TUM School of Computation, Information and Technology
Technical University of Munich

Master's Thesis in Robotics, Cognition, Intelligence

Deep Multimodal Sensor Fusion for 3D Perception in Autonomous Vehicles Using Occupancy Networks

Multimodale Sensorfusion für die 3D-Umfeldwahrnehmung von Autonomen Fahrzeugen mittels Occupancy Netzen

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor Dr.-Ing. Frederik Deroo
Walter Zimmer, M.Sc.
Felix Neumann, M.Sc.

Author Omar Zayed

Date May 15, 2023 in Munich

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Munich, May 15, 2023

(Omar Zayed)

Abstract

In the last year, 3D occupancy prediction for autonomous driving has gained significant attention. As occupancy prediction mitigates some major limitations found in 3D object detection using bounding boxes, it is a promising alternative for creating a reliable 3D representation of the environment. Motivated by this, we present two approaches to predict 3D occupancy using implicit representations in the context of autonomous driving and showcase their performance, and limitations in the infrastructure as well as in the on-board sensor setups. First, we present a new data generation pipeline that tackles the lack of ground truth data in this emerging field by generating multi-sensor synthetic datasets for both the infrastructure and on-board settings. In the first approach, we extend an existing method developed originally for LiDAR-based object and indoor scene reconstruction to the autonomous driving domain. The approach exhibits competitive performance in both setups and is able to outperform the model-based baseline, as well as show great results when tested on real-world data while being trained on synthetic data. In the second approach, we propose a novel method for 3D occupancy prediction in the on-board sensor setup based on deeply fused input from RGB cameras and LiDAR sensors. While the method did not outperform other SOTA methods, we argue that the model has indeed achieved promising results that can be improved in the future.

Zusammenfassung

Im letzten Jahr hat die 3D-Occupancy Prediction für autonomes Fahren erhebliche Aufmerksamkeit erhalten. Da die Occupancy Prediction einige wesentliche Einschränkungen der 3D Object Detection durch die Verwendung von Bounding Boxes mildert, stellt sie eine vielversprechende Alternative dar. Motiviert dadurch präsentieren wir zwei Ansätze zur 3D-Occupancy Prediction im Kontext des autonomen Fahrens und präsentieren ihre Potenziale und Einschränkungen sowohl in der Infrastruktur- als auch in der On-Board-Sensor-Konfiguration. Zunächst präsentieren wir eine neue Datenerzeugungs-Pipeline, die das Fehlen von Ground-Truth-Daten in diesem aufstrebenden Bereich bewältigt. Im ersten Ansatz erweitern wir eine vorhandene Methode, die ursprünglich für LiDAR-basierte Indoor-Objekt und Szenenrekonstruktion entwickelt wurde, auf den Bereich des autonomen Fahrens. Der Ansatz zeigt eine wettbewerbsfähige Leistung in beiden Konfigurationen und übertrifft das modellbasierte Baseline-Modell sowie zeigt gute Ergebnisse bei Tests mit realen Daten, während es mit synthetischen Daten trainiert wurde. Im zweiten Ansatz schlagen wir eine neuartige Methode zur 3D-Occupancy Prediction in der On-Board-Sensor-Konfiguration vor, die auf RGB-Kameras und LiDAR-Sensoren basiert. Obwohl die Methode andere State-of-the-Art-Methoden nicht übertrifft, argumentieren wir, dass das Modell tatsächlich vielversprechende Ergebnisse erzielt hat, die in Zukunft weiter verbessert werden können.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 3
- 1.3 Contribution . . . 4
- 1.4 Content Overview . . . 4

2 Background **7**
- 2.1 Perception Systems . . . 8
  - 2.1.1 Cameras . . . 8
  - 2.1.2 LiDARs . . . 10
  - 2.1.3 Camera vs. LiDAR Comparison . . . 11
- 2.2 3D Representations . . . 12
- 2.3 Perception Tasks in Autonomous Driving . . . 13
- 2.4 Intelligent Infrastructure: Providentia++ . . . 14
- 2.5 CARLA Simulator . . . 15

3 Related Work **17**
- 3.1 LiDAR-based 3D Object Detection . . . 17
- 3.2 Camera-based 3D Object Detection . . . 20
- 3.3 Multi-modal 3D Object Detection . . . 23
- 3.4 Model-Based 3D Occupancy Estimation . . . 25
- 3.5 Learning-Based 3D Occupancy Estimation . . . 27

4 Methodology **33**
- 4.1 Synthetic Data Generation . . . 33
- 4.2 Occupancy Prediction . . . 36
  - 4.2.1 LiDAR-based Occupancy Prediction . . . 36
  - 4.2.2 Multimodal-based Occupancy Prediction . . . 38
  - 4.2.3 Loss Definition . . . 40
- 4.3 Implementation Details . . . 41

5 Evaluation and Results **43**
- 5.1 Datasets . . . 43
  - 5.1.1 Infrastructure Datasets . . . 43
  - 5.1.2 Onboard Datasets . . . 44
- 5.2 Evaluation Metrics . . . 46
- 5.3 LiDAR-based Methods . . . 47
  - 5.3.1 Model Configurations . . . 47
  - 5.3.2 Onboard Sensor Setup . . . 48
  - 5.3.3 Infrastructure-based Sensor Setup . . . 51
  - 5.3.4 Inference time . . . 55

5.3.5 Further Analysis 57
5.4 Multimodal Methods 58

6 Conclusion and Future Work **61**
6.1 Conclusion 61
6.2 Future Work 62

Bibliography **69**

Chapter 1

Introduction

1.1 Motivation

In recent years, automated driving has gained huge momentum. The main driver behind this increased momentum has been the recent advances in computation technologies, which have made running complex and computationally intensive algorithms in real-time possible. The interest in automated driving is not limited to vehicles only, but also includes other autonomous systems used for transportation, such as driverless trains.

This evolving technology aims to revolutionize the whole transportation industry by removing the need for human drivers to sit behind wheels, thereby reducing the number of accidents that cause significant losses in terms of fatalities and economic costs. In addition to that, automated driving would allow for higher utilization rates for vehicles, ultimately decreasing the cost of transportation.

Automated driving, at its core, relies on combining various sensors, complex algorithms, and sophisticated control systems to perceive and interpret the surrounding environment, and based on that, make decisions that control the movement of the vehicle.

Perception in automated driving is a complex process that involves processing data from multiple sensors of different modalities, such as cameras, LiDARs, and radars, that capture information about the environment. An example of such a sensor setup can be seen in the Nuscenes research platform shown in Figure 1.1. This information is then processed to compute an internal representation of the system's operating environment. Further systems depend on this to recognize and interpret objects and their movements, and predict their future trajectories. Perception is critical for automated driving systems to operate safely and efficiently, as it enables the vehicle to navigate through complex environments, avoid obstacles, and make decisions in real-time.

The perception system is not limited to the vehicle-mounted setup only. The use of infrastructure-mounted sensors has been an active research field in recent years. The research in this field aims to provide automated driving vehicles with additional information about the environment and the road conditions, where the onboard sensors may have been insufficient in capturing. The Providentia++ test bed [Krä+; Cre+] is an example of such an installation, where different sensors (cameras, LiDARs, radars) are mounted on gantry bridges as seen in Figure 1.2. The sensors used in the infrastructure setup are primarily similar to those mounted on the vehicles. Nevertheless, some differences have to be taken into consideration when developing or adapting perception algorithms for the infrastructure setup.

Autonomous trains rely on concepts and technologies in perception similar to those developed for vehicles. These systems have to be able to operate in complex dynamic environments as well. The key difference may be the need for the perception systems to operate at extended

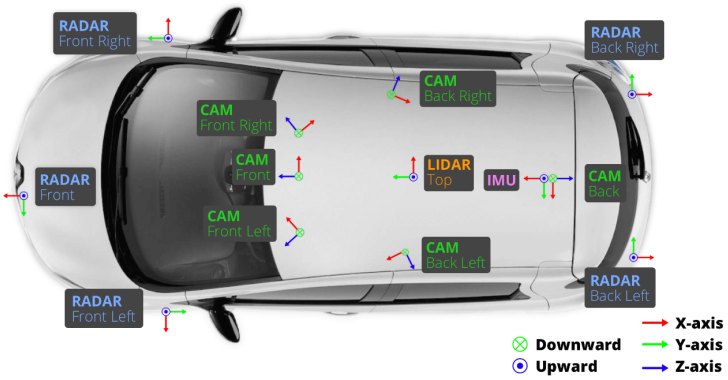

Figure 1.1: Motional (formerly NuTonomy) research platform vehicle used in Nuscenes dataset [Cae+].

Figure 1.2: Overview of the Providentia++ test bed showcasing the different sensor stations on the highway as well as in urban field in Garching, Germany.

ranges, since trains' braking distance is ranges from several hundreds of meters for a regional train up to kilometers for high-speed trains.

Perception methods used in automated driving can be principally divided into model-based approaches and learning-based approaches. Both approaches have their advantages and disadvantages, which in turn affect their performance and applicability in different scenarios.

Classical model-based approaches have been widely used in this context. They typically rely on physical and mathematical models that provide reliable deterministic results. These approaches are often optimized to be computationally efficient, making them adequate for deployment on real-time systems. Nonetheless, these model-based approaches have the disadvantage of being dependent on the accuracy and completeness of the underlying models, which makes them limited to certain situations or use-cases, and limits their ability to generalize to handle complex and dynamic environments.

Learning-based methods have become increasingly relevant in perception. They have proven to be very flexible and adaptive, as seen in their ability to generalize against a wide range of situations. They are also characterized by their ability to capture complex high-level features from raw data, which provides accurate results. Nevertheless, learning-based approaches generally require large amounts of data that are often challenging to collect and label. This data plays a huge role in the performance as well, since non-representative data

can often cause the models to perform poorly. Despite that, they have been able to produce very promising results lately, often outperforming model-based methods in various tasks.

1.2 Problem Statement

Bounding box detections and semantic segmentation are two commonly used methods to model the environment for autonomous vehicles, but for supervised learning-based methods they have shown to be limited to a finite number of known classes present in the training data. Additionally, the quality and sparsity of the input data plays a huge role in the quality of the predictions. This in turn limits the performance of these methods in challenging dynamic environments, where new unseen objects may be present. Such a limitation was showcased by Tesla in as can be seen in Figure 1.3, where the multi-camera-based object detector failed to detect a trailer in plain sight, since it did not resemble any previously seen dynamic object. It is maybe then deducible that using bounding boxes or semantic segmentation is not the most suitable method to model the environment. A potential alternative are occupancy grids.

Occupancy grids are also common representations, but to a lesser extent. They are geometry-based representation which have the advantage of being class agnostic in comparison to semantic representations. Each region in the grid contains a value indicating whether it is occupied or not. A semantic label can additionally be added to each occupied region when needed. In 2D occupancy grids, these probability values represent presence of obstacles on the ground plane, the height dimension is not taken into consideration. 3D occupancy grids on the other hand represent all three dimensions. While 2D grids can be sufficient for various tasks, the additional height dimension in 3D grids provides information necessary to deal with more complex environments and situations.

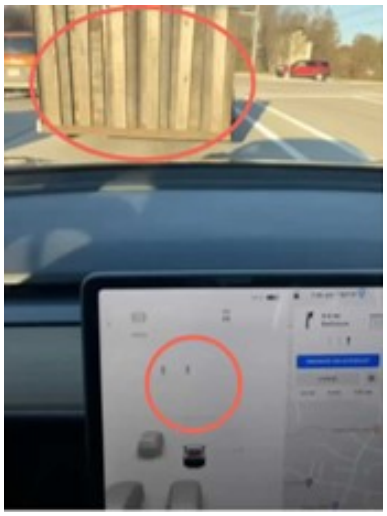

Figure 1.3: An example where camera-based object detector on a Tesla vehicle failed to detect an object. [Tes22a; Tes22b] .

Classically, 3D occupancy maps are constructed using model-based methods such as ray tracing, but in the recent years several learning-based methods have been introduced to further improve the results. In contrast to classical occupancy maps that output the occupancy at fixed discrete locations, occupancy networks approximate a function that can predict the occupancy of any 3D location given an observation and the desired location [Mes+]. Until very recently, occupancy networks were used only for single object reconstruction, or for occupancy prediction in indoor scenes. These methods have used only single sensor modalities,

either cameras, or LiDARs.

In automated driving 3D object detection benchmarks, such as Nuscenes [Cae+], approaches based on multiple sensor modalities have been steadily rising up the leaderboard, frequently outperforming single modality approaches. Complementary sensors such as LiDARs and cameras can provide synergistic information that enables more accurate reasoning about a scene. This provides an interesting direction, where the performance of deep fused camera-LiDAR features can be compared to the case where a single modality is used to predict the occupancy.

The goal of this thesis is to create a class agnostic 3D representation of the surrounding environment in autonomous vehicles, based on deeply fused inputs from camera and LiDAR sensors. This can be achieved by leveraging the concept of occupancy networks and adapting them to be used in autonomous driving for 3D representation of the surrounding environment, instead of indoor scene reconstruction.

1.3 Contribution

The main contribution of this thesis is developing a novel approach for 3D perception that can be used in either vehicle mounted sensor setups or infrastructure-based sensor setups. The work provides an exploration to applying occupancy networks for creating a class agnostic 3D perception system, that can be applied using different sensor configurations (camera-LiDAR, LiDAR only and camera only). The proposed approach is tested and evaluated on real-data, while being mainly trained on synthetic data generated in a novel way. The results of the LiDAR-based methods have shown competitive performance outperforming the selected baseline, while the camera-LiDAR-based methods have shown promising results that can be further improved.

1.4 Content Overview

This thesis is structured in 6 chapters. The remainder of this work is organized as follows:

- Chapter 2: Background

  This chapter provides relevant background information, that is necessary to understand the proposed methods. It includes a brief introduction to the field of automated driving, and the different sensors used to tackle perception tasks this context. Additionally, it provides an overview on different 3D representations. A brief introduction on the Providentia++ [Krä+] test-bed is also provided, in addition to an overview of the CARLA simulator [Dos+].

- Chapter 3: Related Work

  In this chapter, we present a detailed review of the literature and related work in the area of 3D occupancy prediction for automated driving using different sensors, and multi-modal sensor fusion as a general task.

- Chapter 4: Methodology

  In this chapter, the proposed methods for occupancy prediction are discussed in detail. Different models and approaches are presented and discussed, where the models not only differ in their architecture, but also in their sensor modalities inputs.

- Chapter 5: Results and Evaluation

  This chapter presents the results of the experiments conducted on the proposed methods. It includes quantitative and qualitative analysis of the results, as well as supporting visualizations and diagrams. Additionally, we present a detailed analysis on the results, where the strengths against other approaches are highlighted, in addition to the limitations of the developed approaches.

- Chapter 6: Conclusion and Future Work

  This chapter provides conclusions of the thesis, bases upon implications and limitations of the proposed methods, as well as future directions for further research.

Chapter 2

Background

Since the industrial revolution, humans have been trying to automate every process, from production to transportation. Autonomous vehicles have been a conceptual idea since the early 1900s, and they were always part of the imagination of how the future would look like. With the advances in technology in the last century, first prototypes of assisted driving started to appear in the 1950s, when vehicles equipped with radar sensors for brake assistance were developed. Later, in the 1980s, autonomous vehicles became a hot research topic for academia and industry alike, with many prototypes being developed, such as in Figure 2.1to drive automatically solely based on sensor readings from mounted cameras. The research in this area continued as more technologies have matured. This allowed the integration of new sensors such as laser scanners, in addition to more complex computing platforms for decision-making.

For an autonomous vehicle to navigate safely, the vehicle has to first sense the surrounding environment. While sensors generally capture what is happening around the vehicle, this information has to be processed in order for the vehicle to understand the complex context of the whole scene. Only then, the planning and decision-making can be executed.

Human drivers rely primarily on visual input from the eyes while driving, which is then complemented by other sensory inputs from the ears. The cognitive and perceptual abilities of humans allow them to process complex contextual information such as predicting the intent of other traffic participants which is challenging for machines. Additionally, humans have the ability to make judgements based on incomplete information in ambiguous situations, and to adapt to changing situations as well. This proved to be also difficult for machines, however the gap is being steadily decreased as the technology and research advances, with machines even outperforming humans in some specific tasks [He+15].

In this chapter, we provide a brief introduction to perception systems in autonomous ve-

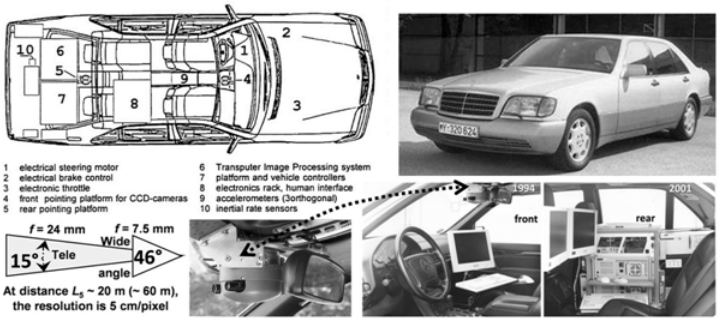

Figure 2.1: The seeing passenger car (VaMoRs-P), one of the first self-driving vehicles equipped with camera sensors as part of the EUREKA project [E D+94] .

hicles. We start by briefly explaining how cameras and LiDAR sensors work, and how they are used in autonomous vehicles. Then, we compare the two sensors in terms of their advantages and disadvantages. After that, we provide a brief introduction on different methods for 3D representations of the environment, with an emphasis on their advantages and disadvantages. Additionally, we highlight perception tasks in autonomous driving, which are relevant to this work. In Section 2.4, we introduce the infrastructure Providentia++ test-bed which is used in this work. Finally, CARLA simulator is introduced in Section 2.5.

2.1 Perception Systems

The perception system in autonomous vehicles is a cornerstone for their success. Different sensors have been used in autonomous vehicles in order to capture what is happening around the vehicle. The most commonly used sensors for perception are cameras, radars, LiDARs, and ultrasonic sensors. In this work, we focus on inputs from camera and LiDAR sensors only since they have complementary strengths.

2.1.1 Cameras

In order to better explain how a camera principally works, we consider the simple pinhole camera model. This is a purely geometric model which explains how 3D points in the world are projected into the 2D image. A camera works by capturing light emitted from objects. This light passes through the camera lens, which focuses the light onto a digital sensor or a film [Pri12]. The color information is extracted by measuring the intensity of specific visible light spectra. Commonly red, green and blue are measured to determine the color information.

In the real world, a pinhole camera is composed of a sealed chamber with a small hole (called the pinhole) in the front. Light rays from an object in the outside world pass through the hole and form an inverted image on the image plane. However, the inverted image produced by the pinhole camera can be inconvenient to work with. Therefore, instead of using the true pinhole model, we consider a virtual image that would result from placing the image plane in front of the pinhole. Although it is not possible to physically construct a camera in this manner, the resulting image is mathematically equivalent to the true pinhole model (with the exception that the image is right-side up) and is easier to comprehend. This can be seen in Figure 2.2.

The pinhole itself (the point at which the rays converge) is called the optical center. We will assume for now that the optical center o is at the origin of the 3D camera coordinate system in which points are represented as $\mathbf{w} = [u, v, w]^\mathsf{T}$. The virtual image is created on the image plane, which is displaced from the optical center along the w-axis or optical axis. The point where the optical axis strikes the image plane is known as the principal point. The distance between the principal point and the optical center is known as the focal length f.

The standard convention is for the origin of the image coordinate system to be located in the top left corner of the image. Pixel coordinates $[x, y]$ point to the right and downward, respectively. The center of the image (principal point) is designated as $c = [c_x, c_y]$ in xy-coordinates. The camera coordinate system is established as a right-handed coordinate system, with the x-axis parallel to the u-axis and the y-axis parallel to the v-axis of the image coordinate system. This setup means that the w-axis extends toward the image plane at the center of the image. To project a 3D point $\mathbf{w} = [u, v, w]^\mathsf{T}$ in camera coordinates onto the image plane, a ray is cast from p to o to generate a 2D point $\mathbf{x} = [x, y]$ on the image plane.

