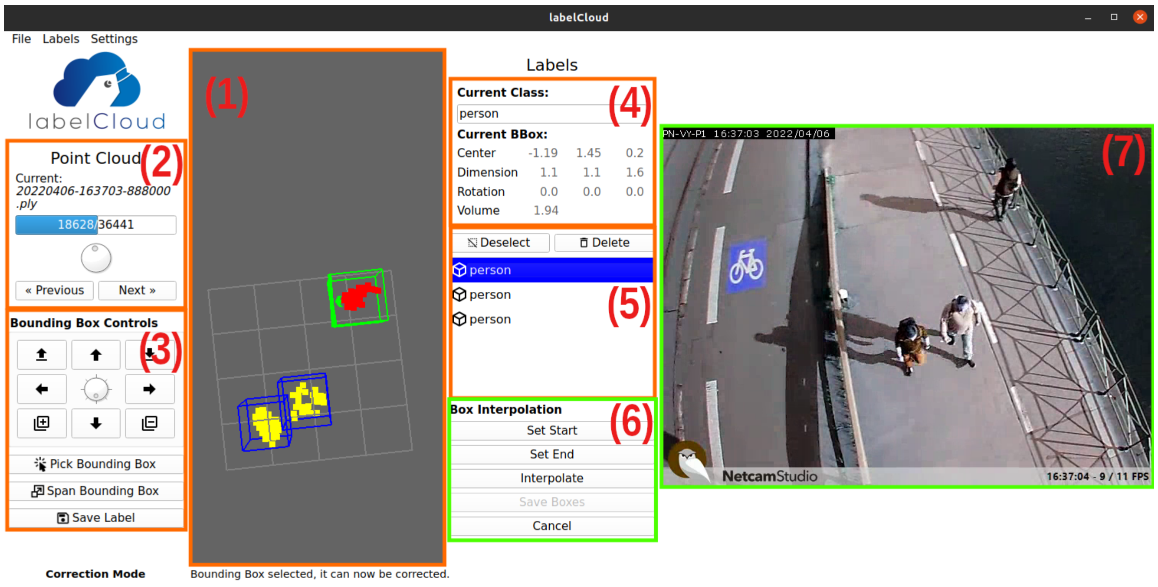
cloud is above n, randomly keep n points; (2) otherwise cluster the points using DBSCAN[3] clustering, and randomly repeat points in clusters only; where n is the resampling goal. The DBSCAN parameters were set as: $\varepsilon = 1$, $\text{minSamples} = 10$. If the clustering algorithm failed to produce any clusters, the frame is omitted. The points are finally saved in the PLY[4] format with a filename based on the corresponding timestamp.

5.2 Annotation Process and Labeling Tool

The aim of the annotation process is to assign each point cloud a list of ground truth boxes, defined by a class label, centroid (x, y, z), dimensions (l, w, h) and rotation from the bird's view θ. Since the radar point cloud doesn't provide as much detail as lidar point clouds, the annotation process requires camera image for reference. This can be achieved by fetching the video frame corresponding to the point cloud timestamp. The respective frame number can be calculated as the time delta between the video start time and the point cloud timestamp, divided by the video's frame rate. For this, the video's frame rate must be constant. The provided video with a variable frame rate was thus converted to constant frame rate video using the *ffmpeg*[5] tool.

To complete the annotation process, a tool is necessary for the tasks of visualizing point clouds, playing reference camera recording, bounding box inserting and manipulation, and basic recording flow controls. I have chosen the *labelCloud* tool for labeling 3D bounding boxes in point clouds. It is part of the PyPI[6], with its source code also made available at

[3]**Density-based spatial clustering of applications with noise (DBSCAN)**.
[4]**Polygon File Format (PLY)** is a simple file format designed for storing 3D data.
[5]https://ffmpeg.org/
[6]https://pypi.org/project/labelCloud/



Current: 20220406-163703-888000 .ply

18628/36441

Current Class: person

Current BBox:

Center	-1.19	1.45	0.2
Dimension	1.1	1.1	1.6
Rotation	0.0	0.0	0.0
Volume	1.94		

Correction Mode

Bounding Box selected, it can now be corrected.

Figure 5.4: Extended *labelCloud* annotation tool. Green boxes show the contribution of this thesis to the GUI. (1) Point cloud visualization, (2) flow controls, (3) boundig box controls, (4) label information, (5) list of labels, (6) interpolation controls, (7) reference video frame.

GitHub[7] under the GPL-3.0 license. This lightweight tool incorporates most of the basic functionality necessary for the proposed annotation process. It can easily be extended to fit the listed requirements, as the source code is made available, and the tool is implemented in Python and uses Qt for Python[8] for the GUI implementation. A complete list of changes to the *labelCloud* tool within this thesis can be found in the `patches` folder.

Using this tool, shown in Fig. 5.4, the annotation process is quite straightforward and efficient. As the empty and sparsely populated point clouds were not saved during the recording processing, the annotator's time isn't being wasted on skipping through empty frames. Using the dial widget in section (2) of Fig. 5.4, the annotator can quickly skip to the portion of the recording, where they left off. Then using shortcuts „`[`“ and „`]`“, jump a smaller number[9] of frames to find a recording segment, where either a pedestrian or a cyclist passes through the route. Then using shortcuts „`O`“ and „`P`“, find the precise frame, where the target person is first visible. Using the „`Pick Bounding Box`“ function in section (3) of Fig. 5.4, hover over the cluster with the mouse cursor and click to place the bounding box. Its position and size can be then fine-tuned using the arrow keys and `Page Up/Down` keys to change its position, „`X`“, „`Y`“ keys to change its rotation from the bird's view, or scroll wheel while hovering above a bounding box plane to change its dimensions. When done with the fine-tuning, set the interpolation start point to the currently selected bounding box by clicking the `Set Start` button in section (6). Repeat the process for the final frame, where the target person is visible, and click the `Set End` button, then `Interpolate` in section (6). This generates proposals for bounding boxes set by the position of the first and last appearance of the target person in the recording. This way, only two frames per each

[7]https://github.com/ch-sa/labelCloud/
[8]https://www.qt.io/qt-for-python
[9]The number of frames to be skipped was chosen to be about half of the frames it takes for an average cyclist to pass through the route.

target need to be annotated, with interpolation between the two frames generating the rest of the bounding boxes. After inspecting the interpolation outcome (visualized as orange bounding boxes with dashed lines), it can either be saved or canceled by clicking the `Save Boxes` or `Cancel` buttons in section (6) of Fig. 5.4, respectively.

5.3 Dataset Analysis

The final dataset consists of 19,629 samples of pedestrians and cyclists. The following analysis of its contents is based on the statistics in Tab. 5.2.

	Samples		Average Size		
Class	N	%	x	y	z
Pedestrian	17,678	90.1	1.1	1.1	1.6
Cyclist	1,951	9.9	1.1	2.7	1.6

Total Frames	16,266
Total Samples	19,629
Avg. Samples per Frame	1.21

Upsampling	Avg. Points per Target
Before	53
After	97

Table 5.2: Dataset Statistics

The number of samples produced from comparatively a short recording (1 hour 50 minutes) might be attributed to two factors: (1) (almost) each frame between the target's first and last appearances contains a valid sample, (2) the number of frames in which the target is valid (i.e. contains enough points) is increased by the grouping of frames described in Section 5.1. This also helps to reduce the blind spots of the radar sensor, shown as wave-like empty spaces in Fig. 5.3a. The average number of points per target after the resampling is nearly double the average for points before the resampling, showing that the clustering stage of the resampling process proposed in Section 5.1 makes an expected difference, enriching the point clouds of the valid targets. Fig. 5.5 shows, that the amount of empty frames from the majoritively empty recording has been greatly reduced. Their number is still significant when compared to the number of annotated non-empty frames, but this might be attributed to the occasional detection of targets „fading in the distance“,

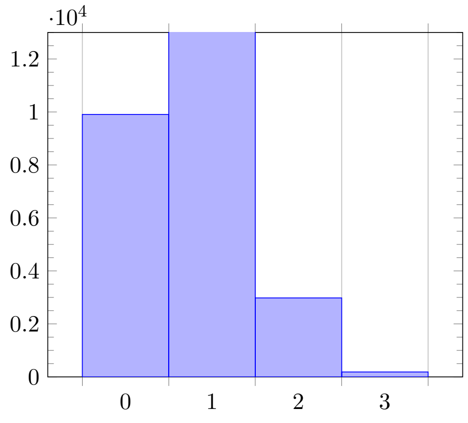


Figure 5.5: Number of Samples pre Frame.

which were omitted from this dataset, as their detection and amount of points per target are very inconsistent. This will require more attention in future work, as the scenes might be more complex and sensor-to-target distance increased. The tracks of targets run mostly along the edges of the bike route, shown on the right side in Fig. 5.2, as seen in Fig. 5.3b.