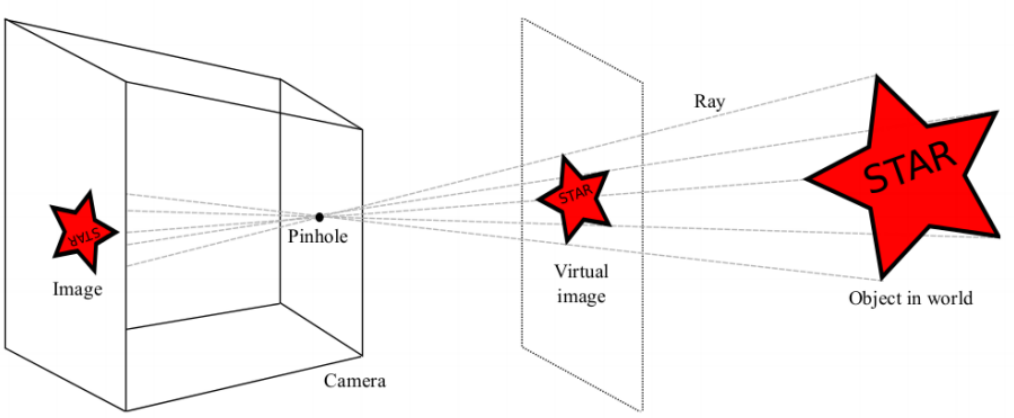

Figure 2.2: The pinhole camera model: Rays from an object in the world pass through the pinhole in the front of the camera and form an image on the back plane (the image plane) [Pri12].

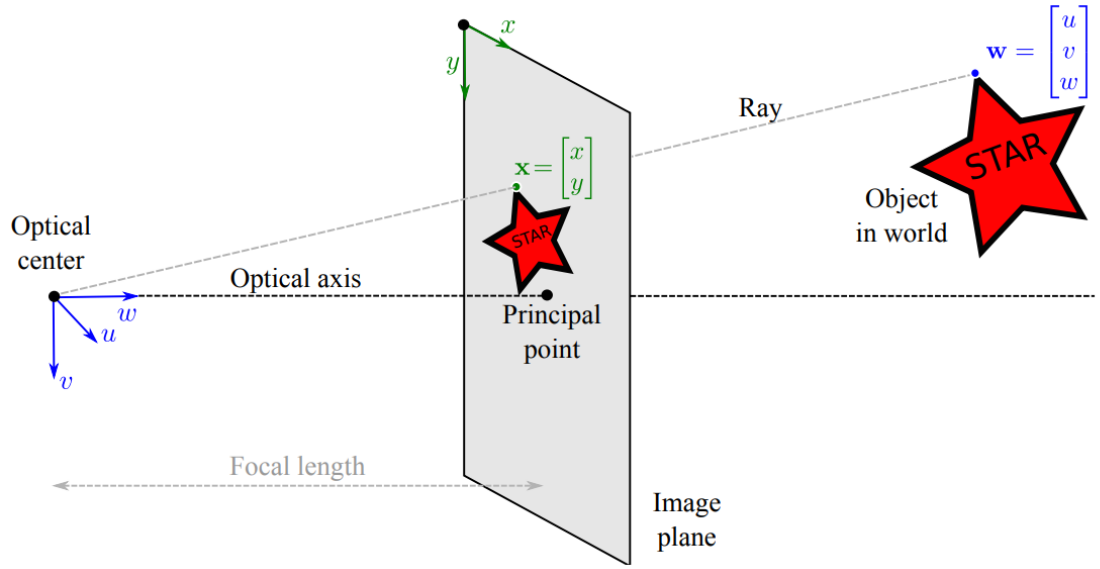

Figure 2.3: Pinhole camera model terminology.

This is illustrated in Figure 2.3.

To calculate the 2D projection x of a single 3D point w onto the image I, we multiply the focal length f with the normalized camera pixel coordinates $\mathbf{x} = [\hat{x}, \hat{y}]^T$ where $\hat{x} = u/w$ and $\hat{y} = v/w$, where this serves as a scaling factor. The photoreceptors may differ in the x- and y-directions, and so the scaling differs according to the respective focal length. An offset parameter $c = [c_x, c_y]$ has to be additionally added, since the pixel position $\mathbf{x} = [x, y]^T$ is at the top-left of the image rather than the center as can be seen in Equations 2.1 and 2.2.

$$x = \frac{f_x u}{w} + c_x \tag{2.1}$$

$$y = \frac{f_y v}{w} + c_y \tag{2.2}$$

These calculations can also be expressed using matrix notation, by introducing the camera matrix $\mathbf{M}$ and the intrinsic matrix $\mathbf{K}$. The camera matrix $\mathbf{M}$ describes the mapping of

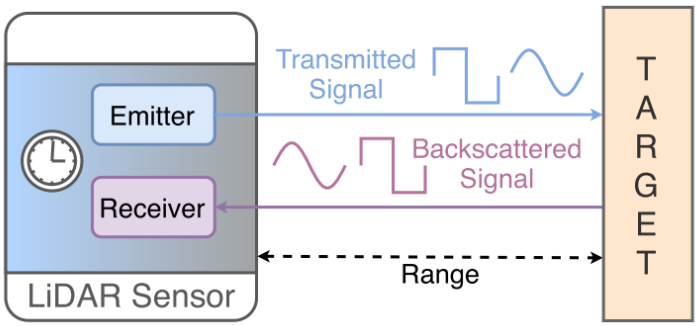

Figure 2.4: Operation principal of LiDAR technology [RCG22].

a 3D point from the world coordinate system to the camera coordinate system, while the camera intrinsic matrix **K** transforms the points from the camera coordinate system to pixel coordinates. This can be seen in Equations 2.3 and 2.4.

$$\mathbf{x} = \frac{1}{w}\mathbf{KMw} \tag{2.3}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{w}\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} \tag{2.4}$$

In this simplified model, we assume that there is neither rotation nor translation between the world coordinate system and the camera coordinate system. This means that the camera matrix **M** is the identity matrix. Additionally, it does not take distortions and skew into account.

2.1.2 LiDARs

Light detection and ranging (LiDAR) has long been utilized in military and other applications since the 1930s [McM19; RCG22], however it first appeared in the context of automated driving in the Defense Advanced Research Projects Agency (DARPA) 2005 Grand Challenge. Since then, LiDARs have played an increasingly important role in the recent advances in automated driving. When first introduced, LiDARs were extremely expensive, with the costs that can exceed the price of the car. This has been changing with the wide adaption of such sensors and the continuous maturation of the technology. LiDARs have been also used in different applications other than automotive; these applications range from 3D reconstruction, mapping to environmental monitoring and surveillance.

A LiDAR is an active sensor, that falls under the category of Time of Flight (ToF) sensors [McM19; RCG22; Raj+20; Hec18]. A LiDAR works by emitting a laser beam towards the object, and then measuring the travel time as seen in Figure 2.4. The range R to the target can be calculated using Equation 2.5, where c denotes the speed of light and τ is the round trip delay of the emitted signal.

$$R = \frac{1}{2}c\tau \tag{2.5}$$

There exists different technologies for LiDARs, the four most common scanning mechanisms are opto-mechanical, electromechanical, micro-electromechanical systems (MEMS),

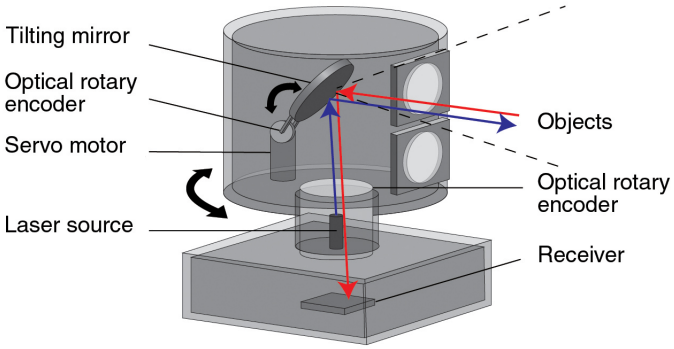

Figure 2.5: Typical spinning 3D LiDAR.

and solid-state scanning, with the electromechanical LiDAR being currently most commonly used. These can produce 1D, 2D or 3D scans [Raj+20; McM19].

Autonomous vehicles use mainly 2D or 3D scanning LiDARs. While the field of view (FoV) of 2D LiDARs is only limited to the horizontal plane by employing single channels, 3D LiDARs involves both the horizontal and vertical plane using multiple stacked channels. This results in a very high cost for 3D LiDARs in comparison to 2D ones. The horizontal FoV also depends on the type of the LiDAR, where spinning LiDARs provide a wider view up to 360° compared to nonrotating LiDARs. For the scope of this work, we only consider the spinning variant.

LiDAR typically rotates at a speed of 5 to 20 Hz, resulting in a complete measurement frame every 100 ms for a sensor operating at 10 Hz. The rotational speed also affects the angular resolution of each frame, typically ranging from 0.2 to 0.8 degrees. While most LiDAR manufacturers state a maximum operating range of 50 to 500m, the point density decreases exponentially as distance from the sensor increases due to the sensor emitting all laser beams from one point with varying vertical angles. As a result, an object may only be sensed by one LiDAR channel or may be completely missed, making it invisible to the sensor, when the vertical spacing is greater than the object's height. Additionally, LiDAR can only measure the distance of surfaces facing the sensor and cannot sense what is behind the first surface that blocks a laser beam's path, resulting in occlusion and an incomplete perception of an object's shape. However, laser beams can partially penetrate glass and windows, leading to an absence of reflections in the concerned spot. Finally, laser light can scatter when it hits the edge of walls, leaves, or tree branches [Bru22; Raj+20; RCG22; War19; McM19; LI20].

2.1.3 Camera vs. LiDAR Comparison

Having explained the idea of how cameras and LiDARs work, it is interesting to compare both sensors in the spectrum of autonomous driving. The LiDAR technology is capable of real-time sensing of the surrounding environment, and generating accurate 3D representation out of the box. However, the point cloud data obtained is usually sparse and can include noise due to various factors, including acceleration, deceleration, change of driving direction, and skewing due to ego motion. While most LiDAR systems can provide intensity information about the target surface of different objects based on the reflected rays and properties, this information can vary greatly depending on the LiDAR system, weather conditions, and the target material among others. Additionally, the LiDAR point cloud data only contains spatial coordinate information of the object's surface, with hardly any other object specific information that can

Capabilities	Human	Camera	LiDAR
Object detection	⋆⋆⋆	⋆⋆	⋆⋆⋆
Object classification	⋆⋆⋆	⋆⋆⋆	⋆⋆
Distance estimation	⋆⋆	⋆⋆	⋆⋆⋆
Edge detection	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
Lane tracking	⋆⋆⋆	⋆⋆⋆	⋆
Visibility range	⋆⋆⋆	⋆⋆	⋆⋆
Poor weather performance	⋆⋆	⋆	⋆⋆
Bad illumination	⋆	⋆	⋆⋆⋆

Table 2.1: Performance matrix of LiDAR and camera sensors in comparison with human driver [Wan21].

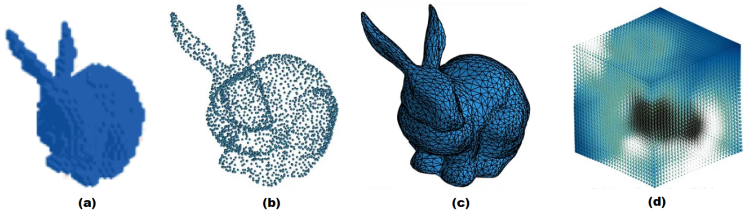

Figure 2.6: The four common types of 3D representation. (a) Voxels. (b) Point cloud. (c) Explicit representation (mesh). (d) Implicit representation [M M+19].

aid in object detection and classification tasks [Wan21; BGZuu; LI20].

Cameras on the other hand are a widely used technology in autonomous vehicles because they offer the most accurate way to visually represent the vehicle's surroundings in a rich, dense representation. They can be placed on every side of the car to capture a wide picture of the environment, aiding object recognition. However, adverse weather conditions and external light sources can severely affect camera output, which poses a limitation on their use due to the drop in performance. Additionally, the depth information is lost during the camera projection process. Although monocular cameras can be used to estimate the depth, it performs very poorly when compared to other sensors such as LiDARs or Radars. Multi-camera setups can however be used to robustly estimate the depth and create a 3D representation of the environment with a much better performance compared to single cameras. Compared to LiDARs, which are active sensors providing self illumination, cameras use natural light or an external illumination source to recognize colors of objects, such as cars and traffic lights, and identify distant objects with higher resolution and lower costs in full-light conditions [Wan21; BGZuu; RCG22].

Both sensors have their advantages and disadvantages, as seen in Table 2.1, when it comes to 3D perception in autonomous driving. A combination of using both sensors is often used in autonomous driving systems to provide the most comprehensive and reliable results [Wan21; BGZuu].

2.2 3D Representations

There are multiple 3D representations, which can be categorized into 4 main categories as can be seen in Figure 2.6; point-based, voxel-based, implicit, and explicit representations [YSA18; Mes+; Son+20; Mic+; Nie+; M M+19].

Point-based representations have a long history in robotics and computer graphics, but

their irregular structure complicates their usage in deep learning. This changed with the proposal of PointNet [Qi+a] which achieved invariance by means of a global pooling operation over all points. Since point clouds are typically sparse, they often require extensive post-processing to create denser representation. Additionally, although they require fewer parameters to store than dense volumetric grids, for instance, they are typically limited by the number of points, and cannot represent topological relations [YSA18; Mes+].

Voxel-based representations are commonly used in learning-based 3D reconstruction tasks due to their simplicity, but they are limited in terms of memory and computation. Several works have proposed to operate on multiple scales or use octrees for efficient space partitioning to address the cubic memory requirements of voxel-based representations. However, even with adaptive data structures, voxel-based techniques are still limited. While voxel representations can be processed in a fully-convolutional manner, they are memory-intensive and inherently restrict their ability to produce fine-scale, detailed surfaces [Mes+; Son+20].

Explicit surface representations, such as triangle meshes, are widely used in the graphics community due to their ability to capture detailed geometry. However, they are irregular in nature, not uniquely defined, and not easily integrated into learning frameworks. In contrast, implicit surface representations are more structured and can be organized as a set of vertices and edges. Several works have explored using neural networks to classify and segment shapes based on the graph of a 3D mesh, but they assume a fixed input graph while 3D reconstruction requires inferring the graph itself. Methods for mesh-based inference are limited and often restricted by a fixed 3D topology or mild deviations from a 3D template. Some works have proposed directly regressing the vertices and faces of a mesh using a neural network, but these approaches can result in non-watertight reconstructions with self-intersecting mesh faces [YSA18; Mes+; M M+19].

Implicit surface representations define a volumetric function that characterizes the embedding space of a 3D object by classifying each 3D point as inside, outside, or exactly on the surface. They have continuous representations of 3D geometry without topology restrictions, and they are becoming increasingly popular as they can be processed using standard 3D CNNs. Two examples of such representations are binary occupancy grids, and Truncated signed distance function (TSDF). Binary occupancy grids are being used more often recently, but they can suffer from the limitation of accuracy being restricted to the size of a voxel. Adaptive space partitioning techniques can scale up the resolution, but sub voxel estimation is required to avoid voxel-based discretization artifacts. TSDF is another representation that stores the truncated signed distance to the closest 3D surface point in each voxel. However, post-processing is required for isosurface extraction, which can be avoided by an end-to-end trainable solution [YSA18; Pou+; Son+20].

2.3 Perception Tasks in Autonomous Driving

Detection, segmentation and tracking are considered to be the main perception tasks in automated driving [Sun+; Liu+17; HC; Li+22]. Although tracking is a very important task, we consider it out of the scope of this work.

Object Detection: It refers to the task of detecting presence of an object in addition to localizing it by fitting a bounding box around it. In the 2D object detection, objects are detected and localized using a 2D bounding box in the image space, without providing any information about the objects' position in 3D. In contrast, 3D object detection provides information about the objects' position, orientation and size in the 3D space.

Semantic Segmentation: In 2D semantic segmentation, each pixel in the image is assigned a semantic category such as roads, buildings, vehicles, and pedestrians based on a

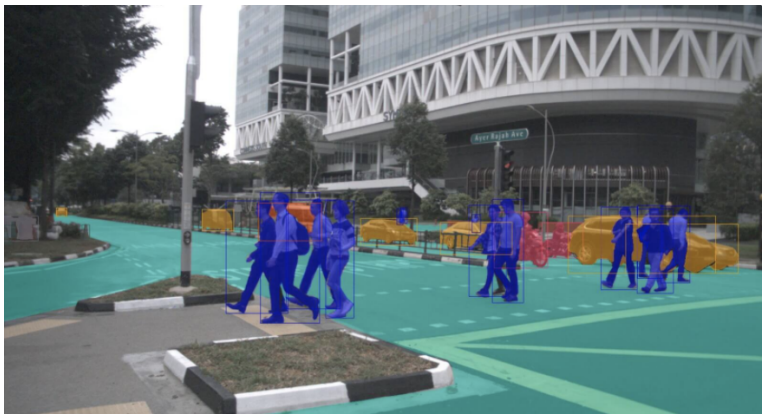

Figure 2.7: 2D Object detection and semantic segmentation in autonomous driving as seen in Nuscenes dataset [Cae+].

classification of which category it belongs to. 3D semantic segmentation is similar to 2D semantic segmentation, but point clouds or voxels are classified into the semantic categories instead of image pixels.

An example of 2D object detection and semantic segmentation in autonomous driving is shown in Figure 2.7.

2.4 Intelligent Infrastructure: Providentia++

Sensors can be mounted in different settings. On-board sensors have limited visibility due to occlusions and a restricted field of view. Infrastructure sensors on the other hand, can avoid occlusions and offer wider visibility, but installation requires a considerable effort in setup and construction, and can be difficult to maintain. Additionally, autonomous vehicles cannot solely rely on infrastructure sensors due to possible signal interference and disconnections, it is however argued that on-board sensors may be not sufficient on their own. Integrating both on-board and infrastructure sensors might therefore be a solution to improve dependability, integrality, and credibility of environmental perception.

The last years saw the introduction of several intelligent infrastructure projects such as Test Area Autonomous Driving Baden-Württemberg [Tob+18], the Austrian highway project [Sie+19] and the Providentia project [Krä+] near Munich.

The Providentia test bed was initially built along a stretch of the A9 highway close to Munich as a large-scale distributed multi-modal sensor system. In addition to the sensors, the systems includes multiple edge computing units running different algorithms providing a real-time digital twin of the road traffic. Initially, the setup was composed of two gantry bridges equipped with 8 sensors each, with two cameras and two radars for each viewing direction providing a redundant coverage of the whole stretch. This can be seen in Figure 2.8.

The Providentia++ project followed up shortly after, where the test bed was expanded to cover other scenarios rather than just the highway. The covered scenarios now include freeway, highway, roundabout and intersection in urban areas. The map of the test bed can be seen in Figure 2.9. Covering urban areas ensures that more pedestrians and cyclists are present in the captured data. The expansion saw the addition of new sensor types such as LiDARs and event cameras as well, in addition to increasing the numbers of the existing sensors. This way, the test bed now includes 75 sensors along 7 measurement points. Particularly interesting for this work is the $s110$ intersection.

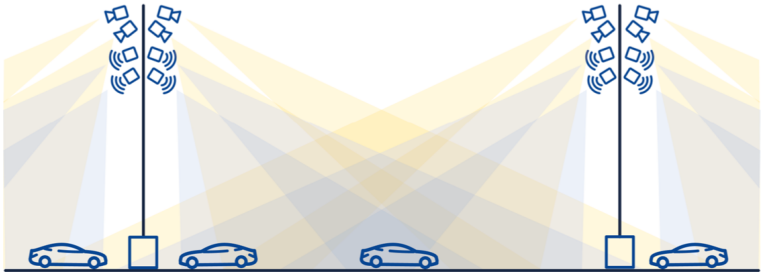

Figure 2.8: The initial sensor setup of the Providentia test bed on the A9 highway [Krä+].

Figure 2.9: Top: Overview of the test bed Providentia++. The original stretch highlighted in yellow, and the extension is highlighted in blue [Cre+]. Bottom: Gantry bridge within the s110 intersection in the Providentia++ project.

2.5 CARLA Simulator

Learning-based methods require a huge amount of data when compared to model-based ones. To overcome the absence of real data, simulators such as CARLA [Dos+] can be used.

CARLA (Car Learning to Act) is an open source simulator for driving, built specifically to overcome the need for physical systems to develop and test algorithms in both perception and control. CARLA contains various maps denoted as "Towns" that cover a wide range of driving environments such as cities, suburban and rural areas, mountains and even highways. These maps were developed from the ground up while taking small details like vehicle models, buildings, pedestrians, street signs, and similar into account. Additionally, the simulator also enables the specification of different environmental conditions, including weather and time of day, making it a highly flexible and adaptable tool for autonomous driving research. The environments are composed of 3D models of static and dynamic objects, designed for visual quality and rendering speed with the option to add additional models by the user if needed.

At its core, CARLA is built as an open-source layer on top of Unreal Engine 4 (UE4), allowing it to be flexible and realistic. The server-client system runs the simulation and renders the scene, with the client API implemented in Python to interact with the server via sockets. The client sends commands and meta-commands to the server and receives sensor

Figure 2.10: An example scene from CARLA.

readings in return. Users can also use the client to inquire additional information, which is possible using the Python API.

The behavior of non-player characters such as vehicles and pedestrians can be configured as well, which are critical for creating a realistic simulation. The vehicles are controlled by a basic controller that governs their behavior in terms of lane following, respecting traffic lights and speed limits, and decision-making at intersections. This allows the creation of random as well as deterministic, recurring scenarios. Different sensors are supported as well, such as RGB cameras, LiDARs, Radars, and event cameras in addition to a number of pseudo-sensors providing ground-truth depth and semantic segmentation.

Chapter 3

Related Work

The 3D perception is a complex task that is vital for safe and reliable autonomous vehicles, and although this work focuses on representing the surrounding environment with occupancy, 3D object detection in the form of bounding boxes is currently most commonly used. The methods developed for 3D object detection have been extensively researched for longer times, and they provide a starting point to build upon in order to develop a learning based occupancy prediction approach. We, therefore, present some state-of-the-art methods in 3D object detection using cameras, LiDARs in addition to fusion based models. Afterwards, we present some model-based methods for 3D occupancy prediction. Finally, we present some learning-based methods for occupancy prediction.

3.1 LiDAR-based 3D Object Detection

Point clouds have represented a challenge to researchers in learning-based 3D object detection. But due to their irregular format, they had to be first transformed into another representation before being processed. In 2017, PointNet [Qi+a] presented a solution for this problem. PointNet is a unified architecture that directly processes the input point clouds without converting them first to intermediate-representation. The main idea is to make use of max pooling, which is a single symmetric function, allowing the network to learn how to select informative points. These are then aggregated into a global representation that can be used for different tasks. An overview of the architecture can be seen in Figure 3.1. It is important to note that initially, PointNet was not developed as a 3D object detector, but rather as a classifier or part of segmentation network. It has only afterwards been extensively used as a feature extractor, due to its ability to process point clouds effectively.

Shortly afterwards, VoxelNet [ZT] was introduced. VoxelNet propose an end-to-end generic 3D detection network, that unifies the feature extraction and bounding box prediction steps in a single stage. This made manual feature engineering unnecessary. First, the point cloud is transformed into an equally spaced voxel grid. This is then processed using multiple stacked voxel feature encoding (VFE) layers, allowing the network to learn complex 3D shapes in which point-wise features are combined with locally aggregated features. The resulting high dimensional feature representation of the points is then fed into a region proposal network (RPN) based on Faster-RCNN [Ren+], to output the detection results.

In 2018, PointPillars [Lan+] introduced a new method that achieved significant gains in performance and speed compared to other state-of-the-art models at the time. The idea behind PointPillars is to convert the 3D point cloud input into vertically stacked pillars that are processed by simplified PointNet [Qi+a] encoding layers, allowing it to be processed by 2D convolution architectures. The resulting features are scattered, back forming a 2D pseudo

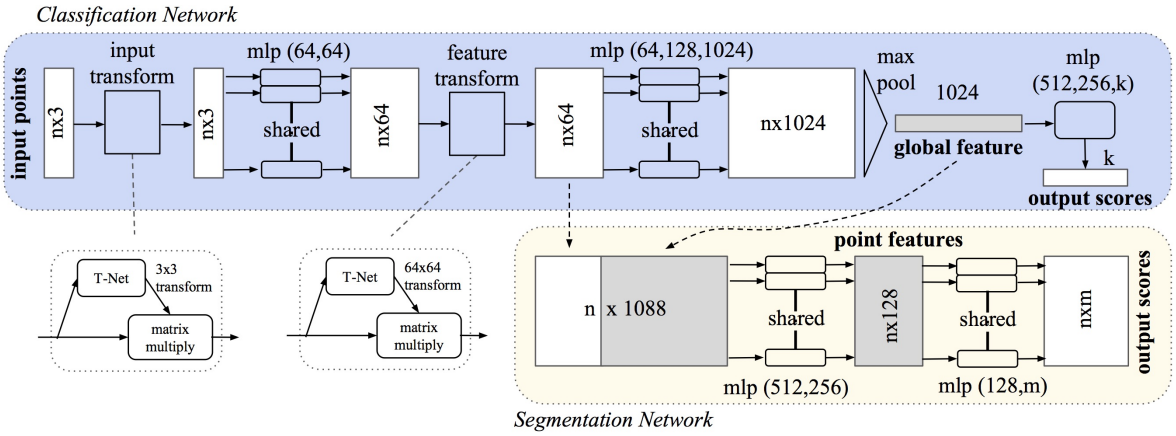

Figure 3.1: PointNet architecture. Input and feature transformations are applied to n input points, the resulting features are then aggregated using max-pooling [Qi+a].

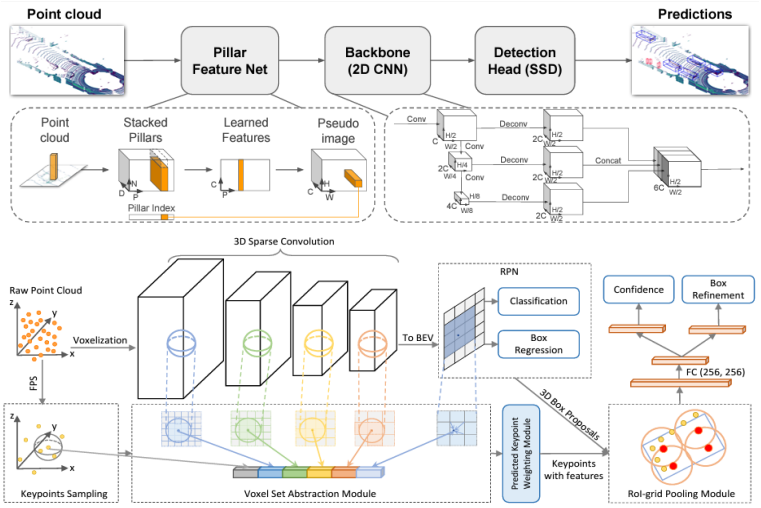

Figure 3.2: PointPillars architecture overview [Lan+] (top). PV-RCNN architecture overview [Shi+] (bottom).

image. The 2D pseudo images are further processed, and fed into a detection head based on Single Shot Detector (SSD) [Liu+15], a 2D detection network. Since the detection head is originally developed for 2D detection, it had to be adapted to regress 3D bounding boxes. The head matches predicted boxes with the ground using 2D Intersection over Union (IoU), and uses the bounding boxes' height and elevation as additional regression targets. An overview of the method can be seen in Figure 3.2.

PV-RCNN [Shi+] deeply integrates point-based and voxel-based methods to combine the advantages of both approaches. Voxel-based methods, that leverage sparse convolutions, are generally efficient and can produce high-quality proposals, while point-based methods are more flexible and can deal with irregularity and are thus contextually more aware. In order to achieve that, the paper proposes a voxel-to-keypoint operation to aggregate all the voxels to a selected number of feature keypoints, and a point-to-grid operation to convert these feature keypoints to region of interest (RoI) grids to refine the previous object proposals. The pipeline of PV-RCNN can be seen in Figure 3.2.

Sparse-to-Dense 3D Object Detector (STD) [Yan+b] presented a two-stage detector. The first stage is a bottom-up proposal generation network where a spherical anchor is seeded for each point individually, then processed by a PointNet++ [Qi+b] backbone. The sparse fea-

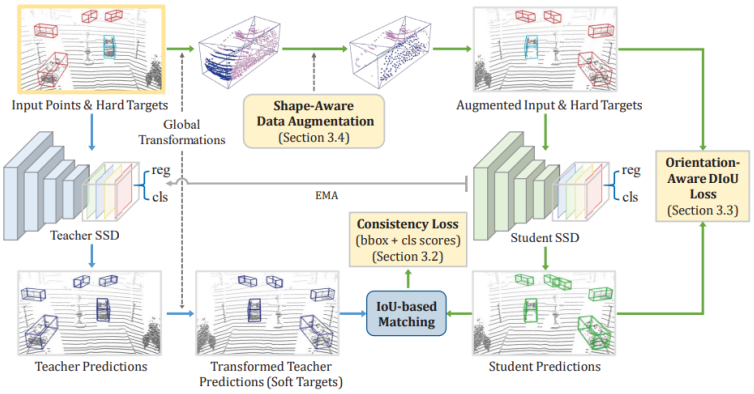

Figure 3.3: Overview of the SE-SSD model depicting the student-teacher architecture and the formulated losses [Zhe+].

tures are then converted into a more compact form to increase the efficiency of convolutional operations. The second stage has two branches for box and IoU estimations to further boost performance, and to alleviate inappropriate removal during post-processing.

Instead of using anchors, CenterPoint [YZKa] predicts objects' centers. In the first stage of the two-stage detector, a class-specific heat map along with other properties such as size and rotation are predicted from features processed by a standard point-based or voxel-based backbone. The first stage is based on the 2D object detector CenterNet [Dua+]. The predictions are then further refined in the second stage using point features at the 3D centers of each face of the estimated 3D bounding box.

In 2021, SE-SSD [Zhe+] was introduced, leveraging a student-teacher architecture based on single stage SSD [Liu+15] models. The student model is supervised using soft targets provided by the teacher, as well as the manually annotated hard targets to optimize the model jointly. In order to achieve this, multiple losses are used. The authors formulate a consistency loss to reduce prediction misalignment between the student and the teacher, in addition to an orientation-aware distance-IoU (ODIoU) loss between the student predictions and the hard targets, as can be seen in the model overview in Figure 3.3. The models are trained in an end-to-end supervised manner, while only the student model is used during inference.

The last couple of years saw an increase in the use of attention layers [Vas+] allowing models to selectively focus on more important features. VISTA [Den+] propose a plug-and-play multi-view fusion module that additively fuses bird-eye-view (BEV) and range view (RV) features in a global spatial context. The multi-view approach provides richer and more comprehensive information than single-view methods to the detection head, allowing for an improved performance. The high-level architecture of VISTA can be seen in Figure 3.4. D-Align [Lee+] is another module that uses attention layers in order to align BEV features over time. As a plug-and-play module that can be used with any BEV feature extractor, D-Align boosts the performance of 3D detectors by utilizing the spatio-temporal information in a sequence of frames.

Recently, MDRNet [Hua+22] was proposed to alleviate the information loss during dimensionality reduction while constructing BEV feature maps. The model retains rich 3D information in the BEV, in which the 3D voxel features and BEV features are fused along different scales. Figure 3.4 provides an overview of this architecture. VoxelNext [Che+a], which was also recently proposed, achieves a boost in efficiency and performance by using a voxel-to-object direct approach. This allows it to directly predict 3D objects from voxel features, with fully sparse convolutional networks, without the need of anchor proxies, sparse-to-dense

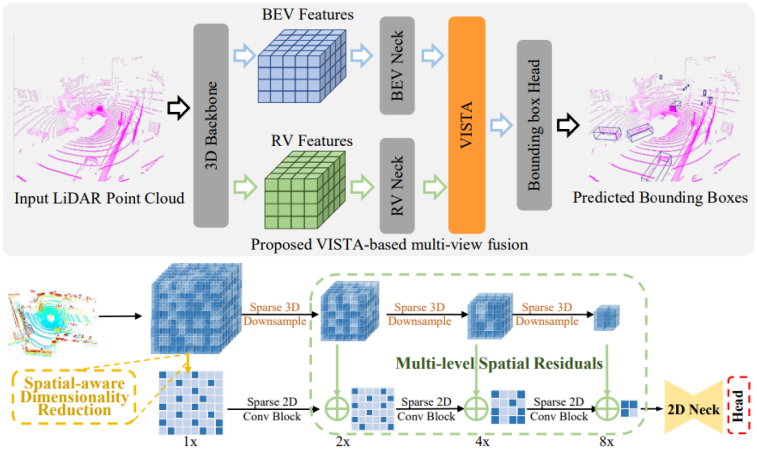

Figure 3.4: VISTA module [Den+] (top). MDRNet architecture overview [Hua+22] (bottom).

conversions or region proposal networks.

3.2 Camera-based 3D Object Detection

In the past, 3D object detection using cameras was challenging due to the lack of depth information. However, this changed with the increasing availability of high-resolution cameras, and the advances in deep learning techniques. Some state-of-the-art multi-view camera-based 3D Object detectors are briefly presented in this section, as they provide a good basis for the following sections.

ImVoxelNet [RVK] is a 3D object detector for monocular or multi-view camera images. The model accepts an arbitrary-sized set of images and their corresponding camera poses. These input images are processed by a 2D backbone network for feature extraction. The extracted features are then projected onto a 3D voxel volume, in which the projected voxel features from the different cameras are fused together using simple element-wise averaging to produce a unified voxel representation. The 3D voxel features are passed into a 3D "neck" network for further processing, before being fed into a detection head. It is worth mentioning, that the method is not dependent on a certain backbone or head architecture, and therefore provides flexibility according to the desired task.

Similarly, authors of Lift, Splat, Shoot [PF] propose a method that extracts a 3D bird-eye-view representation of the scene, based on an arbitrary number of cameras. The method can be used for several tasks such as detection, segmentation or motion planning. In the first step, a categorical distribution is predicted over the depth as can be seen in Figure 3.5, in order to transform the images from the local 2D coordinate system to the global 3D coordinate system. The 3D information is then processed following the PointPillars [Lan+] architecture, allowing the attachment of any compatible task-specific head.

In 2021, the authors of DETR3D [Wan+] introduced a 3D object detector that follows a top-down approach in which the 3D bounding boxes are predicted directly from the 2D image features bypassing the need for the intermediary depth estimation step. In the model, 2D features extracted from the images are linked with the 3D predictions using geometric back-projection given the cameras' transformation matrices. Initially, a sparse set of priors are learned in an end-to-end manner across the target dataset, then the locations of the object priors are back-projected into each camera view. The corresponding 2D features are then extracted using a 2D backbone network. A multi-head attention layer and multiple

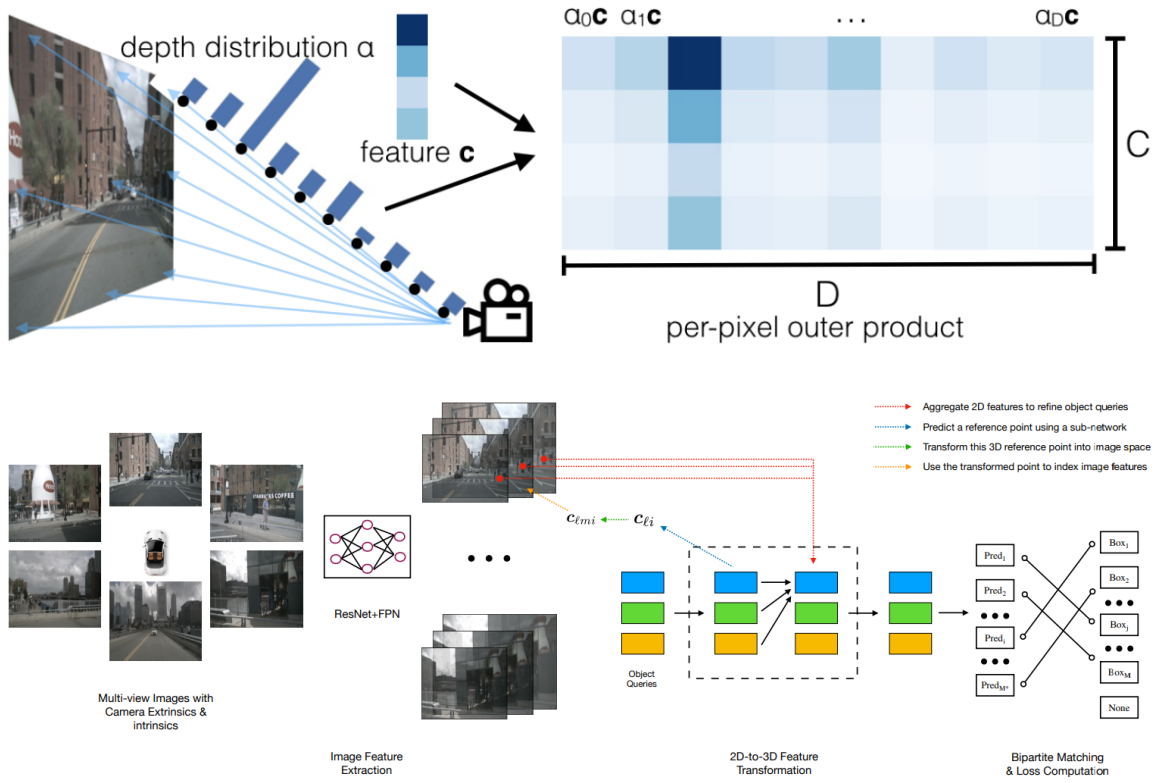

Figure 3.5: Depth prediction step in Lift, Splat, Shoot [PF] (top), for each pixel p, a context feature c and a depth distribution α are predicted, features at each point along the ray are determined by the outer product of α and c. Model overview of DETR3D [Wan+] (bottom).

self-attention layers are used to extract the 3D bounding boxes parameters while using a loss inspired by DETR [Car+]. Ego3RT [Lu+] is also a 3D object detector that follows a top-down similar to DETR3D [Wan+]. The authors propose an end-to-end formulation to construct dense BEV representations inspired by the ray-tracing principle widely used in computer graphics.

The authours of SpatialDETR [Dol+23] build upon DETR3D [Wan+]. They argue that the DETR3D formulation is not sufficient since it only uses the pixels corresponding to the query center while ignoring feature patches from other camera views. To address this issue, the authors propose to use a global cross-sensor attention module that can compare object queries with keys corresponding to feature patches in the different camera images. Additionally, they propose a new geometric positional encoding motivated from the success of pseudo-LiDAR methods. It is worth mentioning that the authors suggest that their method can be scaled to include different sensor modalities such as LiDARs or radars.