Chapter 6

Model Training and Evaluation

This chapter aims to build on the deep learning methods for object detection on point clouds discussed in Chapter 4, to design and implement a system that would operate on point clouds produced by a radar sensor, as described in Section 3.3, that would be trained on the dataset described in the previous chapter. First, the draft of the solution is proposed in Section 6.1. Then, the process of training the model is described in Section 6.2, followed by Section 6.3 that details the evaluation metrics and analysis of the model's performance. The subsequent deployment of the model for real-time inference and the people counting application that uses it are presented in Chapter 7.

6.1 Solution Draft

The model trained as a part of this thesis is to be used for people detection on point clouds produced by a radar sensor. Here, the people detection is to be understood as the task of locating a person within a region of interest, classifying the detection and estimating its bounding box in 3D. It is also followed by a tracking algorithm to separate individual people within the whole recording. The additional data produced by the tracking stage can then be used for specific use cases, such as people counting, occupancy monitoring and trajectory prediction.

I chose to build off of the PointRCNN architecture, introduced in Section 4.3, as it is designed to jointly address the tasks of bounding box proposing and detection classification. It consists of two stages that need to be trained separately — (1) Region Proposal Network (**RPN**), and (2) Region-based Convolutional Neural Network (**RCNN**). Same as in [23] I utilize a PointNet++ [14] backbone for point cloud segmentation, used to limit the number of bounding box proposals generated by the first sub-network. For this, I use the implementation [24] by the authors of [23], that further improves the PyTorch[1] implementation [31] of the original implementation [15] by the authors of [2, 14]. As the implementation of the PointRCNN architecture was intended for the task of single-class 3D object detection on the KITTI 3D Object Detection Evaluation 2017 dataset [5], I extended the implementation of the RCNN sub-network to allow for multiclass classification. To train on my own dataset, presented in Chapter 5, I build off of the existing scripts for the KITTI dataset, and implemented a custom pipeline for data preprocessing and loading. Finally, I used the evaluation framework of [24] to implement the evaluation process, as proposed in Section 6.3.

[1] `https://pytorch.org/`

For the model inference, I implemented scripts for both offline and online prediction generating. The offline version is designed to be run on a batch of data to produce a batch of predicted bounding boxes, together with segmented input points, that can then be used for offline visualization. The online version is to be hosted on a CUDA-capable device[2] for prediction generating, that can — depending on the connection speed to the host machine — be used for predicting in real time even on a non-CUDA-capable device. The model deployment is further discussed in Chapter 7.

For the task of instance segmentation, I decided to omit the use of a neural network [13, 27], as the PointRCNN network naturally separates the object instances by the predicted bounding boxes. Given the high quality of the refined bounding box proposals, I decided to use a combination of Kalman filter [9] and Hungarian algorithm [11], inspired by [20, 30], to track the movements of multiple detected objects. As discussed in Section 7.2, the tracker can be used to track individual detections through a time frame (i.e. through a series of point clouds), allowing for a straightforward implementation of e.g., people counting application, as discussed in Section 7.4.

6.2 Model Training

The model was trained on the training split of the dataset discussed in Chapter 5. The training split is roughly 85 % of the total dataset, with 13,967 point clouds and 16,568 labels. The first stages of the training process were conducted in the Google Colab environment[3], but due to the strict limitations on the GPU usage, the training was soon migrated to a virtual machine on the Google Cloud Platform[4]. Here, the training, evaluation, and deployment of the model for inference in real time (see Chapter 7), were conducted on a single NVIDIA Tesla K80 GPU using the CUDA toolkit.

The two sub-networks of the PointRCNN model are trained separately. First, the RPN stage needs to be trained for generating small number of high quality proposals. This is done by first segmenting the point cloud into the foreground and background points, using a PointNet++ backbone. The evaluation of the backbone network is omitted from the following Section 6.3 on model evaluation, as its performance is in tandem with the performance of the RPN sub-network. Here I will only state that the point segmentation accuracy drops significantly on objects near the edges — the start and the end, of the recorded portion of the bike route. This might be attributed to the fact that the ground boxes in those areas are annotated sparsely, as a decent portion of the detections located there is nearly indistinguishable from the noise generated e.g., by cars on the nearby road — and thus were sometimes omitted. The only difference to the configuration of the set abstraction levels of the PointNet++ backbone (see Section 4.2) introduced in this thesis is the smaller number of sampled points and slightly reduced radii of the local regions, according to the average size of the ground truth labels of the dataset discussed in Chapter 5.

For the training of the RPN sub-network, a combination of a higher batch size and a larger number of epochs seemed to converge the fastest. The batch size used to train the final version of the RPN stage was set to 256, and the number of epochs to 500, after which all intriguing evaluation metrics seemed to plateau (see Section 6.3). After each configuration change and subsequent retraining, the results were evaluated both visually and using the metrics discussed in Section 6.3. When both high precision and recall were

[2]CUDA® is a parallel computing platform developed by NVIDIA for computing on GPUs.
[3]https://colab.research.google.com/
[4]https://cloud.google.com/

achieved, and the visual results were deemed satisfying, the model was used to train the RCNN sub-network.

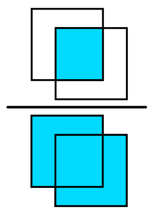
Of the bounding box proposals generated by the RPN stage, the top 100 proposals with the highest confidence are kept, and further reduced to 10 proposals using non maximum suppression with an IoU threshold of 0.3. Similar to the config of the RPN sub-network training, the number of points and the radii of the set abstraction levels have been slightly reduced to account for the sparsity of the radar point clouds and the relative proximity of related points. Further reconfiguration includes using Cross-entropy loss with weights of 0.05, 0.35, and 0.6 for background proposals, and pedestrian and cyclist bounding boxes, respectively. Assigning a larger weight to the underrepresented cyclist class has also been tested, but the only noticeable difference in the output has been the higher confidence in the misclassification of pedestrians pushing trolleys as cyclists.

6.3 Model Evaluation

In this section I propose the evaluation methods for the bounding box proposals of the RPN sub-network and the proposal refinement and classification of the RCNN sub-network. Evaluations of the subsequent tracking algorithm and people counting application are described in Chapter 7. The evaluations described in this section were conducted on the testing split of the dataset, thus evaluating the model's performance on a set of unseen data. Below is an introduction to the methods commonly used for the performance evaluation of object detection and bounding box prediction algorithms and models.

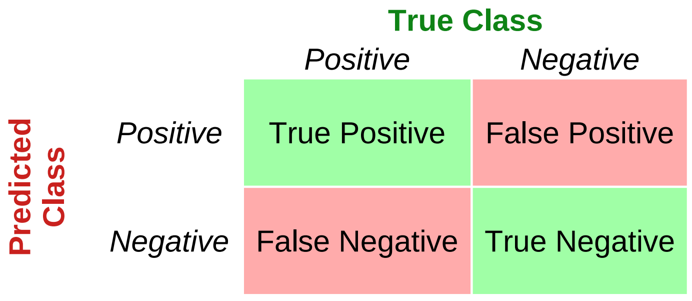
Intersection over Union (IoU) is an evaluation metric commonly used for object detection tasks to describe the overlap between a ground truth bounding box and corresponding prediction. It is calculated as the area of intersection over the area of the union of the bounding boxes, as shown in Eq. 6.1. The IoU values range from 0 to 1, meaning no overlap at all and complete overlap, respectively. An IoU threshold can be set to classify a detection as a true positive if the IoU is above the set threshold, and false positive otherwise.

$$[t]IoU = \frac{area(gt \cap pred)}{area(gt \cup pred)} = \tag{6.1}$$



Confusion Matrix is a matrix comparing the actual truth values and the corresponding predictions. As shown in Fig. 6.1, a confusion matrix for binary classification is composed of four values:

- *True Positive* means that the predicted class matches the ground truth object.
- *False Positive* means that the predicted class doesn't match the ground truth object.
- *True Negative* means that the model correctly predicted the negative class, where the ground truth object was also missing.