The authors of BEVDET [Hua+a] propose a model that performs 3D object detection in the BEV. For the development of the model, the authors reuse existing modules from previous works and managed to achieve state-of-the-art performance on the Nuscenes [Cae+] test set. High-level features are first extracted from the input images using a 2D backbone network such as ResNet [He+] or the attention-based SwinTransformer [Liu+b], followed by a Feature Pyramid Network (FPN) to further process the image features at multiple scales. To transform the 2D features into the BEV, the authors use the view transformer module based on Lift, Splat, Shoot [PF], which can be seen in Figure 3.5. The BEV encoder, which is similar to the initial image encoder, processes the BEV features, then passes them to a 3D detection head based on CenterPoint [YZKa]. To increase the performance of the model, the authors

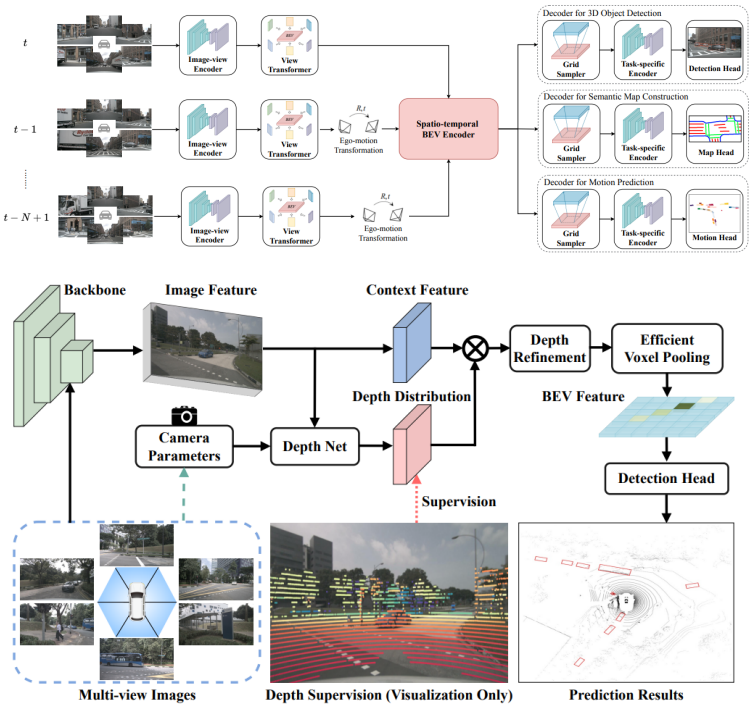

Figure 3.6: An overview of BeVerse [Zha+b] fusing information from previous frames (top). The framework of the depth-guided BEVDepth [Li+d] (bottom).

also introduce a further augmentation step in the BEV space and adapt Non-Maximum Suppression (NMS) for the 3D object detection scenario.

BEVFormer [Li+e] extended the use of transformers by taking not only the spatial information, but also the temporal information into account. The model learns a unified BEV representation of the scene, which can be then used in multiple tasks, similar to LSS [PF]. The authors propose two types of attention mechanisms, the first one is a spatial cross-attention in which each BEV query is able to extract spatial features from RoIs across different camera views. The second one is a temporal self-attention that can extract information based on previous BEV features, aiding in detection of occluded objects. The resulting unified BEV representation can be used for detection as well as segmentation.

BEVerse [Zha+b] incorporates a similar approach as BEVFormer by fusing the spatio-temporal information from multiple camera views to create a BEV representation. An image view encoder similar to BEVDET's [Hua+a] based on SwinTransformer [Liu+b] is used to extract the features from the input images. They also adopt the view transformer module from Lift, Splat, Shoot [PF] to create the BEV representation. An additional spatio-temporal encoder is used to fuse the BEV features over N timestamps with ego motion taken into consideration. This can be seen in Figure 3.6. A task-specific head is then used to predict the 3D bounding boxes.

BEVDepth [Li+d] and CrossDTR [Tse+] approach the problem by using depth-guidance as an auxiliary supervised task. BEVDepth [Li+d] back-projects the LiDAR generated point clouds to create 2.5D ground-truth depth maps. The depth maps are then used to train the camera-aware depth prediction module which takes the respective camera's intrinsic and extrinsic parameters as an additional input. The predicted depth is further refined and converted into a BEV feature map using voxel pooling. The pipeline of BEVDepth can be seen in Figure 3.6. CrossDTR [Tse+] on the other hand, uses a lightweight depth predictor supervised by an object-wise sparse depth map generated from each camera view. The depth

predictor produces depth embeddings that are fed into a cross-view attention module inspired from [Liu+a; Zha+a], along with the 2D images features from the respective camera views, combining cross-view and cross-depth attention mechanisms.

3.3 Multi-modal 3D Object Detection

LiDAR-based 3D object detection methods have generally achieved better performance than camera-based methods. However, as mentioned before in Subsection 2.1.3, each sensor modality has its advantages and disadvantages. Therefore, it is beneficial to combine the information from both sensor modalities to achieve better performance. In this section, we will discuss the different approaches that have been proposed to fuse the information from camera and LiDAR sensors.

MVX-Net [SZT] propose an early-fusion approach based on VoxelNet [ZT]. Semantic features from an RGB image are used to enhance the LiDAR features and provide more contextual information. The model developed two fusion techniques to achieve this; PointFusion and VoxelFusion. In PointFusion, points from the Lidar point cloud are projected onto the image plane, and then concatenated with the corresponding image features produced by a 2D feature extractor based on Faster-RCNN [Ren+]. The produced features are concatenated to the LiDAR point features and then fed into a set of VFE [ZT] layers to produce the final 3D features. In VoxelFusion, features are combined at a relatively later stage when compared to PointFusion. Features from the RGB image are appended at the voxel level. The first stage involves dividing the 3D space into equally spaced voxels and encoding each voxel using VFE layer. Every non-empty voxel is projected onto the image plane to produce a 2D region of interest, and features within the RoI are pooled and reduced, and then appended to the feature vector produced by the stacked VFE layers at every voxel. The fused features are further used by a 3D RPN to produce 3D bounding boxes.

In contrast to MXV-Net, the authors of ContFuse [Lia+a] propose a continuous fusion approach that fuses image and LiDAR features at different levels. The end-to-end architecture consists of two streams; an image stream and a LiDAR stream. In the image stream, a lightweight ResNet18 [He+] backbone is used to extract image features. These features are then projected on the BEV plane and fed into the fusion layers at each level. In the LiDAR stream, the LiDAR point cloud is first projected onto the BEV plane and encoded at different levels using a similar 2D convolutional backbone. The encoded features are then passed to the fusion layers at each level. Finally, a simple detection head followed by a NMS layer is used to predict the final 3D bounding boxes.

Similarly, Dense Voxel Fusion (DVF) [MHW] also uses a continuous approach to fuse the information from camera and LiDAR sensors. The method uses different pipelines during the training and inference phases. During training, ground-truth 3D bounding boxes are used to generate a foreground 2D mask in the image plane. This foreground mask is generated in the inference phase using the output of any pretrained 2D object detector. The foreground masks are associated with the corresponding LiDAR-generated voxel features at different levels. A parameter-free weighting function is used to fuse the features from the two modalities. The fused features are then collapsed onto the BEV plane and fed into a 3D detection head.

MVP [YZKb] takes another approach to fuse the information from the camera and LiDAR sensors. The authors take advantage of the dense representation of the RGB images by converting them into a pseudo-LiDAR representation, creating what they called "virtual points". These virtual points, combined with the LiDAR-generated points are then fed into a 3D object detection network. To generate these virtual points, the LiDAR points are projected onto the image planes and then matched with a 2D detection mask. Randomly sampled 2D points

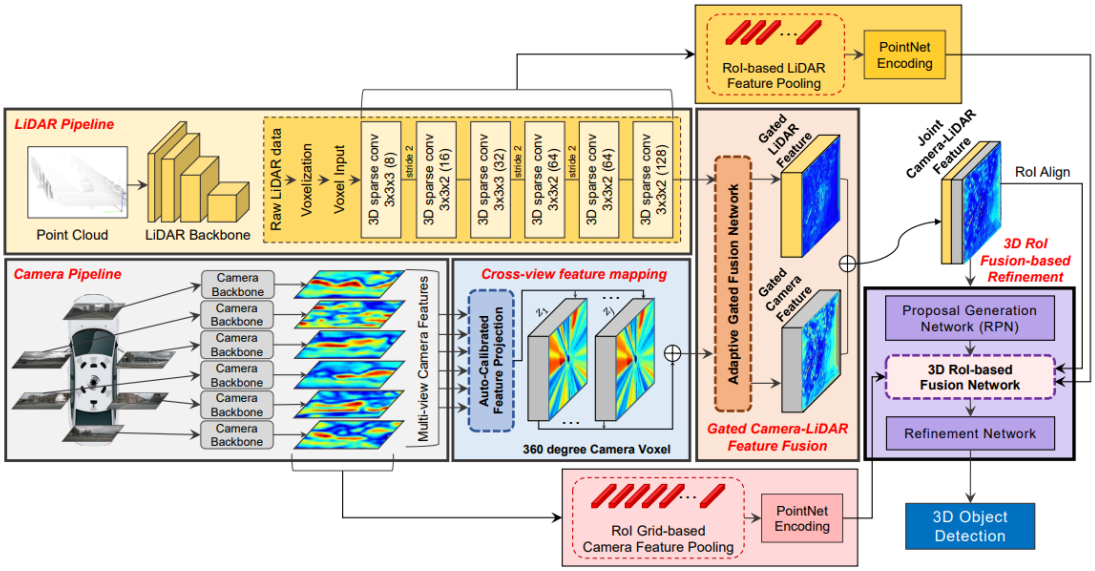

Figure 3.7: Overall structure of 3D-CVF [Yoo+20].

within the mask are appended with depth values from the nearest LiDAR point in the 3D frustum and the objects' semantic features, and then projected back into 3D space, thus creating the virtual points. For the 3D detection, features from the LiDAR points and the virtual points are concatenated and fed into a 3D detection network based on CenterPoint [YZKa].

A similar approach is taken by Sparse Fuse Dense [Wu+], where pseudo-LiDAR points are generated from the RGB images and then combined with the LiDAR-generated points. In the pseudo stream, the LiDAR point cloud is first converted into a sparse depth map in the image space, the depth map is fed into a depth completion network along with the RGB image to produce a denser depth map. The denser depth map is then projected back into the 3D space to produce the pseudo-LiDAR points. The pseudo-points are processed by a specially designed feature extractor that takes advantage of the 2D image features and 3D geometric features of pseudo-point clouds simultaneously. The LiDAR points are voxelized and processed, then fused with the pseudo-LiDAR features in a grid-wise attentive way. A simple detection head is finally used to generate the predictions.

3D-CVF [Yoo+20] is another approach that fuses the information from the camera and LiDAR sensors. The overall structure of the model is shown in Figure 3.7. The model consists of two streams; a LiDAR stream and an image stream. In the LiDAR stream, the LiDAR point cloud is first voxelized before being encoded to produce a feature map in the BEV space. Parallelly, the RGB images are processed by a 2D CNN to produce feature maps in the image space are then projected onto the BEV space using the proposed "Cross-View Feature Mapping" module. Spatial attention maps are applied to the feature maps, to adaptively select the most relevant features from the different modalities and fuse them. Once the region proposals have been created using the combined camera-LiDAR feature map, the RoI pooling technique is utilized for proposal refinement. As the joint feature map still lacks adequate spatial information, multi-scale LiDAR features and camera features are extracted through 3D RoI-based pooling. These features are then encoded independently by a PointNet [Qi+a] encoder and combined with the joint camera-LiDAR feature map via a 3D RoI-based fusion network. The resulting fused feature is ultimately used to generate the final predictions.

Authors of DeepFusion [Li+c] argued that deep fusion of camera and LiDAR features results in better performance when compared with early and late fusion approaches. In Deep

fusion, high-level features from the camera and LiDAR sensors are fused together to generate a joint feature map. The authors propose two techniques that can be used to fuse 2D and 3D features from generic feature extractors. The two techniques deal with common misalignment between camera and LiDAR features when fused together. *InverseAug* inverses the geometric-related data augmentation applied to the inputs before fusing the features. While *LearnableAlign*, utilizes a cross-attention mechanism to dynamically correlate the features from the two modalities. Extracted features are fed into the fusion block. The resulting fused features can be processed by any 3D object detection head.

Interestingly, two concurrent works named BEVFusion have been proposed in [Liu+c] and [Lia+b]. Both works propose deep fusion approaches that fuse the information from the camera and LiDAR sensors in the BEV space. In [Liu+c], the authors used a modified view-transformer based on LSS [PF] to transform the multi-view image features into the BEV space while achieving a huge speedup in comparison with the original implementation. The transformed image features are fused with the LiDAR features, produced by an encoder, using a BEV encoder-decoder network, after which they are fed into a 3D detection head. In [Lia+b] on the other hand, the authors transform the multi-view image features into the 3D space first, before transforming them into the BEV space. They also fuse the image features with the LiDAR using an attention-based dynamic fusion module. In the fusion module, channel-attention is used on the concatenated BEV features.

UVTR [Li+a] also fuses the information from the camera and LiDAR modalities, after converting them into a unified representation. In contrast to BEVFusion [Liu+c] and [Lia+b] methods, UVTR's unified representation is in the 3D space, rather than the BEV space. The multi-view images are first processed by a 2D backbone, then depth is predicted for each image using a simple depth estimation network. The features, along with the predicted depth maps, are then transformed into the 3D space using a view transformer. Having the image features in the 3D space, the LiDAR point cloud is also voxelized. The voxelized features from both modalities are separately encoded by a voxel encoder and fused together using a single convolution to create the unified 3D representation. To enhance the features of each modality, a knowledge-transfer module is utilized to transform knowledge from the geometry-rich modality to the geometry poor taking advantage of the unified representation. Finally, the unified features are passed through a 3D detection head based on DETR [Car+].

AutoAlignV2 [Che+b] follows a different approach to fuse the information from the camera and LiDAR sensors. First, the input images are processed by a ResNet lightweight backbone [He+], then are passed through a feature pyramid network to obtain the feature maps. Next, a learnable alignment map is utilized to aggregate relevant information from the images to enhance the 3D representations of non-empty voxels during the voxelization phase of the 3D LiDAR inputs. Finally, the improved features are fed into the subsequent 3D detection pipeline, which generates the final bounding box predictions. The proposed feature alignment map reduces the sampling candidates when compared to other attention-based methods and dynamically decides the key-point regions on the image plane for each voxel query feature.

3.4 Model-Based 3D Occupancy Estimation

Before the deep learning era, model-based occupancy grids were used to represent the environment in a probabilistic manner. While there are plenty of 2D model-based occupancy estimation approaches, we are particularly interested in 3D model-based methods. The 3D occupancy grid is a 3D grid that represents the environment as a set of voxels, where each voxel can be either occupied or free. The occupancy grid is usually represented as a 3D data

structure, where each element in the structure represents a voxel in the 3D space.

Octomap [Kai+10] is a popular representation of 3D occupancy grids. The approach is based on an octree data structure, where the surrounding space is represented as a set of different size voxels in a tree-like structure. The approach explicitly represents surrounding space as a mixture of occupied, free, and unknown voxels. The probability of a node being occupied, given the sensor measurements, is estimated using an update formula. This formula depends on the current measurement, a prior probability, and the previous estimate. The probability of a voxel being occupied is specific to the sensor that is used. The update rule is based on a uniform prior probability assumption. When using the 3D map, a threshold is applied to determine occupancy. To adapt the map to changes in the environment, a clamping update policy is proposed, which sets upper and lower bounds on the occupancy estimate. This policy limits the number of updates needed to change the state of a voxel and ensures bounded confidence in the map.

GPOctoMap [WE16] and BGKOctoMap [DWE17] are two methods that further build on Octomap [Kai+10]. GPOctoMap [WE16] combines gaussian process (GP) regression and binary classification functions to estimate the probability of occupancy for each map cell. The authors argue that the use of covariance functions captures variations in occupancy more effectively. Additionally, the authors introduce a new data structure called test-data octrees, which is more effective than octrees by pruning nodes with the same state. BGKOctoMap [DWE17] uses Bayesian non-parametric inference and sparse kernel techniques to maintain a predictive distribution of occupancy states in 3D maps. It allows for exact inference and updates and provides mean and variance estimations. Additionally, it categorizes cell states based on occupancy probabilities and variance thresholds. The computational efficiency of the algorithm is achieved through a sparse kernel and recursive updates, enabling the generation of maps offline or incrementally.

Another method was proposed in [HUD09], which represents the environment in the form of stixels based on stereo cameras. Stixels are a set of rectangular sticks, where each stixel is defined by its 3D position relative to the camera and stands vertically on the ground, having a certain height. An example of stixel representation is shown in Figure 3.8. Initially, the method computes the disparity map from the stereo images using dense stereo matching. An occupancy grid is then generated based on the stereo disparities. This grid represents the likelihood of occupancy for different regions in the scene. To identify the free space in front of the obstacles, dynamic programming is used to find an optimal solution while taking spatial and temporal clues into consideration. To further enhance the accuracy of the free space boundary, a background subtraction step is performed. After that, the height of the obstacles is estimated through segmentation between foreground and background disparities. Finally, stixel extraction is performed to obtain a precise 3D model of dynamic objects in the scene.

Multi-Volume Occupancy Grids (MVOGs) [IWJ10] are another method to represent spatial occupancy information. MVOGs utilize a 2D grid composed of square cells in the xy-plane to represent the environment. Each cell contains two lists of volumes: positive volumes representing obstacles and negative volumes representing free space. Volumes are defined by their height, occupancy mass, and occupancy density, where the occupancy mass represents the amount of sensory information observed, while the density represents the information per unit space. The update of an MVOG using laser data involves three steps: rasterization, creating new volumes, and applying constraints. Rasterization involves determining which grid cells are intersected by laser readings. New volumes are then created based on the height information of the laser rays, and positive and negative volumes are inserted into the corresponding cell lists. Constraints, such as minimum volume size and no overlap between volumes, are enforced to ensure the validity of the volume lists. Occupancy probabilities are calculated based on the density of positive and negative volumes containing a point.

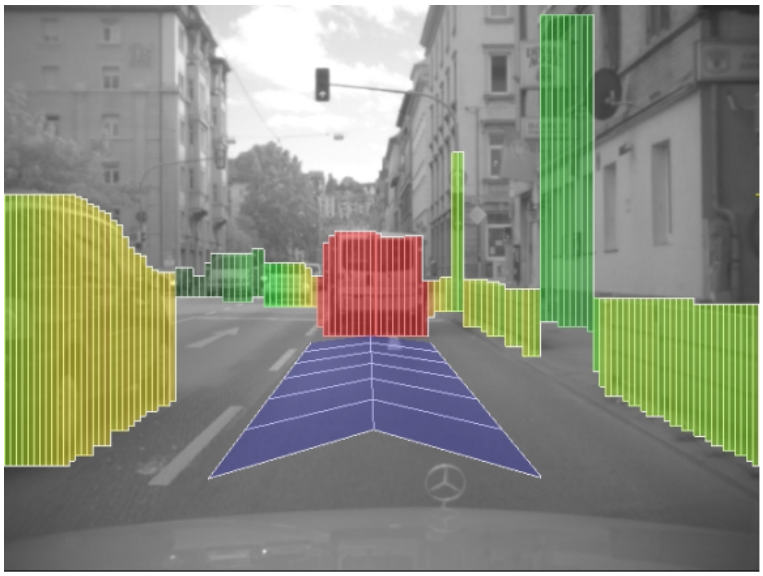

Figure 3.8: Stixel representation of 3D environment [HUD09].

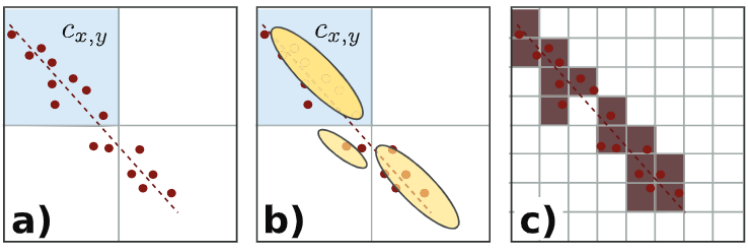

Figure 3.9: An example of NDT representation compared to a normal occupancy grid [Saa+13a]. (a) The measurement points are placed into grid cells and (b) The distribution parameters are computed for each cell for which there are measurement points. (c) An example of an occupancy grid using the same measurement points.