Predicted Class \ True Class	Positive	Negative
Positive	True Positive	False Positive
Negative	False Negative	True Negative

Figure 6.1: Confusion matrix for binary classification.

- *False Negative* means that the model failed to produce a prediction, where the ground truth object was defined.

Recall describes the portion of the ground truth positives, that were identified correctly by the model. It is calculated as the number of true positives over the sum of the true positives and false negatives.

Precision describes the proportion of the correct predictions to all the predictions made. It is calculated as the number of true positives over the sum of the true positives and false positives.

Mean Average Precision (mAP) is one of the most common metrics for the evaluation of object detection and segmentation algorithms. It is calculated as the mean of precisions at each IoU threshold. **Average Precision (AP)** at a single IoU threshold is also often presented as the main evaluation metric for object detection, where the average precision at e.g., IoU threshold 0.5 would be written as *AP@0.5*.

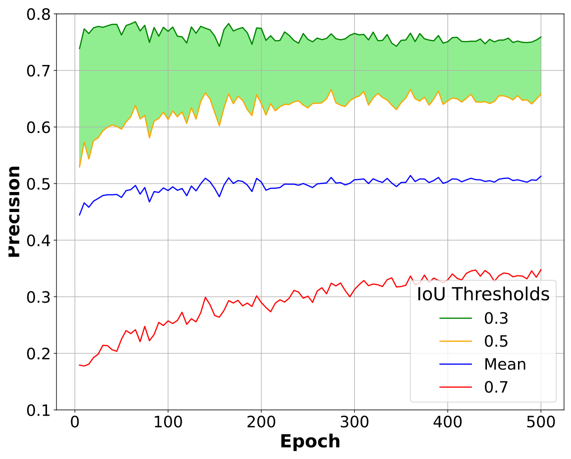
F-score is used to measure the accuracy of a model, when seeking to balance between precision and recall. It is calculated as Eq. 6.2.

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{6.2}$$

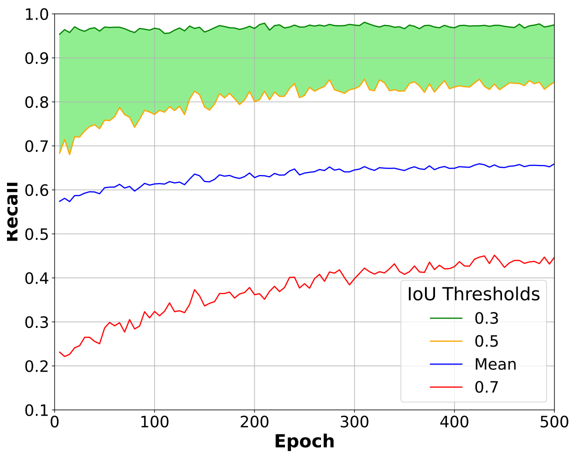
RPN Sub-Network Evaluation

The first of the two sub-networks that compose the PointRCNN architecture is designed to produce a *small number* of *high quality* bounding box proposals. To evaluate the accuracy of these two statements, we take a look at the precision and recall of the generated proposals.

As seen in Fig. 6.2, both the precision and recall start to plateau after epoch 400. The only increase for the latter epochs is observed for the IoU threshold of 0.7, which represents a near perfect overlap for two objects of the same size. The lower thresholds, 0.3–0.5 correspond to approx. 50–70% overlap. As the top and bottom walls of the bounding box proposals are not as important for the people detection task within the context of this thesis, this range is perfectly fine for the proposal stage of the final model, resulting in both high precision and recall (seen as a green area in Figs. 6.2a and 6.2b).



(a) History of precision of the RPN stage.



(b) History of recall of the RPN stage.

Figure 6.2: History of the RPN sub-network training for different IoU thresholds.

RCNN Sub-Network Evaluation

The second sub-network of the PointRCNN architecture is tasked with the refinement of the bounding box proposals, and their classification as one of the two people classes — pedestrian and cyclist, or discarding them as a `none` class. The proposal is also discarded in case that neither of the class scores is dominant enough, implemented using a softmax function. The boxes with normalized scores above a certain threshold (set to 0.75 for the final model) are kept and further refined using a non-maximum suppression to eliminate the overlapping boxes (by removing the boxes with non-maximum confidence score that overlap with the proposal with maximum confidence beyond a specified IoU threshold). The kept bounding box proposals can then be compared with the ground truth.

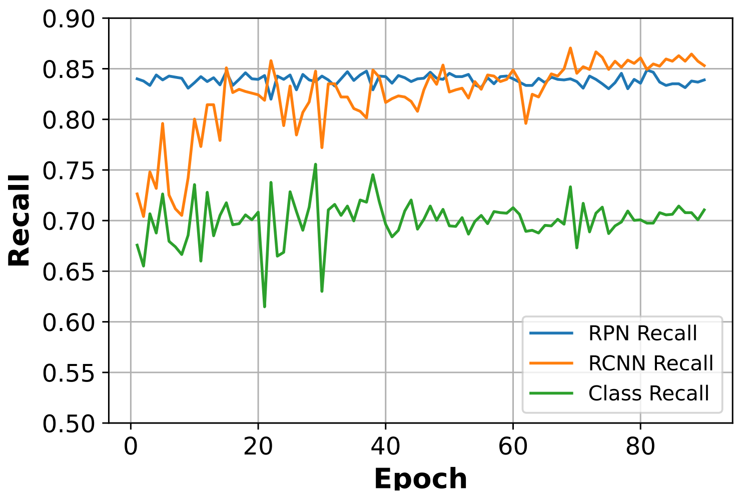
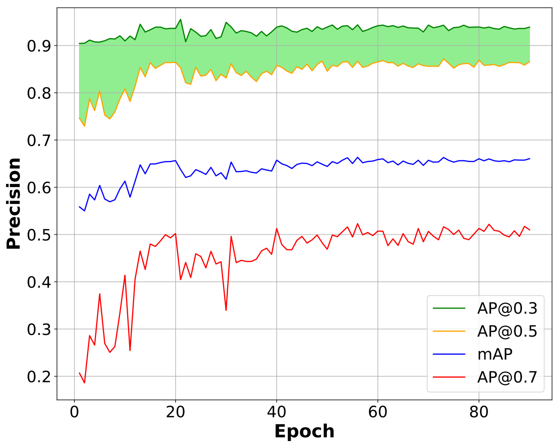
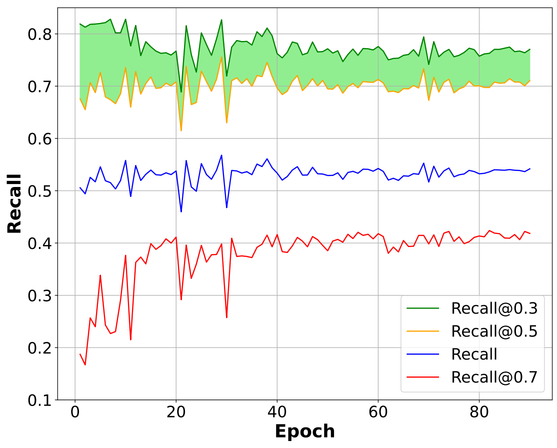


Figure 6.3: History of RCNN classified and unclassified recalls.

After about 70 epochs, the recall of the bounding boxes refined by the RCNN sub-network starts to plateau slightly above the RPN recall, as shown in Fig. 6.3 for the IoU threshold of 0.5. For comparison, the *classified recall* calculated only from the correctly classified predictions follows some 15% below the unclassified recall. The comparison for different IoU thresholds of both precision and recall is shown in Figs. 6.4a and 6.4b, respectively. Even for the higher IoU threshold of 0.7, the performance starts to plateau after the 60th epoch. Analogically to the RPN stage (see Fig. 6.2), for the IoU threshold range of 0.3 and 0.5 (green area in Figs. 6.4a and 6.4b), the model achieves precision of approx. 85–95 % and classified recall over 70 %. Additionally, just above 40 % of all ground truth bounding boxes are recalled with near perfection (at IoU threshold of 0.7). This performance evaluation also encompasses the high classification precision, as the presented precision and recall values are considered valid only for the predictions that were classified correctly.



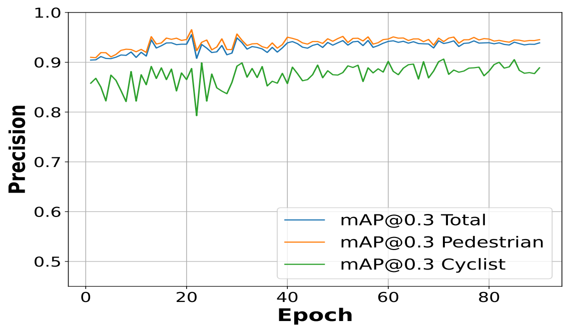
(a) History of precision of the RCNN sub-network.



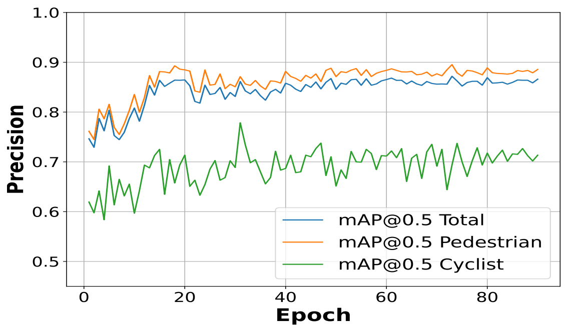
(b) History of recall of the RCNN sub-network.

Figure 6.4: History of the RCNN sub-network training for different IoU thresholds.

Looking at the per class evaluations in Fig. 6.5a, we see that the precision of classifying cyclists really benefits from a lower IoU threshold, compared to a 0.5 threshold in Fig. 6.5b. This might be partially attributed to the fact that the length of the bounding boxes of cyclists varies according to the cyclist's speed, as multiple frames are grouped together to produce richer point clouds (see Section 5.1).



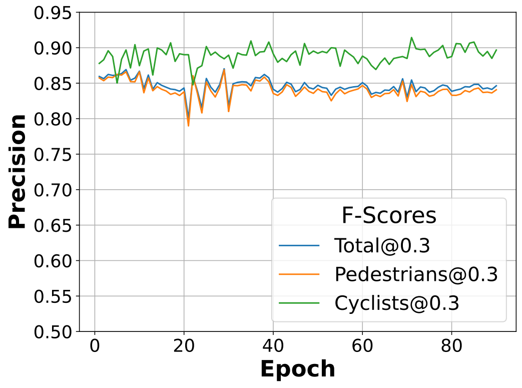
(a) History of mean average precision at IoU threshold of 0.3.



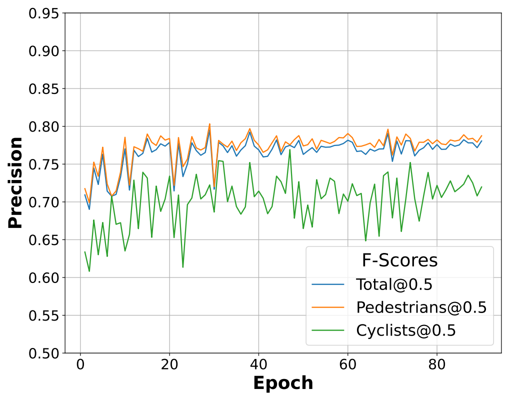
(b) History of mean average precision at IoU threshold of 0.5.

Figure 6.5: Class-wise precision comparison.

Lowering the IoU threshold also significantly boosts the recall of cyclists, resulting in an above the average F-score, as shown in Fig. 6.6a, in comparison to the F-score for a higher IoU threshold, shown in Fig. 6.6b. With average precision almost at 90 % even for the IoU threshold 0.5 (see Fig 6.5b), higher recall might also contribute to better results of e.g., a people counting application, that might consider multiple frames before deciding the object's class.



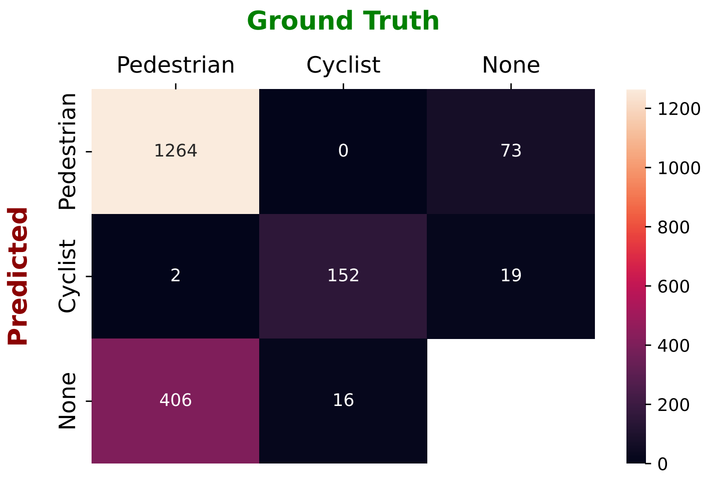
(a) History of the F-score at IoU threshold 0.3.



(b) History of the F-score at IoU threshold 0.5.

Figure 6.6: Class-wise F-score comparison.

The classification accuracy of the model can also be deduced from the confusion matrix in Fig. 6.7. Here, the true positives for both the pedestrians and the cyclists are the dominant outcome. Next significant portion is comprised of the false negatives, that were predicted at a much higher rate for pedestrians than for cyclists. This might be attributed to the fact that the model's confidence in predictions on individual pedestrians in groups is fairly low for approx. 20 % of the area of interest on the very edges. Nevertheless, the people counting accuracy, as discussed in Chapter 7, remains mostly unaffected, as even most pedestrians walking in groups are separable in majority of the area of interest.



Predicted \ Ground Truth	Pedestrian	Cyclist	None
Pedestrian	1264	0	73
Cyclist	2	152	19
None	406	16	

Figure 6.7: The final confusion matrix.

Chapter 7

People Counting

The model trained as a part of this thesis and the underlying dataset were produced with broad applicability in the field of scene understanding for the task of object detection in mind. On the other hand, many design and implementation choices for both the dataset and the model aimed at improving the performance, accuracy, and robustness, have rendered the produced system more and more specialized for the task of people detection and classification. To showcase the robust applicability and the straightforwardness in deploying the system discussed in the previous chapters and developed as a part of this thesis, I decided to focus on one of the many influential applications of modern millimeter-wave radar systems — people counting. It requires not only robustness even under severe weather conditions, but also high accuracy and high performance for measuring in real time. This chapter details the deployment of the model described in Chapter 6 for real-time inference in Section 7.1; and describes the tracking algorithm used for distinguishing the detections through sequences of frames, and further improvements to the object classification, in Section 7.2. Then, the employed methods are evaluated and discussed in Section 7.3. The application designed to visualize the real-time bounding box prediction, tracking, and classification results, and to showcase its real-time people counting capabilities, is described in Section 7.4. Other possible applications of the system detailed here are discussed in Chapter 8.

7.1 Model Deployment

To allow for real-time inference, required for the task of people counting in real time, the model was deployed on a virtual machine hosted on the Google Cloud Platform. For inference, a single NVIDIA Tesla K80 GPU is used, just as for the training process detailed in Section 6.2. The connection is handled using persistent WebSockets, and the Hypercorn[1] web server is used to deploy the model.