Several other approaches [DWE16; GR18; GSRuu] use 3D Hilbert maps to represent the environment. Hilbert maps model the environment by projecting spatial coordinates into a high-dimensional feature representation known as a Hilbert space. Linear separation methods, such as logistic regression classifier, can be then used to classify the occupancy of the environment.

Normal Distributions Transform (NDT) occupancy maps are also used to represent the environment, as in [Mar09; Saa+13a; Saa+uu; Saa+13b]. The methods use a probabilistic approach to represent the environment, in which 3D range points are represented as a set of Gaussian probability distributions. An example can be seen in Figure 3.9.

3.5 Learning-Based 3D Occupancy Estimation

Learning-based 3D occupancy estimation methods have been proposed in the literature for different use cases. These use cases are not limited to autonomous driving but also include object and scene reconstruction and completion. In this section, we will discuss some of the most relevant methods in the literature.

One of the first learning-based methods for scene completion was proposed in [Son+]. The semantic scene completion network (SSCNet) propose an end-to-end 3D CNN that takes a single depth image as input and outputs a completed 3D voxelized scene, in the form of

occupancy and semantic labels for all voxels in the camera view frustum. The input depth image is first transformed as a 3D volume, which is then fed into a set of 3D convolutional layers, enabling the network to learn a local geometric representation of the scene. To reduce the resolution to one-fourth of the original input, convolution layers with stride and pooling layers are utilized. Following this, a dilation-based 3D context module is employed to capture higher-level inter-object contextual information. The network then combines the responses from different scales by concatenating them, and passing them through two additional convolution layers to aggregate information. Finally, a voxel-wise softmax layer is used to predict the final voxel label.

DeepSDF [Par+] models shapes as the zero iso-surface decision boundaries of feed-forward networks trained to represent Signed Distance Functions (SDFs). An SDF is a continuous function that provides the distance from a given spatial point to the nearest surface, with its sign indicating whether the point is inside or outside the surface. In the paper, the authors suggest directly regressing the continuous SDF using deep neural networks, enabling the prediction of SDF values for query positions and extraction of the zero level-set surfaces. To train the model, the authors use a dataset of 3D shapes represented as triangle meshes, in which they compute the SDF values for a set of points through a distance transform for the watertight meshes. The resulting network represents the shape's surface as the zero iso-surface of the learned SDF.

The work Occupancy Networks [Mes+] was introduced to reason about the occupancy of 3D objects at any given point without being bound by the limitations of voxel representations. To address this, the authors propose a neural network that assigns an occupancy probability between 0 and 1 to each point in 3D space, effectively approximating the desired occupancy function. The authors model the problem as a binary classification network, which focuses on the decision boundary that implicitly represents the surface of the object. Where objects can be reconstructed from an observation, such as an image or a point cloud. The network approximates a function that outputs the probability of a point being occupied, conditioned on the observation.

Several works [Son+20; Lio+] have extended the occupancy networks framework to improve the reconstruction quality and enable scene reconstruction instead of being limited to single objects. In Convolutional Occupancy Networks [Son+20], the input point cloud is first encoded into a 2D or 3D feature grid. These features are additionally processed using convolutional networks and then decoded into occupancy probabilities using a fully connected network. Contrary to the original Occupancy Networks, the proposed method exploits convolutional operations, resulting in a scalable and translationally equivariant implicit representation.

LMSCNet [RCV] introduced an end-to-end learning-based method, that performs multiscale semantic completion and occupancy estimation of a 3D environment based on sparse point clouds. The method is based on a U-Net-like architecture, which enhances feature and gradient flow due to the multiscale skip connections. To further reduce the computational cost, the authors propose using a 2D U-Net instead of 3D, where 2D convolutions are used along the xy plane while turning the height dimension into a feature dimension. 3D segmentation heads are appended at multiple scales, where the dimensions of the inputs are expanded again to the 3D space, and fed into 3D convolutional layers.

S3CNet [Che+c] takes another approach to solve the semantic scene completion task using sparse point clouds. The method involves the construction of two sparse tensors from the sparse point cloud input, to represent the scene in 2D and 3D. The 2D sparse tensors represent a set of non-empty pillars approximating the point distribution along the xy plane (BEV). The tensors are then fed into their respective semantic scene completion networks, 2D S3CNet and 3D S3CNet, to complete the scene with semantic information. To address

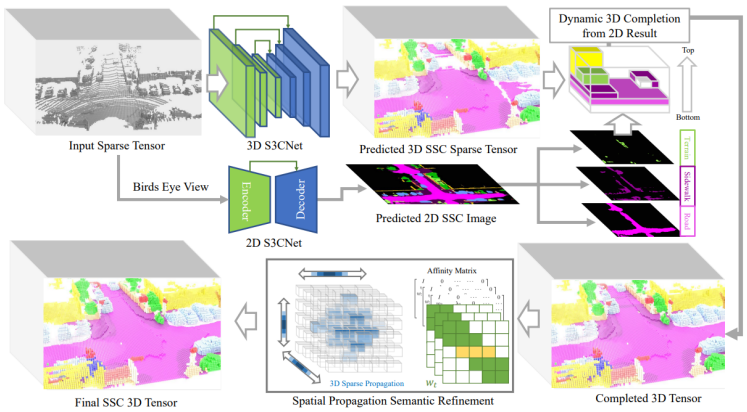

Figure 3.10: An overview of the S3CNet architecture [Che+c].

memory demands on the 3D network caused by exponential sparsity growth, a dynamic voxel fusion method is proposed, which densifies the reconstructed scene using a predicted 2D semantic BEV map. A sparse tensor spatial propagation network is utilized to refine the semantic labels in noisy regions of the fused 2D-3D predictions. The complete architecture can be seen in Figure 3.10

JS3C-Net [Yan+a] is also a semantic scene completion network that operates on point cloud data. It consists of three main components: Semantic Segmentation, Cascaded Semantic Scene Completion (SSC), and Shape-aware Point-Voxel Interaction (PVI). The Semantic Segmentation module takes a point cloud as input and uses sparse convolutions to perform semantic segmentation. It converts the input into voxel grids and applies convolution operations efficiently by storing only non-empty voxels. The resulting voxel-based output is then transformed back to point-wise features using nearest-neighbor interpolation. To incorporate shape priors, the features are further processed through multi-layer perceptions (MLP) to obtain shape embeddings (SE). These shape embeddings are fused with the features from the SparseConv UNet and the final output is obtained through MLPs. The SSC module aims to complete the scene by leveraging contextual shape priors from the entire LiDAR sequence. It takes the semantic probability output from the semantic segmentation component and predicts the completion results. The module voxelizes the input point cloud to obtain a high-resolution 3D volume and then uses convolution and pooling layers to reduce complexity. The PVI module further enhances the completion results by combining the incomplete point clouds and the complete voxels obtained from the SSC module.

Local-DIFs [Ris+] uses implicit functions to obtain a continuous scene representation that is not based on voxelization, contrary to other methods. The implicit function maps 3D positions in a scene to a probability vector representing the semantic class of the position. The function incorporates both geometric and semantic segmentation of space, including objects and free space. The overall function is constructed from local functions, each with its own coordinates of interest and parameterization vector. A grid structure is used to encode spatial structure in the latent space, and an hourglass convolutional encoder generates feature maps for the grid. The composition of the global completion function involves multiple conditioning vectors and grid resolutions. Bilinear interpolation is used to obtain the final interpolated classification result based on the query coordinates.

MonoScene [CC] tackles the scene completion task by using a single RGB image as input. The proposed approach uses a pipeline that combines 2D and 3D UNets with a Features Line of Sight Projection module (FLoSP) and a 3D Context Relation Prior component (3D CRP). The FLoSP module lifts multi-scale 2D features to plausible 3D locations, allowing the 3D network to leverage high-level 2D features for 3D disambiguation. It projects 3D voxel

centroids to 2D and samples corresponding features from the 2D decoder feature map. These features are then aggregated into a single 3D representation that serves as input to the 3D UNet. To capture long-range semantic context, the 3D CRP component is inserted between the 3D encoder and decoder. It learns voxel-to-voxel semantic scene-wise relation maps, providing the network with a global receptive field and increasing spatio-semantic awareness. The component considers bilateral voxel-to-voxel relations grouped into free and occupied categories, disregarding voxel semantics. The outputs of the 3D UNet are fed afterwards to a completion head, which outputs the final semantic labels in the desired resolution.

In 2022, Tesla Inc. presented its approach for 3D occupancy prediction [Tes22b; Tes22a], while referencing the work of [Son+20]. Although Tesla has not published any paper, the approach has gained a lot of attention in the scientific community. Motivated by this, several datasets [Tia+; FAN23; Ope23] and approaches tackling the 3D occupancy prediction task have been published the following year, well after the start of this work. In the following, we will briefly describe the most relevant approaches that were recently published.

TPVFormer [Hua+b] propose a method for 3D occupancy prediction that uses a transformer architecture to map 2D image features into a 3D planar representation similar to [Son+20]. Multi-view input images are first processed using a 2D backbone, the resulting feature maps are passed into the TPVFormer module. The TPVFormer consists of TPV queries, image cross-attention (ICA), and cross-view hybrid attention (CVHA). TPV queries encode view-specific information from pillar regions, and cross-view hybrid attention allows interactions between TPV queries to gather contextual information. The TPVFormer also includes hybrid-cross-attention blocks (HCAB) and hybrid-attention blocks (HAB) for querying visual information and encoding contextual information, respectively. The planar features for each query location are aggregated and then passed to a lightweight MLP to predict the semantic occupancy.

VoxFormer [Li+b] is also a transformer-based approach to learning 3D voxel features from 2D images for Semantic Scene Completion (SSC). The architecture involves extracting 2D features from RGB images and employing a sparse set of 3D voxel queries to access these 2D features, linking 3D positions to the image stream using camera projection matrices. The voxel queries are learnable parameters shaped like a 3D grid, designed to query features within the 3D volume from images using attention mechanisms. The framework consists of two stages: stage one generates class-agnostic query proposals, and stage two utilizes an architecture similar to MAE (masked autoencoder) to propagate information to all voxels. The voxel features are then up-sampled for semantic segmentation to match the desired resolution.

The pipeline of SurroundOcc [Wei+] starts with a 2D backbone network, such as ResNet-101, to extract multi-scale features from N cameras and M levels. For each level, a transformer is utilized to fuse these multi-camera features with spatial cross-attention. The output of the 2D-3D spatial attention layer is a 3D volume feature instead of a BEV feature. A 3D convolution network is applied to upsample and combine the multi-scale volume features. The occupancy prediction in each level is supervised using dense occupancy ground truth with a decayed loss weight. Additionally, the authors propose a pipeline to generate dense occupancy ground truth, leveraging existing 3D detection and 3D semantic segmentation labels without extra human annotations, after arguing that a network supervised by sparse LiDAR points is unable to predict dense occupancy.

SimpleOccupancy [Gan+] and OccFormer [ZZD] are two other approaches that use transformers their architectures. SimpleOccupancy uses a transformer-based encoder to predict only the 3D occupancy without semantic labels. The approach uses a shared 2D backbone to extract image features from multiple views. A parameter-free interpolation is then used to obtain the initial 3D volume, while an hourglass 3D convolution network with position-prior

guidance is used to effectively aggregate the 3D feature in the volume space. The resulting volume space can be used to predict the occupancy using simple binary classification.

Chapter 4

Methodology

In this chapter, the proposed approach is presented in detail. The proposed approach is composed of two main parts: synthetic data generation and 3D occupancy prediction. In Section 4.1, the proposed approach for generating synthetic data for the 3D occupancy prediction is presented. In Section 4.2, the LiDAR-based approach is first presented in Subsection 4.2.1. Then, the multimodal-based approach, where inputs from LiDAR and multi-view cameras are described. Finally, the different loss functions used for training the proposed approaches are presented in Subsection 4.2.3. In the last section, Section 4.3, the implementation details of the proposed approaches as well as the hardware and software used for the experiments are presented.

4.1 Synthetic Data Generation

After initial research, it was found that there is no publicly available dataset for continuous 3D occupancy prediction. Therefore, as part of this work, we generated a synthetic dataset for the 3D occupancy prediction task.

The required data should be ideally not bounded to a specific resolution, this way a continuous representation of the environment can be learned. CARLA simulator, which we briefly introduced in Section 2.5, provides a flexible way to generate synthetic data using different sensors. Using the simulator, data for object detection, semantic segmentation, and depth estimation can be generated. However, the simulator does not provide a direct way to generate 3D occupancy data.

Initially, different approaches for data generation were considered. One approach of which, was to spawn various LiDAR sensors around the ego vehicle, and accumulate the point cloud data over time. Then, the densely accumulated point cloud data can be converted to water-tight meshes or voxel grids, where occupancy values can be queried. However, this approach has several drawbacks. First, the accumulated point cloud data over time will contain dynamic objects such as pedestrians and vehicles, which change their position over time. This would therefore introduce noise to the ground truth data. To mitigate this, probability-based algorithms can be used to filter out these noisy points. However, this would introduce additional computations and complexity to the data generation process. In addition to the dependency on how well these algorithms can filter out the noise, which is not guaranteed. Another drawback of this approach is that the resolution generated data will be dependent on the resolution of the voxel grid, and whether all voxel cells have been observed by the sparse point clouds from the LiDAR sensors. Using meshes would partially solve the resolution problem, however, it would introduce additional computations and errors to the data generation process for generating the meshes for point clouds. To address the sparsity of the

point clouds, additional depth cameras can be used, this way the resolution of the generated data can be increased. However, it does not solve the rest of the aforementioned problems.

Objects in CARLA are rendered using the Unreal Engine. Each object is represented internally as a high-definition mesh, which is then textured and rendered. The LiDAR sensors in CARLA uses these meshes to generate the point cloud data along with Unreal Engine's ray tracing algorithm. Additionally, each object has a collision mesh which is a simplified version of the high-definition mesh. The collision mesh is used for collision detection and other physical simulations.

The **cast_ray** method in CARLA provides the ability to cast a ray from an initial to a final location. The function then detects all geometries intersecting with the ray using the objects' collision meshes, and returns a list of coordinates of all intersections, along with their semantic labels. An example of this is shown in Figure 4.1.

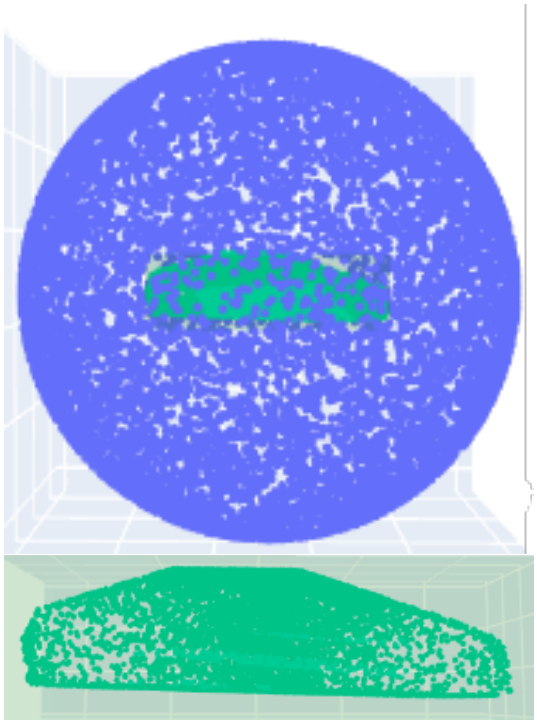

Figure 4.1: An example of the output of **cast_ray** method in CARLA. In the top image, a spherical point cloud is sampled around the ego vehicle. The points are used as an initial location for the **cast_ray** method. The end location is set to be the center of the vehicle. The bottom image shows the returned points with the Car label in CARLA. .

The idea of the proposed approach is to use the **cast_ray** method to generate the ground truth data for the 3D occupancy prediction task. The approach is simple, for a point to be labelled as occupied, it has to be within an object's bounding box, and when a ray is cast from outside the object's bounding box, to the query point, it should intersect with the object's collision mesh in all 6 faces of the bounding box. A simplified example of this in 2D is shown in Figure 4.2.

To generate occupancy data for a whole scene, we spawn first an ego vehicle and attach camera and LiDAR sensors to it. The sensor setup can be flexibly configured using a configuration file. Initially, we spawned the ego vehicle along with a top-mounted LiDAR sensor providing 360° view and a forward-facing camera. However, we extended the setup to include a second forward-facing camera to provide a stereo view analog to the KITTI dataset [Gei+13] sensor setup, and a multi-view camera setup similar to the Nuscenes dataset [Cae+]. Different traffic participants are randomly spawned in the scene, where their positions, orienta-

Figure 4.2: An example of the using **cast_ray** method in CARLA to determine a point's occupancy in a simplified 2D scenario. The left image shows that the point is labelled as occupied since the ray intersects with the object's boundary from both sides while being within the object's bounding box. The right image shows that the point is labelled as unoccupied, since the ray only intersects with the object's boundary from one side, although being inside the object's bounding box.

tions, and velocities are randomly sampled.

For the occupancy points, we uniformly sample n points within a predefined range around the ego vehicle in 3 directions. This was decided since the authors of the original Occupancy Networks paper [Mes+] proved that uniformly sampling points are the most adequate to create a continuous representation of the environment. Using CARLA's API, we retrieve all bounding boxes of the objects in the scene. Points that do not lie within any of the bounding boxes are labelled as unoccupied. For the remaining points, we first assign them to their corresponding bounding boxes, and then cast a ray from the outside of the bounding box to the point, perpendicular to each of the 6 faces of the bounding box. If the ray intersects with the object's collision mesh from all sides, then the point is labelled as occupied, otherwise, it is labelled as unoccupied. An overview of this can be seen in Algorithm 1.

Algorithm 1 Check occupancy

```
def check_occupancy(p, b)
    Require Point p, Bounding box b
    occupancy → true
    for (i → 0; i < 6; i → i + 1)
        labelled_point → cast_ray(b.face[i], p)
        if labelled_point is empty
            occupancy → false
            break
        else
            if (labelled_point is not in b)
                occupancy → false
                break
            else
                occupancy → occupancy&true
    return occupancy
```

Additionally, to have a more dense representation of small objects, we uniformly sample m points within each bounding box, and further check their occupancy using the same approach. The occupancy labels and their coordinates, along with the sensor data and the bounding boxes are then saved. The process is repeated for k frames, where the ego vehicle is put in CARLA's autopilot mode to drive around the scene. This is then repeated for different drives and in different CARLA Towns.

To generate data for the infrastructure setup, we use a CARLA Town that is built to resemble the Providentia++ test bed in Garching [Krä+; Cre+] provided by the chair of Robotics,

Artificial Intelligence and Real-time Systems at the Technical University of Munich. We use the same approach as for the ego vehicle setup, however, we do not spawn an ego vehicle and sensors are spawned at the predefined locations according to their locations in the test bed.

4.2 Occupancy Prediction

In this section, we describe the proposed approaches for 3D occupancy prediction using LiDAR point clouds as well as camera-LiDAR fusion. In the first subsection, we describe the LiDAR-based approach, where we use the LiDAR point clouds to predict the 3D occupancy in a scene based on the original Occupancy Networks paper [Mes+]. The second subsection describes the multimodal approach, where we use the camera and LiDAR data to predict the 3D occupancy in a scene.

4.2.1 LiDAR-based Occupancy Prediction

The first intuition to develop a LiDAR-based occupancy prediction approach in the context of autonomous driving is to adopt the original Occupancy Networks approach [Son+20] and use it as a starting point. An overview of the approach can be seen in Figure 4.3.

In summary, for the model to capture an implicit representation of the environment, 2D and/or 3D feature grids are constructed from the input. The feature grids are further refined using convolutional networks. Finally, the features are decoded into occupancy probabilities using fully connected layers.