The visualizer application (further referred to as *the visualizer*), described in Section 7.4, incorporates a separate plot for visualizing the real-time predictions. An example of implementation of the online prediction calls would be to batch the generated point clouds in a fixed time interval, and send them to the remote machine to generate the predictions, which are then written to the corresponding frame indices just-in-time before they are required for visualization. In the visualizer, the process running in a separate thread tasked

[1]https://pgjones.gitlab.io/hypercorn/

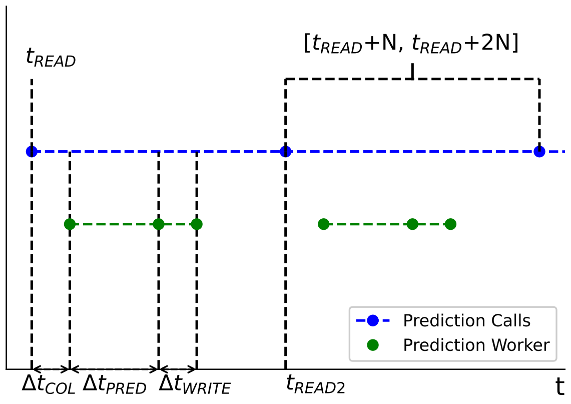


Figure 7.1: Online prediction time frame. Shows the action flow of a single prediction call.

with handling the prediction calls and writes is further referred to as *the prediction worker*. The online predictions operate along these steps, as shown in Fig. 7.1:

- When the video player notifies the application on frame update, the handle function of the real-time plot checks, whether a fixed interval (set to roughly one second by default) has passed from the last prediction update. In that case, referring to the current frame time as t_{READ}, the point clouds are collected from the time frame of $[t_{READ} + N, t_{READ} + 2N]$, where N denotes the fixed time interval of prediction calls. To avoid redundancy of predictions in case of rewinding the recording, only the point clouds missing from the list of online predictions are collected.
- After that, the prediction worker sends the batch of point clouds to the model hosted for inference. The time between the t_{READ} and the time the batch is received by the model is referred to as Δt_{COL} (`COL` for *collecting*).
- The prediction period is referred to as Δt_{PRED}, after which the predictions are sent to the prediction worker, and written to a list[2] of online predictions. The writing period is referred to as Δt_{WRITE}.
- The predictions from t_{READ} are recalled at time $t_{READ} + N$. Concurrently, the prediction call for the time interval $[t_{READ} + 2N, t_{READ} + 3N]$ is also made at time t_{READ2}, where $t_{READ2} = t_{READ} + N$.

Using a fixed interval of one second, the latency of the deployed model per a batch of point clouds is shown in Fig. 7.2. The prediction calls were made using the visualizer (see Section 7.4) on a real-time replay of the recording discussed in Chapter 5.

7.2 Tracking

The tracking system implemented as a part of this thesis uses the 3D bounding boxes generated by the trained PointRCNN model. Each such box is defined by its center coordinates

[2]In fact implemented as a Python dictionary, with frame indices for keys and predicted bounding boxes as values.

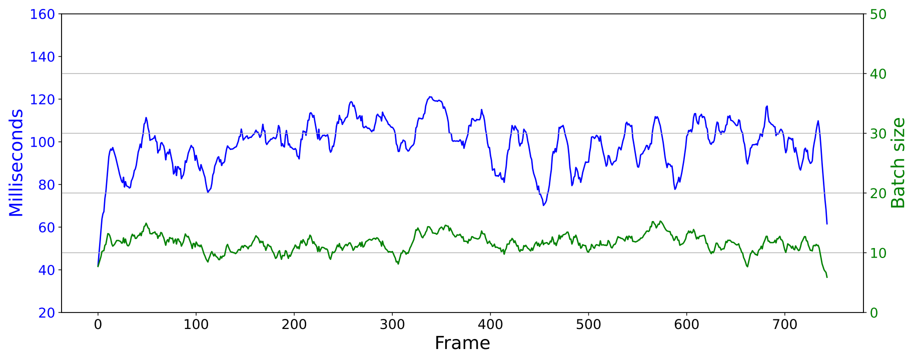
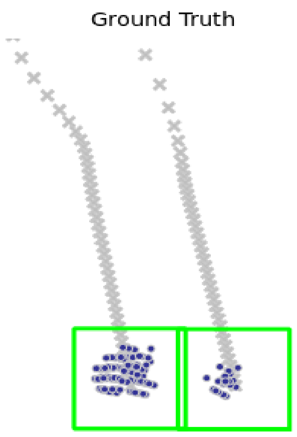


Figure 7.2: Inference latency and corresponding batch size.

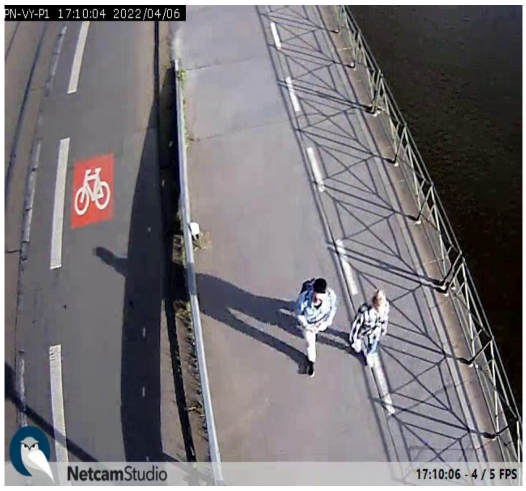
(x, y, z), size (l, w, h) and orientation from the bird's view θ. [30] proposes to formulate the state of an object as a vector of its central location, size, rotation, and velocity in 3D space. As the z center coordinate of the bounding boxes is in a very small range (due to the nature of the recording and objects of interest), and the visualizer application discussed in Section 7.4 plots the data in 2D only, the tracking system discussed here omits the 3rd dimension of the bounding boxes. To account for the rare cases of a single object being assigned different class labels through a set of frames (e.g., a pedestrian pushing a stroller being labeled correctly as a pedestrian in one frame, and as a cyclist together with the stroller in another frame), box size has also been omitted, to allow for the rare association of objects labeled as different classes. A soft class voting has been adopted instead. It is implemented by keeping a list of predicted labels and corresponding confidences for each tracked object. On each update call, the tracker returns the class with the greatest weighted average confidence. Bounding box orientation has also been omitted, as the orientation from the bird's view of a pedestrian is unreliable, probably due to the symmetric shape of the corresponding set of points. This has not been observed to cause any significant decrease in the tracking accuracy, possibly due to the fact that the bike route (see Fig. 5.2) is traversed along the same straight paths, seen as vertical lines in Fig. 5.3b.

I have decided to define a tracked object as 4-dimensional vector (x, y, v_x, v_y), where x and y are the center coordinates and v_x, v_y represent object velocity in 2D space from the bird's view. For each frame, the state of the trajectories is predicted from the previous frame. Then, to associate the predictions with detections for the current frame, an association cost is calculated from the Euclidean distance of the predicted and the detected centers. The assignment is then completed by assigning the pairs with minimal assignment cost. This is done using a Hungarian method — implemented as the `linear_sum_assignment` function of the SciPy[3] Optimize module. For each track that wasn't assigned a prediction in the current frame, a „skipped frame“ counter is increased. If it reaches a specified threshold, the track is deleted. The new detections that weren't assigned a predicted track are appended as new tracks. An example of the tracking process on radar point clouds is shown in Fig. 7.3. The maximum cost valid for assignment is also limited by a threshold set to approx. 1.5 times the length of an average cyclist bounding box. Other empirically configured thresholds include the minimum number of frames before a tracked object is counted

[3]https://scipy.org/



(a) The ground truth bounding boxes (green) and their respective trailing tracks (grey).



(b) Reference video frame.

Figure 7.3: Tracking on ground truth bounding boxes. Made using the visualizer application (see Section 7.4).

as a valid person detection and a maximum number of frames that a tracked object can be missing in, before being forgotten by the tracker.

7.3 Evaluation

The scope of the evaluation of the people counting application proposed in this thesis is limited by the length of the recording used to produce the final dataset. Still, a decent understanding of the precision of the system can be achieved by comparing the progress of the people counting on generated predictions, to the ground truth of the referential video and radar detections. To monitor the progress of the people counting output, a manual counting of pedestrians and cyclists was conducted, using the labelCloud application (see Section 5.2), already repurposed for the annotation of radar point clouds. To this end, a simple two-button interface was temporarily added to the labelCloud application. Using the shortcuts already described in Section 5.2, the annotator plays through the recorded radar detections and pushes two keyboard shortcuts (*minus* and *equal* keys by default) to increase the respective counters for pedestrians and cyclists. The serialized annotations are exported as a single file in the *pickle (PKL)* format, consisting of point cloud indices and counts per class of people counted to up that frame. The following evaluations were run on the whole recording, including the train split of the dataset used to train the model used to produce the predictions for these evaluations.