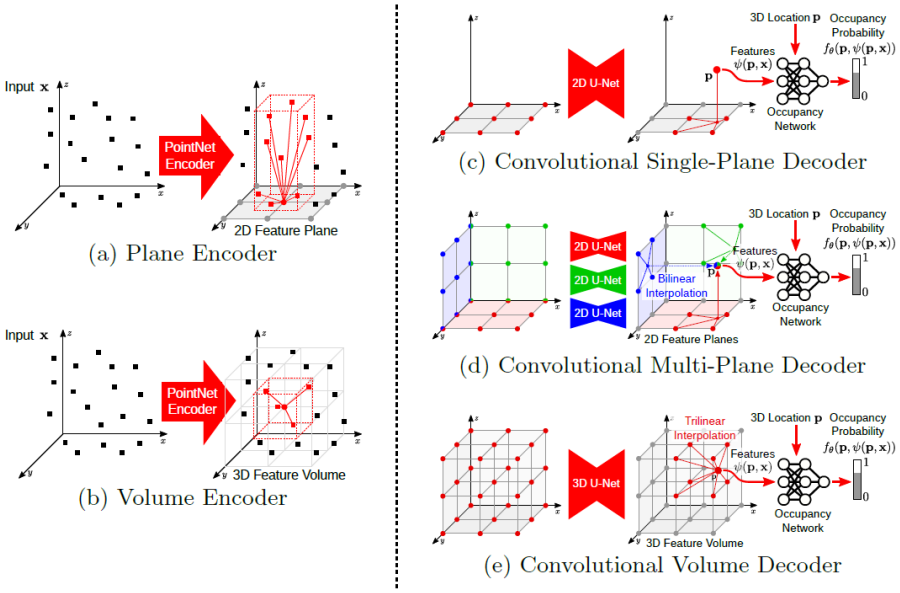

Figure 4.3: Overview of the LiDAR-based occupancy prediction approach [Son+20]. Left: The input point cloud is encoded using a PointNet-based encoder with local pooling and the feature grids are constructed. Right: The feature grids are refined using UNet hourglass networks. The interpolated features for a given query point are then decoded into occupancy probabilities using fully connected layers.

To extract features from the input point cloud, the points are encoded using a shallow PointNet [Qi+a] encoder with local pooling, instead of the originally proposed max pooling.

Additionally, ResNet blocks are used instead of the originally proposed fully connected layers. Since the approach is independent of the encoder architecture, it can be swapped with any point or voxel-based encoder. For this, support for the PointNet++ [Qi+b] encoder is also implemented. The advantage of using a PointNet++ encoder is that it can generally capture higher-level features of the input point cloud, due to its hierarchical structure. The output of the encoder has a dimension of $n \times d$, where n is the number of points in the input point cloud, and d is the feature dimension that can be set by the user as a hyperparameter.

The implicit representation of the environment is then captured in the form of discrete 2D and 3D feature grids. The 2D feature grids are constructed by projecting the encoded features on the canonical xy, xz, and yz planes, based on their respective coordinates. Since the grids are discrete, features projected on the same cell are aggregated using average pooling. The 3D feature grid is constructed analogously, by projecting the encoded features onto a 3D grid, where features falling into the same cell are aggregated using average pooling as well. This results in 2D feature grids of size $H \times W \times d$, and a 3D feature grid of size $H \times W \times D \times d$, where H, W, and D are the height, width, and depth of the grid, respectively. This can be seen in Figure 4.3 parts a and b.

An efficient representation can be constructed by projecting the encoded features onto the ground xy plane only instead of all three planes. The volumetric feature grid can also be used solely, as well as in combination with the 2D feature grids.

The 2D and 3D feature grids are then refined further refined using UNet [RFB] hourglass networks, where the combination of up- and down-sampling layers in addition to skip connections allows the network to capture and integrate both local and global features. Additionally, this step allows the creation of a denser feature grid, where empty cells will be filled with features as a result. The 2D feature grid is refined using a 2D UNet, while the 3D feature grid is processed using a 3D UNet. In the setting where multiple 2D feature grids are constructed, they are processed using a 2D UNet with shared weights. The translational equivariant nature of the convolutional operations allows the network to capture the spatial relationships between the features.

The feature grids and planes are then upscaled to a higher resolution using bilinear interpolation or 2D transpose convolutional layers and trilinear interpolation or 3D transpose convolutional layers for 2D and 3D feature grids, respectively. This step can be necessary especially for the 3D feature grid, since the resolution of the grid is bounded by memory constraints.

Finally, the features are decoded into occupancy probabilities using fully connected layers. Each query point is projected onto the respective 2D or 3D feature grid, and the feature values are acquired using bilinear interpolation for 2D grids, and trilinear interpolation for 3D grids. In the case where multiple feature grids are used, the features are summed up before being passed to the prediction network. The prediction network consists of multiple linear layers, followed by multiple ResNet blocks. The output of the last ResNet block is then passed into a final linear layer, which outputs the occupancy probability for the query point. The process can be seen in Figure 4.3 parts c, d, and e. This step can be modelled as a function f_θ, as shown in Equation 4.1, where $\mathbf{p}$ is the query point, $\mathbf{x}$ is the input point cloud, and ψ is the feature vector representing the input point cloud at the query point $\mathbf{p}$.

Additionally, class labels can be predicted by either using a separate classification branch that takes the interpolated features as input or by using the same prediction branch where a class label is directly predicted instead of an occupancy probability.

$$f_\theta(\mathbf{p}, \psi(\mathbf{p}, \mathbf{x})) \rightarrow [0, 1] \tag{4.1}$$

The model takes a single point cloud as input, in addition to the query points, where both inputs are in the same coordinate system. To scale the model further, and allow it to process

multiple point clouds, the point clouds as well as the query points are first transformed into a common coordinate system as shown in Equation 4.2. The transformation matrix $\mathbf{T}_{w,ego}$ represents the transformation from the world coordinate system to the ego vehicle coordinate system, while $\mathbf{T}_{w,lidar_1}$ represents the transformation from the world coordinate system to the coordinate system of the first LiDAR sensor. $\mathbf{p}_{lidar_1}$ is a point cloud from the first LiDAR sensor, in the LiDAR's coordinate system.

$$\mathbf{p_{ego}} = \mathbf{T}_{w,ego}^{-1} \mathbf{T}_{w,lidar_1} \mathbf{p}_{lidar_1} \tag{4.2}$$

After the point clouds are transformed, they are concatenated together and passed to the model as a single input. The query points are also transformed in the same way. While this is a naive approach, it allows the model to process multiple point clouds without any additional overhead. Another approach would be to process each point cloud separately, after transforming them into the same coordinate system. Separate feature grids and planes would be constructed for each point cloud, and interpolated features for each query point would be summed up and passed to the prediction network.

TODO create a figure for the above paragraph

4.2.2 Multimodal-based Occupancy Prediction

Motivated by the comparison between LiDAR and camera sensors in Section **??**, we propose a multimodal-based occupancy prediction model, where the model takes as input both LiDAR point clouds and multi-view camera images. The model is based upon the UVTR framework [Li+a] originally proposed for 3D object detection. An overview of the model can be seen in Figure 4.4.

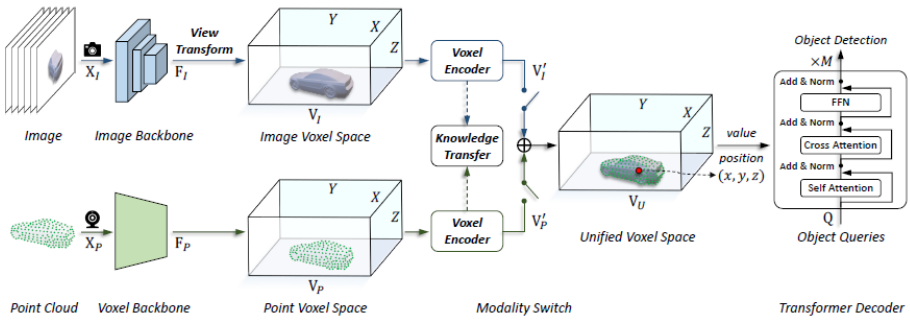

Figure 4.4: Overview of the UVTR model for 3D object detection [Li+a].

The UVTR model creates a unified representation of the input data using a voxel-based representation, this eases the fusion of the different modalities. Initially point clouds and images are processed in separate pipelines: The LiDAR point clouds are first converted into voxels using a VFE [ZT] layer. The resulting sparse voxels are then processed using a sparse convolutional encoder based on the valid sparse convolution (VSC) operators [Gv]. The encoder produces dense feature maps on multiple scales using parallel heads with different strides. The feature maps are then processed using a Second3D [YYB18] backbone and neck, where several convolutions are applied at each scale to aggregate spatial cues. These are then upsampled to the same resolution and summed together to construct the voxel representation for the LiDAR branch $\mathbf{V}_P \in \mathbb{R}^{X \times Y \times Z \times C}$. The Second3D-based backbone and neck additionally allow the accumulation of across multiple LiDAR sweeps.

To create the image voxel space, multi-view images are first processed separately using a single ResNet [He+] backbone followed by a feature pyramid network (FPN) [Lin+] for

multi-scale context aggregation. A depth distribution is predicted for each camera view following the proposed LSS [PF]. The depth distribution $\mathbf{D}_I(u,v)$ is a probability distribution over a set of discrete depth bins D, where u and v are the coordinates in the image plane. The depth distribution is predicted using a single convolutional layer and is not directly supervised. The operation is modelled in Equation **??**, where $\mathbf{F}_I \in \mathbb{R}^{H\times W\times C}$ is the feature map from the FPN, and $\mathbf{D}_I \in \mathbb{R}^{H\times W\times D}$ is the predicted depth distribution.

$$\mathbf{D}_I(u,v) = \text{Softmax}\left(\text{Conv}\left(\mathbf{F}_I\right)(u,v)\right). \tag{4.3}$$

These values are then projected into the voxel space $\mathbf{V}_I \in \mathbb{R}^{X\times Y\times Z\times C}$ using the view transformer, the equation describing the transformation can be seen in Equation 4.4. The point (u,v,d) in the image plane, (u,v) indicates the coordinate in the image plane and is calculated from a point (x,y,z) in the ego vehicle frame with the calibration matrix $\mathbf{P}$. Here d represents the depth value from $\mathbf{D}_I$.

$$\mathbf{V}_I(x,y,z) = \mathbf{D}_I(u,v,d) \times \mathbf{F}_I(u,v) \tag{4.4}$$

Temporal cues can be incorporated into the image voxel space by attaching the relative time offsets with respect to the initial frame along the channel axis and merging the voxel grids using a single convolution.

While knowledge transfer and fusion modules are proposed in the original UVTR framework, we do not use them in our model however they can be easily integrated due to the plug-and-play structure of the model. We instead use the simple proposed fusion module to construct the unified voxel representation $\mathbf{V}_U \in \mathbb{R}^{X\times Y\times Z\times C}$ In this module, the LiDAR and image voxel grids are processed using three convolutional blocks and are aggregated in both coplanar and vertical dimensions. This decision was taken to simplify the model and reduce the number of parameters.

To reap the aforementioned benefits of the hourglass architecture, we further process the unified voxel representation using a lightweight 3D UNet hourglass network, this is to ensure that the model is able to capture both local and global context, and to increase the density of the voxel representation.

Different prediction heads can then be attached after the hourglass network for occupancy prediction. For the continuous occupancy prediction task, a similar prediction head to the one used in the LiDAR-based occupancy prediction model is used. In this setting, query points are projected into the voxel space, and the feature value at the projected point is obtained using trilinear interpolation with the nearest voxels. The feature value is then processed using a linear layer and several ResNet blocks. The output of the ResNet blocks is then fed into a linear layer to predict the occupancy probability at the query point.

The model can also be used for discrete occupancy prediction and scene completion tasks. In this case, the occupancy and class label are predicted for each voxel cell, after bringing the voxel representation to the desired output resolution.

The (X, Y, Z) dimensions of the feature voxel grids, as well as the feature dimension, can be configured based on the specific task. A lightweight version of the model is also implemented, where Feature grids with reduced (X, Y, Z) dimensions are constructed, and processed as mentioned before. Higher resolution grids are then obtained through a configurable number of upsampling or transpose convolutional operations that lift the resolution to the desired dimension while reducing memory consumption. This step is performed before the prediction head.

TODO create multimodal architecture overview

4.2.3 Loss Definition

In this subsection, we define the loss functions used to train the LiDAR-based and multimodal-based occupancy prediction models. The original LiDAR-based occupancy prediction model is trained using the binary cross entropy (BCE) loss function between the predicted occupancy probability $\hat{o}_{\mathbf{p}}$ and the ground truth occupancy label $o_{\mathbf{p}}$ at each query point as shown in Equation 4.5.

$$\mathcal{L}_{\text{BCE}} = -\left[o_{\mathbf{p}} \cdot \log\left(\hat{o}_{\mathbf{p}}\right) + \left(1 - o_{\mathbf{p}}\right) \cdot \log\left(1 - \hat{o}_{\mathbf{p}}\right)\right] \tag{4.5}$$

While this loss function is suitable for the naive binary prediction task, it does not take the imbalance between the occupied and unoccupied points into account. In a typical urban environment, the number of unoccupied points is much higher than the number of occupied points. This imbalance can lead to a model that is biased towards predicting unoccupied points. To address this issue, we propose using a weighted binary cross entropy loss function, where the weight of the occupied points is set to be higher than the weight of the unoccupied points. The weight is calculated for each sample based on the ratio between the number of occupied points and the number of unoccupied points. The weighted binary cross entropy loss function is shown in Equation 4.6.

$$\mathcal{L}_{\text{BCE}} = -\left[o_{\mathbf{p}} \cdot \log\left(\hat{o}_{\mathbf{p}}\right) + w \cdot \left(1 - o_{\mathbf{p}}\right) \cdot \log\left(1 - \hat{o}_{\mathbf{p}}\right)\right], \quad w = \frac{N_{\text{unoccupied}}}{N_{\text{occupied}}} \tag{4.6}$$

For the label classification task, the standard cross entropy loss (CE) function is used as shown in Equation 4.7.

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i \log\left(\hat{y}_i\right) + \left(1 - y_i\right) \log\left(1 - \hat{y}_i\right)\right] \tag{4.7}$$

Additionally, author of MonoScene [CC] proposed two additional losses that are used to train the model. These losses aim to optimize the (P)recision, (R)ecall and (S)pecificity metrics of the model. These metrics are defined in Equation 4.8.

$$\begin{aligned} P_c(\hat{p}, p) &= \log \frac{\sum_i \hat{p}_{i,c} [\![p_i = c]\!]}{\sum_i \hat{p}_{i,c}}, \\ R_c(\hat{p}, p) &= \log \frac{\sum_i \hat{p}_{i,c} [\![p_i = c]\!]}{\sum_i [\![p_i = c]\!]}, \\ S_c(\hat{p}, p) &= \log \frac{\sum_i \left(1 - \hat{p}_{i,c}\right) \left(1 - [\![p_i = c]\!]\right)}{\sum_i \left(1 - [\![p_i = c]\!]\right)}, \end{aligned} \tag{4.8}$$

In the metrics, P_c and R_c measure the performance of voxels with similar class to c, while S_c measures the performance of dissimilar voxels ($\neq c$). The loss function is defined in Equation 4.9, where p_i the ground truth class of voxel i while is $\hat{p}_{i,c}$ its predicted probability to be of class c

$$\mathcal{L}_{\text{scal}}\left(\hat{p}, p\right) = -\frac{1}{C} \sum_{c=1}^{C} \left(P_c(\hat{p}, p) + R_c(\hat{p}, p) + S_c(\hat{p}, p)\right) \tag{4.9}$$

The semantics loss is defined as $\mathcal{L}_{\text{scal}}^{\text{sem}} = \mathcal{L}_{\text{scal}}\left(\hat{y}, y\right)$ while the geometry loss is defined as $\mathcal{L}_{\text{scal}}^{\text{geo}} = \mathcal{L}_{\text{scal}}\left(\hat{y}^{\text{geo}}, y^{\text{geo}}\right)$. The ground truth geometry and semantic labels are denoted as $\{y, y^{\text{geo}}\}$ while their respective predictions are denoted as $\{\hat{y}, \hat{y}^{\text{geo}}\}$.

4.3 Implementation Details

The proposed approaches are implemented in Python 3.8 with PyTorch 1.13.0 and CUDA 11.6 for hardware acceleration. The implementation is split into different Python packages, where a package is implemented for the data generation and utility functions, another package is implemented for the LiDAR-based occupancy prediction model, and a third package is implemented for the multimodal-based occupancy prediction model in the mmdetection3d framework [MMD20]. All three packages are implemented within a docker container, to ensure reproducibility, and to allow for easy deployment. The CARLA simulator is also deployed within a docker container, and it communicates with the data generation package using CARLA's Python API.

LiDAR-based occupancy prediction models are trained on a single NVIDIA GeForce RTX 3090 GPU with 24GB of memory, while the multimodal-based occupancy prediction models are trained on a single NVIDIA RTX 6000 GPU with 48 GB of memory.

Chapter 5

Evaluation and Results

In this chapter, qualitative and quantitative results of the proposed and tested methods from the previous chapter are presented. In Section 5.1, the datasets and sensor setups used for training and evaluation are presented. The evaluation metrics used for the quantitative evaluation are presented in Section 5.2. The results of the proposed methods are presented in Section 5.3 and Section 5.4 for the LiDAR-based and multimodal methods respectively.

5.1 Datasets

Multiple Datasets were used for training and testing the proposed methods. An objective of the work is to evaluate the performance of the proposed methods on the onboard as well as the infrastructure-based sensor setups.

5.1.1 Infrastructure Datasets

Initially, we rely on the synthetic data generated according to the aforementioned procedure in Section 4.1 for training and testing. For the Infrastructure-based sensor setup, we follow the sensor setup of the A9 dataset [Cre+]'s S110 intersection. We spawn two LiDAR sensors namely, the $s110\_lidar\_ouster\_south$ and $s110\_lidar\_ouster\_north$ sensors. Each of the sensors has 64 channels and a range of up to 150 m. Since we are only interested in the area of the intersection, we filter the point cloud data to only include points within a rectangular area of 55 m × 55 m from the center of the intersection. Additionally, two RGB cameras are spawned analog to their counterparts in the real A9 dataset, namely $camera\_s110\_basler\_s1\_8$ and $camera\_s110\_basler\_s2\_8$. The cameras are configured to produce images with a resolution of 600 × 800 pixels, with a field of view (FOV) of 72°. We spawn an additional camera with a better view of the intersection for debugging and visualization purposes.

In each frame, we spawn up to 30 vehicles of different types, colors, and sizes and 10 pedestrians. The vehicles are spawned in a random position and orientation within the intersection area. One hundred thousand points are uniformly sampled within the intersection area and are used to generate the ground truth occupancy data. Each point is assigned a class label in addition to the occupancy value. Each dataset contains 1200 frames for training and 300 frames for testing, An example of the generated data is shown in Figure 5.1.