For the class-wise people counting, a robust classification outputs are necessary. The confusion matrix produced from the whole recording, as shown in Fig. 7.4, shows that the misclassification of pedestrians and cyclists occurs very rarely. The false negatives, introduced rather by the failure to recall every ground truth bounding box, than misclassification of a bounding box prediction as empty, might be partially attributed to the ground truth labels within the approx. 20 % of the area of interest — the start and end of the bike route in scope. Detections in these areas were also annotated, but the corresponding predictions have low confidence, as the point clouds from these areas are very sparse and (even after the

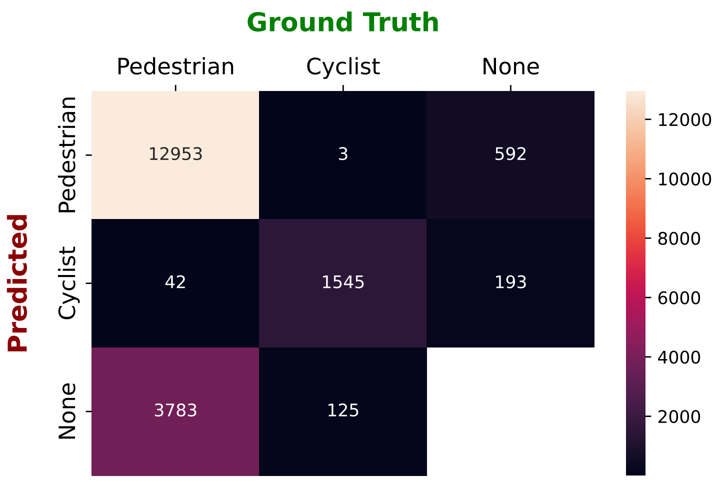


Figure 7.4: Confusion matrix for the whole dataset.

frame grouping described in Section 5.1) the objects detected here do not appear in each consequent frame. This, however, does not seem to affect the people counting performance in any manner, as all the predictions in a sequence of frames are considered, and the central 80 % of the area of interest accounts for the absolute majority of them.

Predictions on the whole dataset were then generated, and frame by frame run through the tracking algorithm described in the previous section. The final results of this process, compared to the manually annotated ground truth, are shown in table 7.1. Here we see that the difference, even for a dataset of smaller size, is minimal. A portion of this error (below 3 % of the total size) might also be attributed to mistakes in the annotation process, but with the total dataset size on the lower hundreds of individual pedestrians and cyclists, it is difficult to estimate the performance on unseen data with any higher certainty.

The progress of people counting using the generated predictions and the tracking algorithm, versus the ground truth, is shown in Fig. 7.5. Here we see that except for singular miscounts in individual frame sequences, the predicted counts follow the ground truth with near perfection. This result was achieved by tuning the configuration of the tracking algorithm according to the observed tendencies of the people within the recording and the performance of the model. Below is a couple of notes from the edge-case scenario evaluation of the people counting performance.

Groups of pedestrians are separated rather accurately, up to a number of four people, where the predicted count is usually 2–3 pedestrians. The undercounting in this situation might be improved by lowering the threshold to separate foreground predictions from the

	Ground Truth	**Predicted**	**Difference**
Pedestrians	297	289	**8**
Cyclists	160	157	**3**
Total	457	446	**11**

Table 7.1: Results of the people counting using predicted bounding boxes and tracking algorithm, versus manual counting.

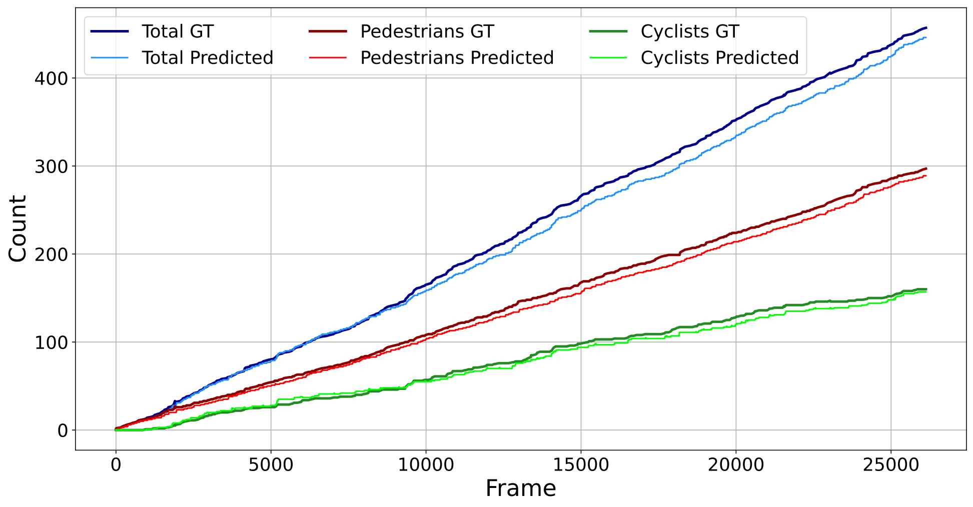


Figure 7.5: People counting progress. The predicted counts for the combination of model predictions and the tracking algorithm are compared to the ground truth.

background for the center of the area of interest, where noise and false positives occur very sparsely. In the edge 20 % of the area of interest discussed above, pedestrians walking in pairs are difficult to separate, which results in low confidence score. This, however, doesn't seem to affect the people counting performance, as they are confidently separated for the remaining 80 % of the corresponding frame sequence.

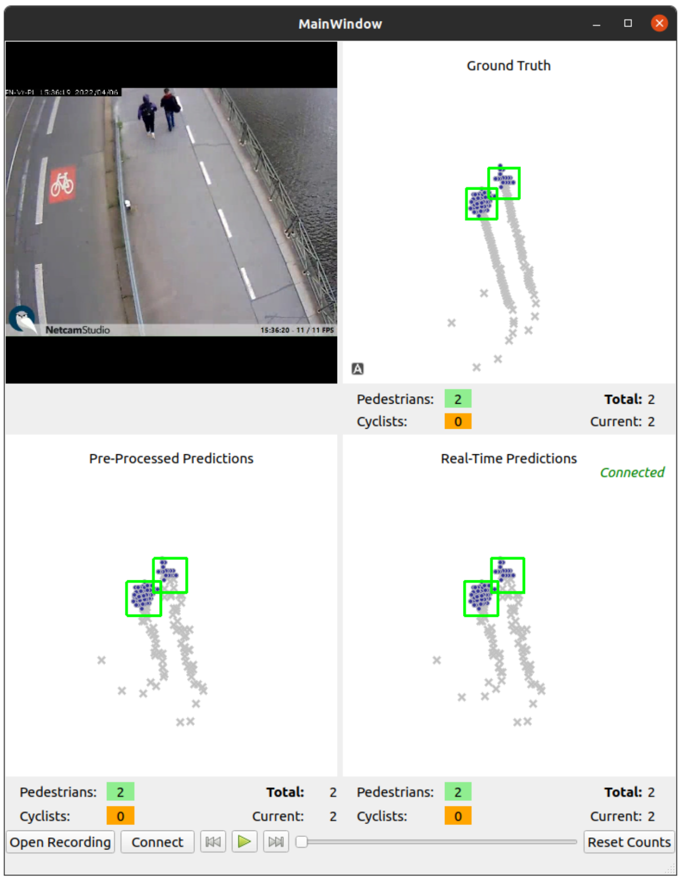
Smaller children walking alongside parents are sometimes shielded from detection (observed in approx. 2 cases), when walking on the right side of the parent (from the camera's point of view). A solution to this case of false negative prediction might require a larger dataset for a more in-depth study, but a straightforward approach might consist of repositioning the radar sensor, or using two and more sensors on different sides of the road, to produce the combined point cloud, where no such shielding by a larger person would occur.

Runners pushing strollers are at times miscounted as cyclists. To seek a robust solution to this rare misclassification (observed in approx. 3 cases on the whole recording), a larger dataset would be be required.

Faster cyclists (less than 5 %) sometimes fail to be counted as valid detection, as they appear in very few frames, even as they are classified with near perfection. Solution to this might be in adapting the minimal number of frames for a tracked detection to be considered valid, to the velocity of the object. Lowering this threshold for cyclists (as compared to the threshold for pedestrians, which is approx. three times higher) has resulted in a decent improvement of performance, but using dynamic thresholding might eliminate this deficiency completely.

7.4 Visualizer

This section details the application developed to showcase the outputs of the model trained for people detection and classification, and the tracking algorithm proposed in the previous section. To allow for the visual evaluation of the predictions, the application needs to visualize the processed point clouds, predicted bounding boxes, object tracks, and the referential video. The video also needs to be synchronized with the point cloud frames and corresponding bounding boxes. In the context of this thesis, the nature of the recordings used to produce the final dataset, and the specific task of this thesis being the people detection, I decided to implement a people counting application, to showcase the straightforwardness of the adaptation of the methods proposed herein, for a real-world scenario. The draft of the application builds off of the basic visualization requirements listed above, accounts for the nature of generating the bounding box predictions, and aims to produce a reliable statistical output, that can also be evaluated in real-time by an overseer. To this end, I decided to use the multi-platform GUI toolkit PyQt[4] for the Python programming language, to implement the front end of the application, and to handle the basic threading tasks for plotting and prediction calls.



Pedestrians: 2 Total: 2
Cyclists: 0 Current: 2

Pedestrians: 2 Total: 2
Cyclists: 0 Current: 2

Pedestrians: 2 Total: 2
Cyclists: 0 Current: 2

Figure 7.6: The visualizer application. From top left clock-wise: (1) referential video, (2) ground truth plot, (3) online predictions, (4) offline predictions.

[4]https://www.qt.io/qt-for-python

The main window of the application, as seen in Fig. 7.6, consists of four separate areas — one for the referential video, and three for plotting. To implement the plotting widgets, PyQtGraph[5] graphics and GUI library built on the PyQt has been used. Together with basic graph plotting framework, it also allows for zooming, moving the plots within the plot window, and exporting the plots. Each plot consists of three basic elements: (1) points of the point cloud corresponding to the video frame, (2) bounding boxes of color corresponding to the object's class, and (3) gray trails, visualizing the track of a detected object. Below each plot is a set of four counters: on the left — one for each of the object classes (of colors matching that of the bounding boxes), and on the right to track the total number of detected objects and the number of objects in the current frame. On the bottom of the main window is a control bar. The basic controls it incorporates include opening the recording directory, connecting to a web server for real-time model inference, video controls and a button to zero the counters.

The bounding box predictions need to be provided either by generating the predictions by the `predict.py` script and moving the output file to the recording folder, or by connecting to the model hosted for inference using the `online_predict.py` script. The connection is handled using persistent WebSockets. By batching the data and handling the predictions in set interval, the predictions can be generated in real-time, and visualized concurrently with the video player. The visualization process consists of the following steps:

- The video player notifies the application on frame change in set intervals.
- If the current time of the video exceeds that of the current point clouds, new point cloud file is loaded. The video time is calculated as the sum of the starting timestamp in the video's filename, and the current frame number divided by the video frame rate. In the case of jumping a large number of frames (e.g., while rewinding), bisection method is used to find the corresponding point cloud frame index.
- On point cloud frame change, the plots are notified to update with the current point cloud. The plot widgets also receive the corresponding bounding box lists, either loaded from a file (ground truth and offline predictions), or are set by the corresponding batch of online predictions. The process of online prediction is further detailed in Section 7.1.
- The plot widgets further use the provided bounding boxes to update the respective trackers — one for each active plot. Each tracker then updates its inner state and the list of tracks, as discussed in Section 7.2. It finally returns the class-wise, total and current object counts, used to update the GUI counters. The track records are also used to plot the trail of a detected object, shown as gray trails in Fig. 7.3.

The folder containing the recording data is to have the following structure:

```
recording_dir
├── labels
│   └── ...labels as *.JSON...
├── pcds
│   └── ...point clouds as *.PLY...
├── <predictions>.pkl
└── video
    └── <start_timestamp>.mp4
```

[5] https://www.pyqtgraph.org/

Chapter 8

Conclusion

The goals set within this thesis were to study and evaluate the state of the art of sensors and methods used for object detection from 3D scenes, to select a method based on neural networks, suitable for object detection on 3D point clouds generated by a millimeter-wave frequency-modulated continuous-wave radar based on the radar-on-chip platform; to produce a dataset for the training of the neural network; evaluate the performance of the trained neural network; and to implement and evaluate an application that utilizes the trained model to solve a real-world task, here selected to be people counting. The model, and the system that utilizes it, are to be robust and accurate enough for the task of people counting to prevent missed detections, overcount, and misclassification, and at the same time allow for a real-time inference.

The content of this thesis is segmented into the study and evaluation of the current state of the art, production of the dataset, training of the neural network, its evaluation, and implementation of the people counting application. The initial three chapters detail the state of the art of sensors used for 3D scene understanding, and the methods for data processing and inference they introduce; introduce the radar sensors, here used to produce the dataset for later training and evaluation; and provide an overview of the state-of-the-art deep learning methods used for object detection on point clouds. They are followed by a chapter on model training and evaluation, and a chapter detailing the requirements, used methods and algorithms, evaluation, and visualization of the people counting application.

The contributions of this thesis include (1) the in-depth study of the state of the art of the sensors and methods used for 3D scene understanding, with a particular focus on the state-of-the-art millimeter-wave frequency-modulated continuous-wave radars based on the radar-on-chip platform. It is followed by (2) the adaptation of an open-source tool for efficient annotation of sparse 3D radar point clouds using referential video, and (3) the adaptation of the state-of-the-art deep learning methods for object detection on lidar point clouds for people detection on sparse radar point clouds, and the training of the adapted neural network. As a part of this thesis, (4) a dataset of 3D radar point cloud annotations was produced. Published datasets of this nature are either very few or do not match the general requirements for people detection set out in this thesis. The model trained on this dataset was then used to develop (5) a people counting application, the evaluation of which has shown to be robust and accurate. The model trained as a part of this thesis and the subsequent tracking algorithm render the final products of this thesis straightforwardly deployable for real-world tasks, such as people counting, occupancy monitoring, trajectory prediction, and collision avoidance. This is also showcased by the application developed using the methods detailed in this thesis, together with the robustness of the trained model and the subsequent tracking algorithm.

Evaluation of the trained model has shown high accuracy and robustness, for the empirically selected thresholds. The approx. 85 % precision and 70 % recall have been achieved for the 3D Intersection over Union (IoU) threshold of 0.5, which describes approx. 70 % overlap. Lowering the threshold to 0.3 (approx. 50 % overlap, deemed as the lower limit of suitability for the people detection in the context of this thesis) results in approx. 92 % precision and 78 % recall, and an F-score just below 0.95. The evaluation of the model has shown room for improvement in resolving the high percentage (approx. 24 %) of pedestrian ground truth bounding boxes that failed to be recalled; and in improving the confidence for detections in the edge 20 % of the area of interest — the start and the end of the area recorded by the radar. The latter deficiency did not seem to affect the performance of the subsequent people counting application, and might be simply avoided by trimming the area of interest even further, but might prove an obstruction for object detection at greater distances in more noisy environments. The model deployed for real time inference on a single NVIDIA Tesla K80 GPU using the CUDA toolkit has shown the possibility of hosting the model for a real-time inference, but further experiments are needed to evaluate the possibility of its deployment e.g., on an embedded GPU.

The evaluation of the people counting application has shown promising results, matching the outcome of a manual people counting with near perfection. The extent of the evaluation and the robustness of its results are limited by the short length of the base recording used and require further analysis. Nevertheless, the error observed in the final results was below 2.5 % of the total people counted. Few of the deficiencies that might require only minor changes to the proposed system include the occasional misclassification of the running pedestrians pushing a stroller as cyclists, shielding of smaller children and people in larger groups from detection, and subsequent undercounting, and occasional failure to detect the fastest cyclists. The last of these deficiencies might be attributed to the configuration of the radar sensor, which was the result of a compromise between reliably detecting the faster cyclists and general higher resolution of the produced point clouds.

The future continuation of this work, apart from the already mentioned improvements to the proposed systems, might include (1) implementation and comparison of the performance of the neural network architectures based on the different approaches mentioned in the study portion of this thesis; (2) extending the final dataset by data of the same nature, traffic, and automotive data, and indoor people monitoring data; (3) extending the visualization application for other purposes, such as traffic analysis, collision avoidance, room monitoring, and others; and (4) optimizing the real-time capabilities of the model for deployment on an embedded system.