The A9 dataset [Cre+] is used to quantitatively evaluate the proposed methods on real data, where we evaluate on a sequence of 100 continuous frames from the S110 intersection. An example of the data is shown in Figure **??**. walter is going to send me an example

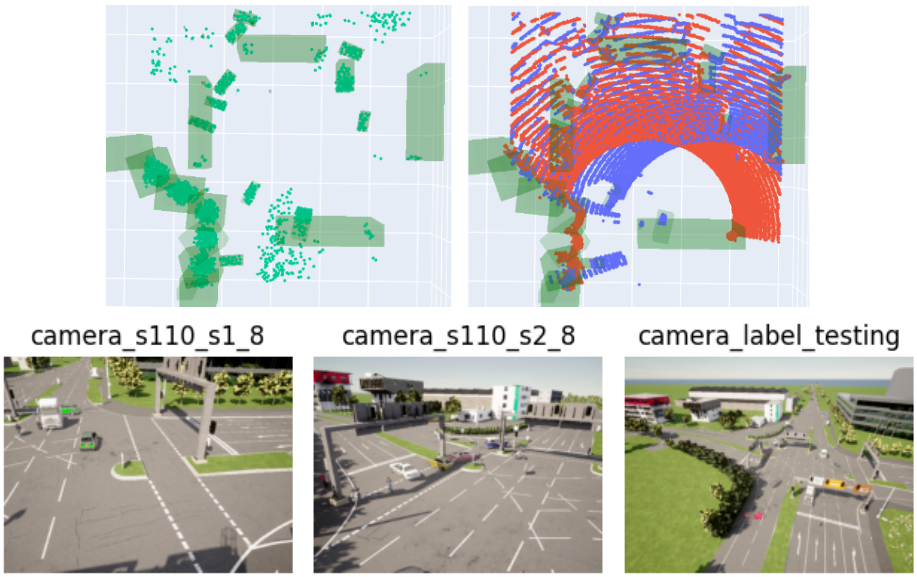

Figure 5.1: Example of the generated data for the infrastructure-based sensor setup. Top: The ground-filtered occupancy data is shown in red in the left. The point cloud data from the north and south LiDAR sensors is shown in blue and red respectively. Objects bounding boxes are shown in green. Bottom: Images from the S1 and S2 cameras.

5.1.2 Onboard Datasets

In the onboard sensor setup, we also rely on the synthetic data generated according to the aforementioned procedure in Section 4.1 for the training and testing of the LiDAR-based methods. For the sensor setup, we follow the sensor setups of the KITTI and Nuscenes datasets [Gei+13; Cae+]. We spawn a top-mounted LiDAR sensor with 64 channels and a range of up to 150 m, and 360° FOV. Additionally, we spawn five RGB cameras with a resolution of 1600 × 900 pixels to create a 360° view of the surrounding environment. Similar to the infrastructure-based sensor setup, we spawn up to 30 vehicles of different types, colors, and sizes and 10 pedestrians. Also, one hundred thousand points are uniformly sampled within a grid of 50 m × 50 m around the ego vehicle and are used to generate the ground truth occupancy data. An example of the generated data is shown in Figure 5.2. While randomly spawning traffic participants in the scene provides a good variety of data, it does not provide a realistic representation of how the traffic participants are distributed in the real world. To address this issue, we use the ground truth data to generate additional datasets with a more realistic distribution of traffic participants. In these datasets, we spawn the traffic participants according to the ground truth bounding boxes of the real A9 dataset [Cre+]. Object's classes, sizes and colors are also carried over from the real dataset and spawned accordingly in CARLA. Sensor data are then captured from the generated scenes.

The only existing dataset available at the beginning of this research was from the scene completion task of the semanticKITTI dataset [Beh+]. However, this dataset is not fully suitable for learning an implicit representation of the surrounding environment. In the dataset, semantic voxel grids are created using segmented point cloud data which is accumulated over time. The format of the data limits the performance of any 3D Occupancy prediction approach since it limits them to the resolution of the voxel grid. Additionally, the dataset is originally tackling LiDAR-based scene completion only. However, camera data from a single forward-facing camera can be also used since the dataset is based on the original KITTI dataset [Gei+13]. Recently though, a new dataset based on the Nuscenes dataset [Cae+] was released [Ope23]. The dataset provides semantic grids for the 3D occupancy prediction

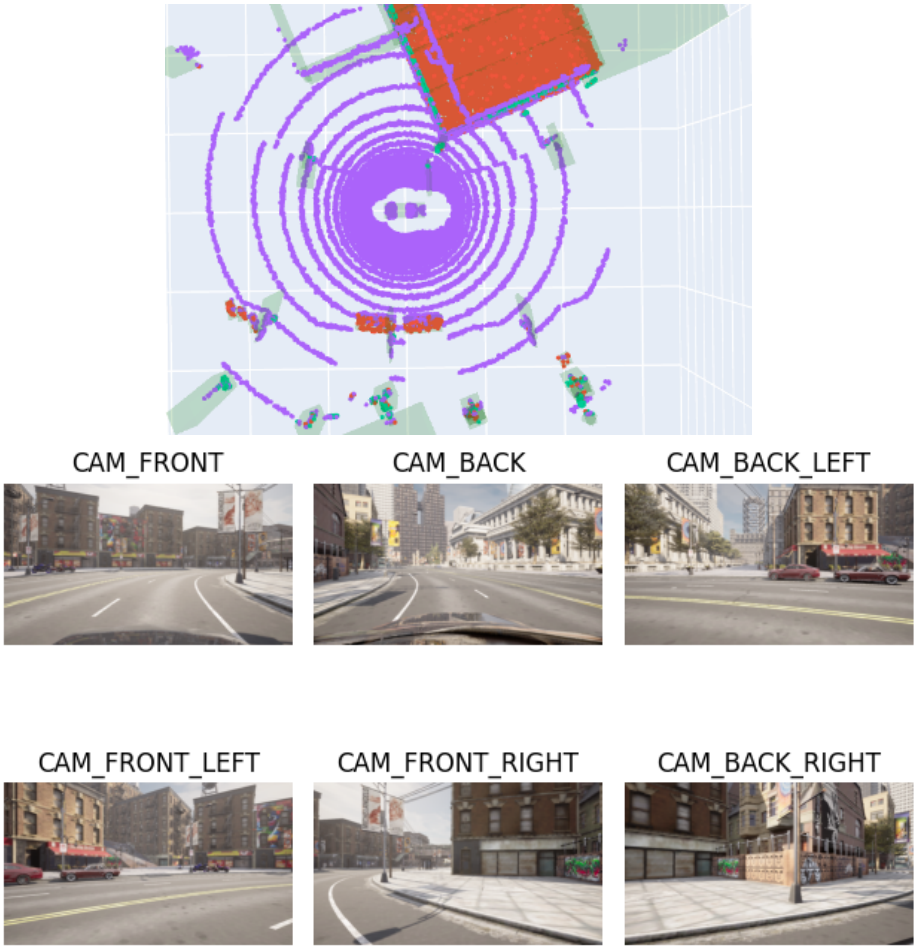

Figure 5.2: Example of the generated data for the onboard sensor setup. Top: The generated point cloud data from the top-mounted LiDAR sensor is shown in purple, ground-filtered occupancy data is shown in red, and extra sampled points are shown in green. Objects bounding boxes are shown in green. Bottom: The multi-view camera setup.

task based on multi-view camera data. LiDAR data can also be used since it is provided in the original Nuscenes dataset. We, therefore, rely on this dataset for the multimodal methods to train and test on real-world data, instead of synthetic data. An example of the data is shown in Figure 5.3.

Figure 5.3: Example of labelled occupancy grids based on the Nuscenes dataset [Cae+; Ope23].

5.2 Evaluation Metrics

To evaluate the performance of the proposed methods, the intersection over union (IoU) metric is used. The IoU metric is defined as the ratio of the intersection between the predicted occupancy grid and the ground truth occupancy grid, and the union between the two. The IoU metric is defined as follows:

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{5.1}$$

Where TP, FP, and FN are the true positives, false positives, and false negatives respectively, a visualization of which is shown in Figure 5.4. The IoU metric is used to evaluate the performance of the proposed methods in the case of binary occupancy prediction.

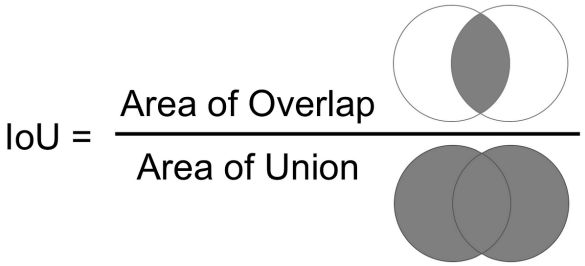

Figure 5.4: Visualization of the meaning of the IoU metric.

For the case of multi-class occupancy prediction, the IoU metric is extended to the multi-class case. The mean IoU (mIoU) metric is used in this case, which is defined as the mean of the IoU metric for each class. The mIoU metric is defined as follows:

$$\text{mIoU} = \frac{1}{C}\sum_{i=1}^{C}\frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i + \text{FN}_i} \tag{5.2}$$

Where C is the number of classes, and TP_i, FP_i, and FN_i are the true positives, false positives, and false negatives for class i respectively.

5.3 LiDAR-based Methods

In this section, we present the experiments and results performed on the LiDAR-based approaches previously explained in Subsection 4.2.1. First we showcase the different models and model configurations tested, then we present the results of these models on the infrastructure-based sensor setup as well as the onboard sensor setup.

To compare these models with a model-based approach, we also implement a model-based approach based on Octomaps [Kai+10]. Since Octomaps splits the space into voxels, we experiment with different voxel sizes and decide at the end for a voxel size of 0.2 m. While our learning-based approaches do not include any temporal information as they output the prediction based on the input from a single frame, it would a fair in comparison if we also use the same approach for the model-based approach. In the case where a single input is used for Octomaps, the results are nothing but a voxelized representation of the input. Therefore, we compare the results of the learning-based approaches with the two versions of Octomaps, one with a single input and one with a temporal window of 5 frames.

To qualitatively evaluate the results, we use three visualization methods. In the first method, meshes are constructed from the predicted occupancy values using the marching cubes algorithm [LC87]. We note that this approach was used in the original occupancy networks approach [**Mescheder.2019**]. In the second method, the predicted occupancy values are directly visualized as a point cloud, while in the last method the predicted occupancy values are visualized as a voxel grid with a voxel size of 0.2 m.

5.3.1 Model Configurations

Models based on [Son+20] have two main configurations when trained and tested which the authors originally proposed. The first of which is what we call 'one-shot' configuration. In this setting, the model is trained and tested on the complete scene at once. The second configuration is the 'crop'. In this setting, the scene is divided into smaller crops, where the model is trained and tested on each crop separately. The results of the crops are then merged together in a sliding window fashion. A comparison between the two configurations is shown in Figure 5.5. For training the cropped model, a point is sampled on the xy-plane and a crop is generated around this point. The crop size is a hyperparameter that can be tuned.

The feature representations, which the models are based on, can vary as well. As mentioned in Subsection 4.2.1, the original approach can use a feature representation based on planes, grids, or a combination of both. The original implementation did not support the use of a feature representation based on grids and planes combined for the crop configuration. We therefore implemented this feature representation for the crop configuration as well. Other parameters such as the resolution of the feature representations and the feature dimensionality are also tunable parameters, that were tested in the experiments.

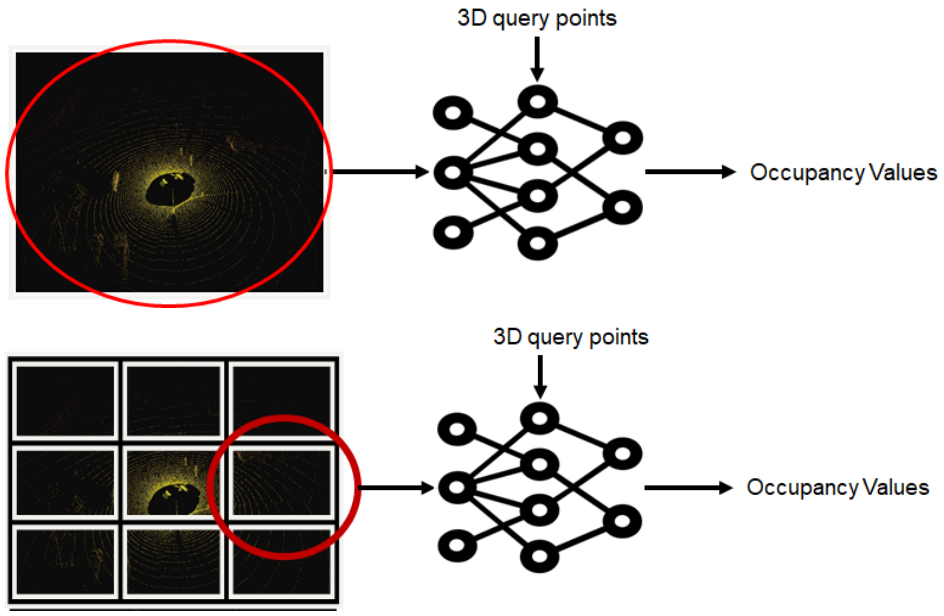

Figure 5.5: Comparison between the 'one-shot' and 'crop' configurations. In the one-shot configuration, the observation point cloud is fed as a whole to the network (top). In the crop configuration, the observation point cloud is divided into smaller crops, which are fed to the network separately (bottom).

5.3.2 Onboard Sensor Setup

Models with different configurations and feature representations have been trained on the synthetic dataset mentioned earlier. We train three different models with different feature representations, but with the same resolutions and feature dimensionality. The first model is a model based on the 'crop' configuration with combined planar and grid feature representations, where the planes had a resolution of 128×128 while the grid had a resolution of $32 \times 32 \times 32$. Both had a feature dimension of 32. The crop dimensions were set to 5m^3 after some experimentation.

The second model is a model based on the 'one-shot' configuration with a combined planar and grid feature representation as well. The resolutions of the planes and grid were kept the same as the previous model in order to have a fair comparison.

Parallelly, a third model based on the crop setting was trained with a grid feature representation only. The grid resolution was chosen to be $32 \times 32 \times 32$ with a feature dimension of 32. The crop dimensions were set to 5m^3 as well.

All three models were trained were left to train without a maximum number of epochs. Instead, an early stopping criterion was used based on the validation loss. The training was stopped when the validation loss did not improve for 3 consecutive epochs.

The training losses of the three models as well as the evaluation score on the validation set are shown in Figure 5.6.

The evaluation scores of the three models show that the one-shot model achieves the highest score on the validation set, when compared to the other two models. While this might look deceiving at first, especially when looking at the training losses, it is important to note that the loss in the one-shot model is calculated on the whole scene at once, while the loss in the crop models is calculated at one crop. When divided by the number of crops, the loss of the one-shot model is then significantly lower than those of the crop models.

It is also noticeable that the crop models have a very fluctuating loss values, which is due to the fact that the crops are randomly sampled from the scene. This means that the crops can contain very different information, which can lead to very different loss values. Similar

Figure 5.6: Training losses (top) and IoU validation scores (bottom) of the three models trained on the onboard sensor setup. In the loss curves, the number of iterations is shown on the x-axis, while the loss is shown on the y-axis. In the IoU curves, the number of iterations is shown on the x-axis, while the IoU score is shown on the y-axis. The first model (crimson) is a model based on the 'crop' configuration with combined planar and grid feature representations. The second model (turquoise) is a model based on the 'one-shot' configuration with a combined planar and grid feature representation. The third model (blue) is a model based on the crop setting with a grid feature representation only.

interpretations can be made for the evaluation scores as well.

When comparing the two crop models, it is also noticeable that the model with the combined feature representation achieves a higher evaluation score and lower loss on average when compared to the model with the grid feature representation only. This is to be expected as the combined feature representation contains more information than the grid which has a significantly lower resolution.

To further analyze the performance of the three models, we evaluate the models on the test set. Additionally, we evaluate the baseline on the same test set. The results are shown in Table 5.1.

Model	IoU
Baseline	X.XXX
Crop (grid)	0.702
Crop (combined)	0.7268
one-shot (combined)	0.249

Table 5.1: IoU scores of the three models trained on the onboard sensor setup, as well as the Octomap baseline.

The results show that the baseline model achieves a very low IoU score on the test set. This is to be expected as the baseline model is a very simple model, that does not take any information about the scene into account. A main reason for the low IoU score is that the baseline model has a very high false negatives rate, since the occupancy values are solely based on previous observations. This means that it is directly affected by the sparsity of the point cloud.

The one-shot model also achieves a very low IoU score on the test set, although it achieved the highest score on the validation set. This is due to the fact that the test set was generated based on a completely different scenes than the training and validation sets. The training and validation sets were split randomly from the same scenes, while the test set was generated from a completely different set of scenes.

To address this issue, an improved data generation pipeline was implemented. The new pipeline ensures a greater variety in the data by generating frames from different scenes and Towns in the same dataset.

The crop models achieve a significantly higher IoU score on the test set when compared to the one-shot model and the baseline. This is to be expected as the crop models have a 5× higher resolution than the one-shot model.

To have a better perspective on the performance of the different models, we also present the qualitative results of the three models as well as the baseline. The input point clouds, as well as images from the multi-view cameras are shown in Figure 5.7. In the shown sample frame, The ego vehicle is surrounded by vehicles from different types, some oh which are partially or fully occluded.

The qualitative results of the baseline Octomap are shown in Figure 5.8. In the figure, we show the performance of the baseline with single and multiple inputs. In addition to the previously mentioned problem of sparsity the baseline model also has a noticeable problem with false positives in areas where the occupancy changes rapidly.

The results of the crop and one-shot models with combined feature representation are shown in Figure 5.9. The performance of the one-shot model does not build an accurate representation of the scene. The model tend to only predict the occupancy of the ground plane correctly, while the rest of the scene is mostly predicted as free, apart from some noticeable false positive artifacts. The crop models achieve a significantly better performance, as they learn to predict complete shapes of the objects in the scene, although only partially

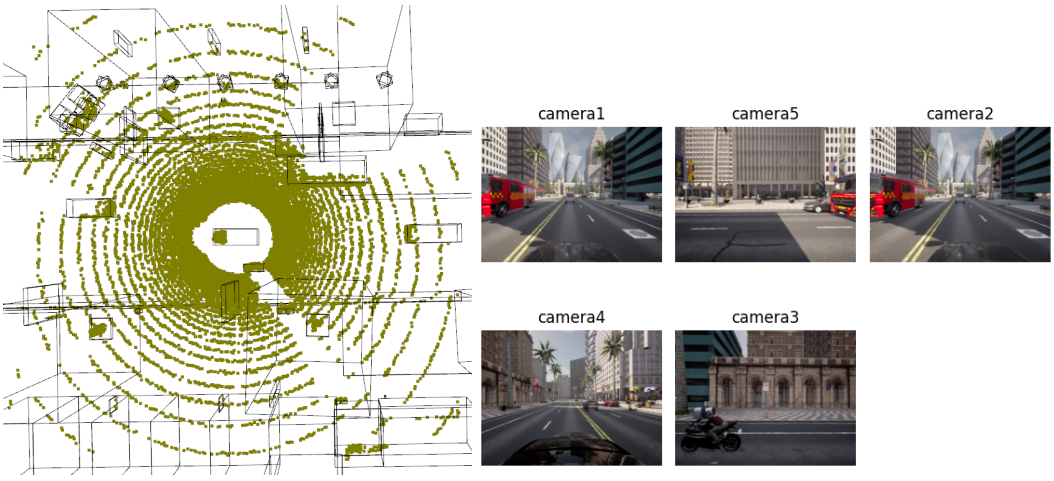

Figure 5.7: Input point clouds as well as images from the multi-view cameras in a sample of the test set.

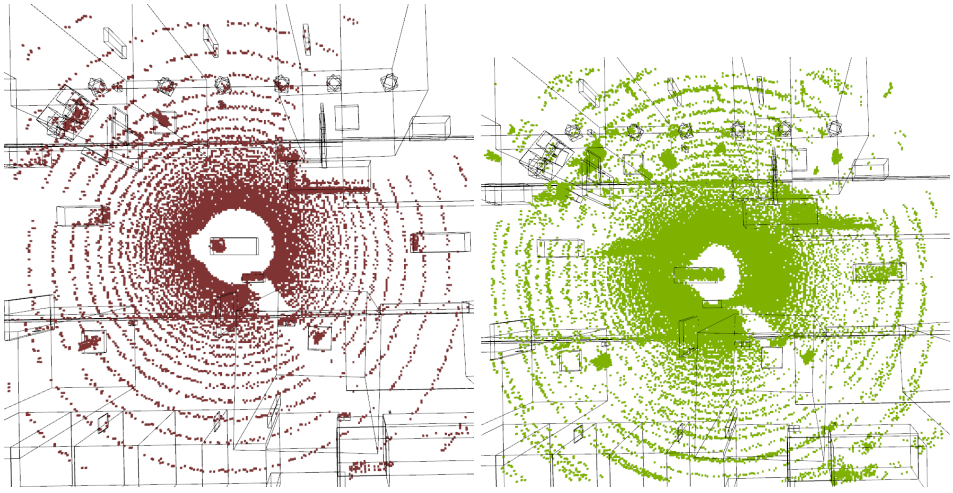

Figure 5.8: Qualitative results of the baseline Octomap with single input point cloud (left), and multiple inputs (right).

observed in the input data. This can be noticed in particular in the top left side of the rendered mesh, where the model predicts the full shape of two vehicles, although only observed from the back side.