Bibliography

[1] Bhatta, N. and Priya, M. RADAR and its applications. *International Journal of Circuit Theory and Applications*. Jan. 2017, vol. 10, p. 1–9.

[2] Charles, R., Su, H., Kaichun, M. and Guibas, L. J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Jul 2017, p. 77–85. DOI: 10.1109/CVPR.2017.16. ISSN 1063-6919.

[3] Chen, X., Ma, H., Wan, J., Li, B. and Xia, T. Multi-view 3D Object Detection Network for Autonomous Driving. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, Jul 2017, p. 6526–6534. DOI: 10.1109/CVPR.2017.691. ISSN 1063-6919.

[4] Dreher, M., Erçelik, E., Bänziger, T. and Knol, A. Radar-based 2D Car Detection Using Deep Neural Networks. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020, p. 1–8. DOI: 10.1109/ITSC45102.2020.9294546.

[5] Geiger, A., Lenz, P. and Urtasun, R. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[6] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S. et al. Deep learning for visual understanding: A review. *Neurocomputing*. 2016, vol. 187, p. 27–48. DOI: 10.1016/j.neucom.2015.09.116. ISSN 0925-2312.

[7] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*. november 1997, vol. 9, no. 8, p. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. ISSN 0899-7667.

[8] Iovescu, C. and Rao, S. The fundamentals of millimeter wave radar sensors. [https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf]. July 2020. Texas Instruments Incorporated; Online; Accessed: 2022-05-18.

[9] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. In:. March 1960, vol. 82, no. 1, p. 35–45. DOI: 10.1115/1.3662552. ISSN 0021-9223.

[10] Khader, M. and Cherian, S. An Introduction to Automotive LIDAR. [https://www.ti.com/lit/wp/slyy150a/slyy150a.pdf]. May 2020. Texas Instruments Incorporated; Online; Accessed: 2022-05-18.

[11] KUHN, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*. 1955, vol. 2, 1-2, p. 83–97. DOI: doi.org/10.1002/nav.3800020109.

[12] LANG, A., VORA, S., CAESAR, H., ZHOU, L., YANG, J. et al. PointPillars: Fast Encoders for Object Detection From Point Clouds. In:. June 2019, p. 12689–12697. DOI: 10.1109/CVPR.2019.01298.

[13] PHAM, Q., NGUYEN, T., HUA, B., ROIG, G. and YEUNG, S. JSIS3D: Joint Semantic-Instance Segmentation of 3D Point Clouds With Multi-Task Pointwise Networks and Multi-Value Conditional Random Fields. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Jun 2019, p. 8819–8828. DOI: 10.1109/CVPR.2019.00903.

[14] QI, C. R., YI, L., SU, H. and GUIBAS, L. J. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 5105–5114. NIPS'17. ISBN 9781510860964.

[15] QI, C. R., YI, L., SU, H. and GUIBAS, L. J. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *GitHub repository* [https://github.com/charlesq34/pointnet2]. GitHub. 2017. Online; Accessed: 2022-05-18.

[16] REDMON, J., DIVVALA, S. K., GIRSHICK, R. B. and FARHADI, A. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, abs/1506.02640.

[17] TEXAS INSTRUMENTS INCORPORATED. *IWR6843ISK radar module* [https://www.ti.com/tool/IWR6843ISK]. Online; Accessed: 2022-05-18.

[18] TEXAS INSTRUMENTS INCORPORATED. Tracking radar targets with multiple reflection points. [https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/1023/Tracking-radar-targets-with-multiple-reflection-points.pdf]. March 2018. Online; Accessed: 2022-05-18.

[19] RUIZHONGTAI QI, C., SU, H., NIEBNER, M., DAI, A., YAN, M. et al. Volumetric and Multi-view CNNs for Object Classification on 3D Data. In:. June 2016, p. 5648–5656. DOI: 10.1109/CVPR.2016.609.

[20] SAHBANI, B. and ADIPRAWITA, W. Kalman filter and Iterative-Hungarian Algorithm implementation for low complexity point tracking as part of fast multiple object tracking system. In: *2016 6th International Conference on System Engineering and Technology (ICSET)*. 2016, p. 109–115. DOI: 10.1109/ICSEngT.2016.7849633.

[21] SAPONARA, S., GRECO, M. S. and GINI, F. Radar-on-Chip/in-Package in Autonomous Driving Vehicles and Intelligent Transport Systems: Opportunities and Challenges. *IEEE Signal Processing Magazine*. 2019, vol. 36, no. 5, p. 71–84. DOI: 10.1109/MSP.2019.2909074.

[22] SCHEINER, N., KRAUS, F., APPENRODT, N., DICKMANN, J. and SICK, B. Object detection for automotive radar point clouds – a comparison. *AI Perspectives*. Nov 2021, vol. 3, no. 1, p. 6. DOI: 10.1186/s42467-021-00012-z. ISSN 2523-398X.

[23] Shi, S., Wang, X. and Li, H. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[24] Shi, S., Wang, X. and Li, H. *PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud* [https://github.com/sshaoshuai/PointRCNN]. GitHub, 2019. Online; Accessed: 2022-05-18.

[25] Skolnik, M. *Radar Handbook, Third Edition*. McGraw-Hill Education, 2008. Electronics electrical engineering. ISBN 9780071485470.

[26] Voulodimos, A., Doulamis, N., Doulamis, A. and Protopapadakis, E. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*. Hindawi. Feb 2018, vol. 2018, p. 7068349. DOI: 10.1155/2018/7068349. ISSN 1687-5265.

[27] Vu, T., Kim, K., Luu, T. M., Nguyen, X. T. and Yoo, C. D. *SoftGroup for 3D Instance Segmentation on Point Clouds*. arXiv, 2022. DOI: 10.48550/ARXIV.2203.01509.

[28] Wang, P. Research on Comparison of LiDAR and Camera in Autonomous Driving. *Journal of Physics: Conference Series*. Nov. 2021, vol. 2093, no. 1. DOI: 10.1088/1742-6596/2093/1/012032.

[29] Watanabe, N. and Ryugen, H. *Cheaper lidar sensors brighten the future of autonomous cars* [https://asia.nikkei.com/Business/Automobiles/Cheaper-lidar-sensors-brighten-the-future-of-autonomous-cars]. May 2021. Online; Accessed: 2022-05-18.

[30] Weng, X., Wang, J., Held, D. and Kitani, K. AB3DMOT: A Baseline for 3D Multi-Object Tracking and New Evaluation Metrics. *CoRR*. 2020, abs/2008.08063.

[31] Wijmans, E. Pointnet++ Pytorch. *GitHub repository* [https://github.com/erikwijmans/Pointnet2_PyTorch]. GitHub. 2018. Online; Accessed: 2022-05-18.

[32] Zhou, Y. and Tuzel, O. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

Appendix A

Contents of the Included Storage Media

```
/
├── dataset
└── src
    ├── eval ........................................... Scripts for evaluation plots
    ├── labeling
    │   ├── ... ..................................................... Utility scripts.
    │   └── labelCloud
    │       ├── labelCloud
    │       │   └── ...labelCloud src files...
    │       ├── labelCloud.py ................................. Main labelCloud script
    │       ├── labels
    │       │   └── ...labels as *.JSON...
    │       ├── pcds
    │       │   └── ...point clouds as *.PLY...
    │       └── ...
    ├── MyPointRCNN
    │   ├── data ........................... Data dir containing the dataset splits
    │   ├── lib ................................... Libraries implementing the model
    │   ├── tools ............ Tools for training, data loading, evaluation, and inference
    │   ├── pointnet3_lib ............................... PointNet++ implementation
    │   └── ...
    ├── parsing
    │   ├── rec_parser.py ............................... Recording parsing script
    │   └── mmwave ........................................ Mmwave parsing src files
    ├── utils ........................................................ Utility scripts
    ├── visual
    │   ├── visualize.ipynb ................................ 4D visualization script
    │   ├── records
    │   │   └── ...recording files...
    │   └── ...Visualizer src files...
    ├── patches ............ Patch files listing the changes to the original source codes
    └── latex ................................................... Latex source files
```