5.3.3 Infrastructure-based Sensor Setup

Analogous to the onboard sensor setup, we conduct several experiments on the infrastructure-based sensor setup. We train the models on synthetic datasets described in Section 4.1 and Subsection 5.1.1. Specifically, we train the models on the dataset with random objects, while using the other dataset based on the A9 for testing. This was decided since the dataset with random objects is more diverse, and contains more variations in the objects' shapes, types, and sizes, while the other dataset is composed of sequences that are similar to each other.

We initially train models with the same configurations as the ones used for the onboard sensor setup. This way we can additionally validate whether the models are able to learn different representations based on different sensor setups. Two models based on the crop configuration, and a one-shot model was trained. The first crop model uses a combined feature representation with a 3D grid and three 2D planes, while the second crop model uses

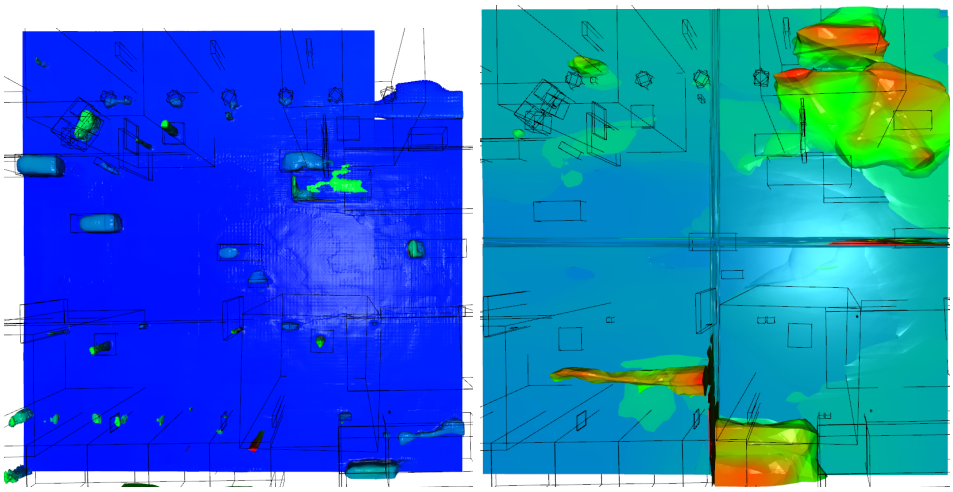

Figure 5.9: Qualitative results of the crop (left) and one-shot (right) models with combined feature representation. To better visualize the results, we construct a mesh from the occupied points and render it using Open3d. The meshes are colored based on the height of the points.

a 3D grid only. The one-shot model uses a combined feature representation as well. The feature dimensions is kept the same in the three models at 32 channels. The resolution of the 3D grid is set to $32 \times 32 \times 32$, while the resolution of the 2D planes is set to 128×128.

To shorten the training period, we use the already pre-trained weights of the models trained on the onboard sensor setup as a starting point, this has brought some improvements on the performance of the models, when compared to their counterparts that were trained from scratch. Figure 5.10 shows IoU evaluation score on the validation set.

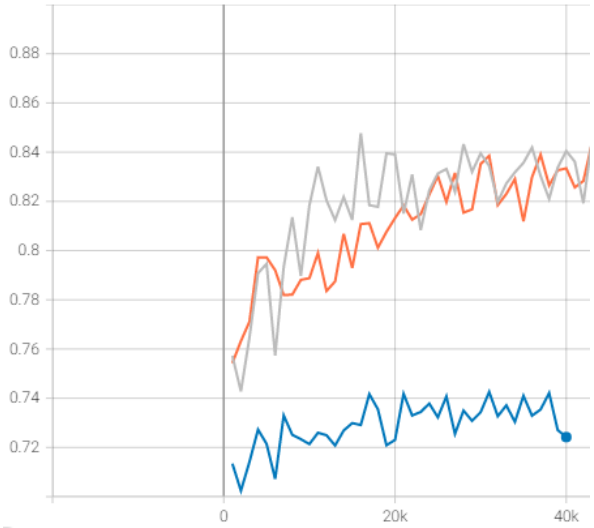

Figure 5.10: The evaluation score on the validation set, the number of iterations is shown on the x-axis, while the IoU score is shown on the y-axis. The first model (grey) is a model based on the 'crop' configuration with combined planar and grid feature representations. The second model based on the crop setting with a grid feature representation only (blue). The model based on the 'one-shot' configuration with a combined planar and grid feature representation (orange).

The validation scores of the models show a comparable performance between the crop and one-shot models that use a combined feature representation. Surprisingly, the crop model that uses a grid feature representation only, achieves a significantly lower performance. This is in contrast to the results of the onboard sensor setup, where the crop model with a

grid feature representation only, achieved a comparable performance to the crop model with a combined feature representation.

Model	IoU
Baseline	X.XXX
Crop (grid)	0.55
Crop (combined)	**0.74**
one-shot (combined)	0.72

Table 5.2: IoU scores of the three models trained in the infrastructure setup, as well as the Octomap baseline on the test set.

We further analyze the performance of the models on the test set. The results are shown in Table 5.2. The crop model with a combined feature representation achieves the highest performance, with an IoU score of 0.74. The one-shot model with a combined feature representation achieves a comparable performance, with an IoU score of 0.72. This is in contrast to the results of the onboard sensor setup, where the one-shot model achieved a significantly lower performance. The crop model with a grid feature representation only, achieves a significantly lower performance, with an IoU score of 0.55.

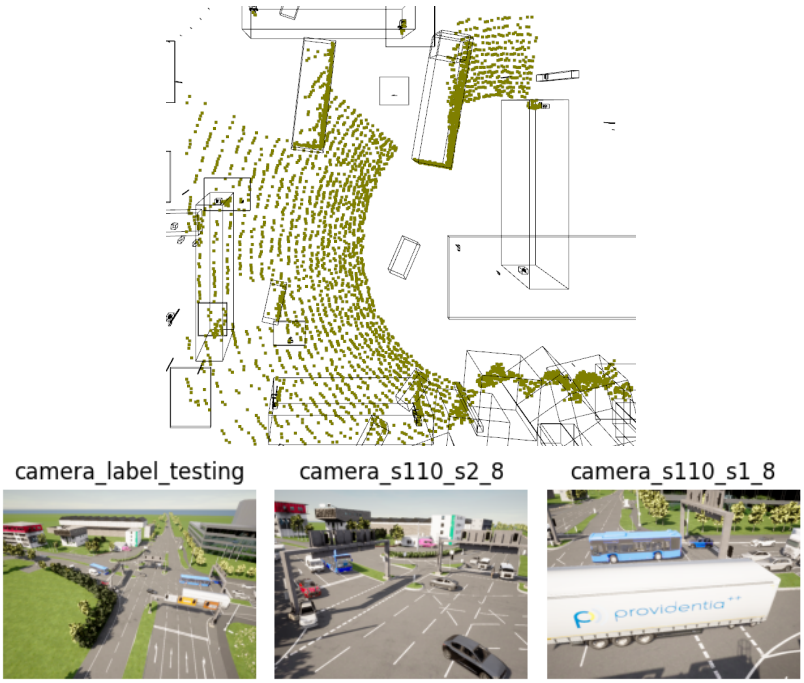

Figure 5.11: Input point clouds as well as images from the cameras in a sample of the test set.

Additionally, we qualitatively analyze the results of the crop and one-shot models with a combined feature representation. A selected frame on the test set is shown in 5.11, where different object types are present. The results of the baseline Octomap method are shown in Figure 5.13. The rendered meshes based on the occupancy values of the two models are shown in Figure 5.12.

The results show that the crop model is able to predict occupancy values accurately, while inferring the correct objects shapes based on a partial observation. However, some false positives are present, especially in the areas where there is not any observation from the input point cloud. While the one-shot achieved a comparable performance during quantitative

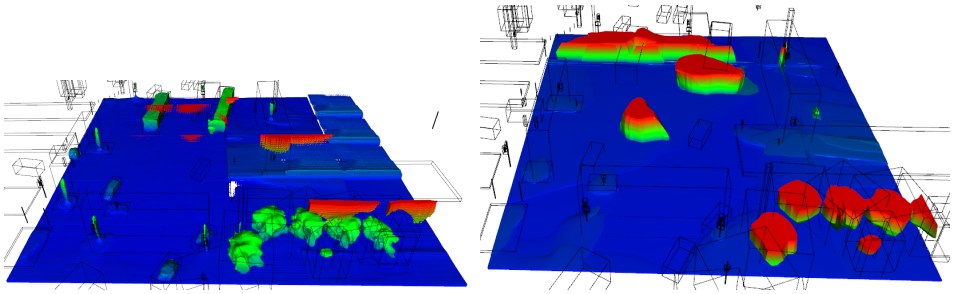

Figure 5.12: Qualitative results of the crop (left) and one-shot (right) models with combined feature representation. To better visualize the results, we construct a mesh from the occupied points and render it using Open3d. The meshes are colored based on the height of the points.

evaluation, the results show that the one-shot model mostly predicts the occupancy of the ground plane, and some other static objects. Areas with dynamic objects are mostly predicted as free space. The results of the baseline Octomap method are shown in Figure 5.13. The baseline suffers from the same issues as in the onboard sensor setup, where the results are sparse and noisy.

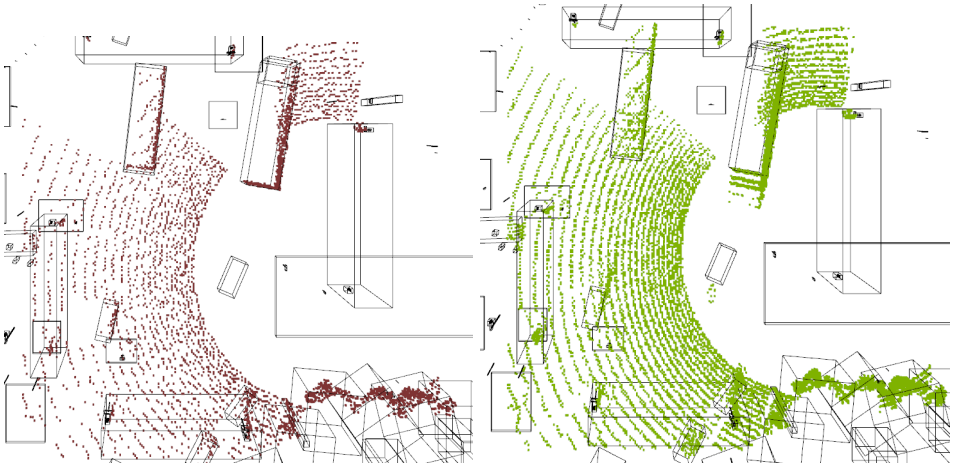

Figure 5.13: Qualitative results of the baseline Octomap with single input point cloud (left), and multiple inputs (right).

In the previously mentioned results, a single input point cloud was used for training and testing. We further analyze the performance of the methods when multiple input point clouds are used. Following the approach presented in Subsection 4.2.1, we extend the models to use multiple input point clouds, by concatenating them together after being transformed to common coordinate frame. Although this maybe a naive approach, it allows us to study the impact of density of the input point clouds on the performance of the models.

While the results of the crop model with a combined feature representation are promising, the model has a high latency. A more in depth analysis of the inference time is presented in Section 5.3.4. To address this issue, we propose a modified version of the crop model where only 2D planar representations are used. This allows for a lighter model, and at the same time allows the exploit the performance of the crop configuration.

A comparison of the results of the crop model with a combined feature representation and the crop model with a planar feature representation is shown in table 5.3. For these experiments, a new dataset was generated, where we spawned two LiDAR sensors (south and north) analogue to their real world counterparts in the Providentia++ test-bed. The model configurations are same to the ones used in the previous experiments. We additionally

train the two models on a single input point cloud from the south sensor to provide a better comparison.

Model	Number of point clouds	IoU
Crop (grid)	1	XXX
Crop (planar)	1	XXX
Crop (grid)	2	XXX
Crop (planar)	2	XXX

Table 5.3: IoU scores of the three models trained in the infrastructure setup, as well as the Octomap baseline on the test set.

Finally, we showcase the results of the crop model with combined feature representation on the test set of the real A9 dataset. While exact quantitative evaluation is not possible due to the lack of ground truth, we can only qualitatively analyze the results. The results shown in Figure 5.14 show that the model is able to predict the occupancy of two vehicles with different shapes, as well as the ground plane, while being trained solely on the synthetic data. The two vehicles are circled in red and black for more clarity. Objects not observed by the LiDAR sensor are predicted as free space as the model is not able to infer the occupancy of the area.

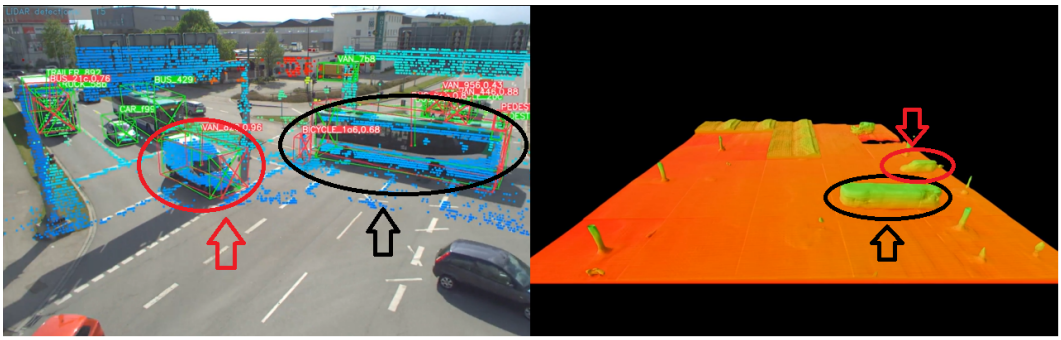

Figure 5.14: Qualitative results of the crop model with combined feature representation on the A9 dataset. On the left, the image from S1 camera is shown with the input point cloud from the LiDAR sensor projected on it. On the right, the rendered mesh based on the occupancy values of the model is shown. (Note that the view is from an opposite perspective to the one in the left image.)

5.3.4 Inference time

Another important aspect of the proposed methods is the inference time. The authors of the original paper have reported very significant accuracy, but have not reported the inference time. We therefore measure the inference time of the proposed methods.

The structure of the network has a significant impact on the inference time. The 3D convolutional operations in models that use 3D grid representations are the most computationally expensive operations. Additionally, the number of parameters and the memory footprint of the network increases significantly with higher resolution 3D grids.

To analyze the impact of the different modules of the network on the inference time, we use `Pytorch Profiler` to create flame graphs. The flame graphs show the time spent in each module of the network during inference. The flame graphs are shown in Figure 5.15.

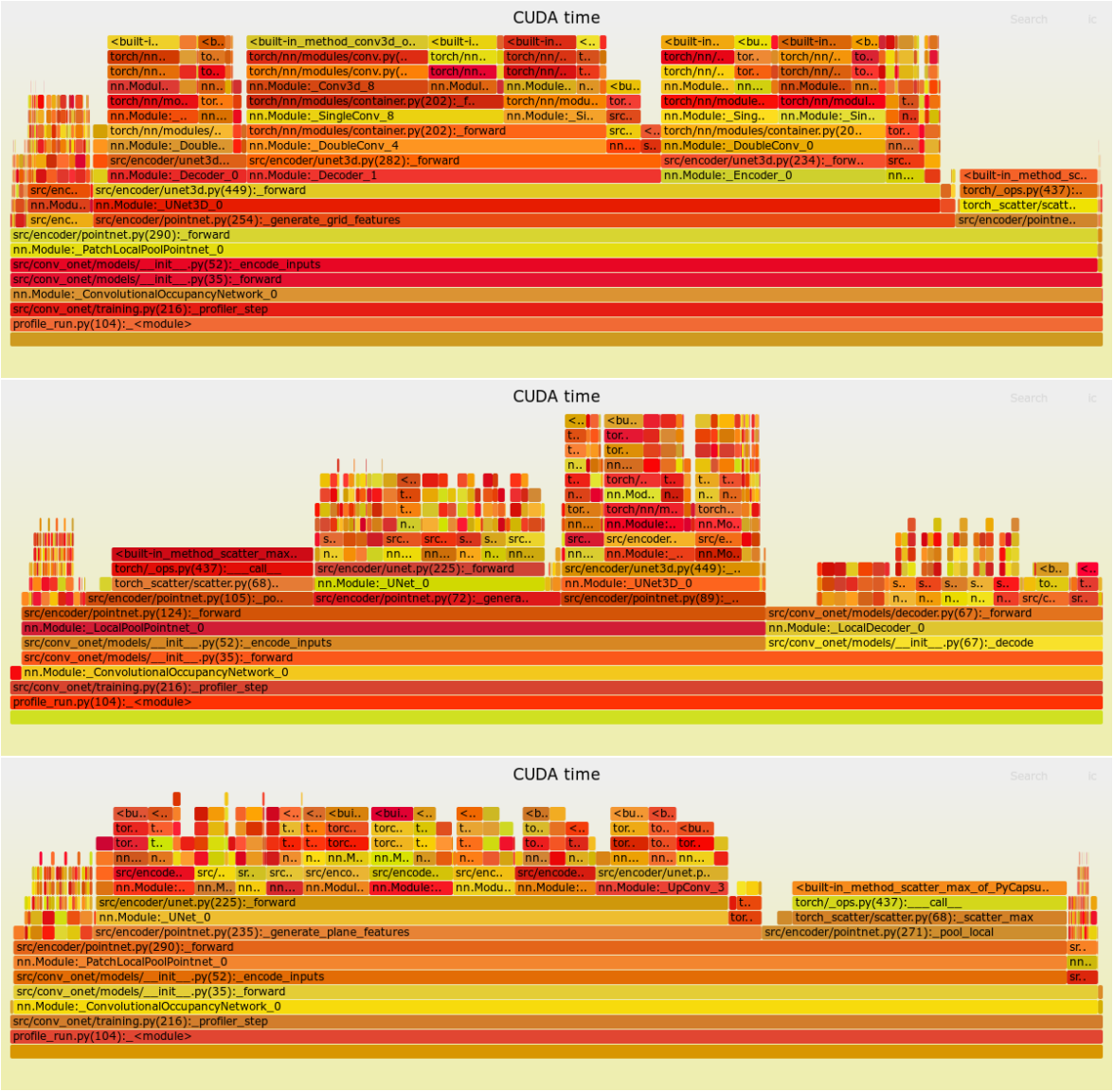

Figure 5.15: Flame graphs showing the time spent in each module of the network during inference. The top graph shows the flame graph of the crop model with combined feature representation, the middle graph shows the flame graph of the one-shot model with combined feature representations as well. Finally, the bottom graph shows the flame graph of the crop model with only planar feature representation.

The trace of the crop model with the combined feature representation shows that the most time during inference is spent in the 3D UNet module. However, the trace of the one-shot model with the combined feature representation does not show a similar behavior. While this is an unexpected result, we leave this for future work. Interestingly, the trace of the crop model with the planar features only shows that the most time is spent in the 2D UNet module instead, which is the module with most parameters and computation footprint.

Model	Feature representation shape	Inference speed	Number of parameters in millions
crop	3×128^2	**0.54** s	$7.8M$
crop	$3 \times 128^2 + 32^3$	3.68 s	$8.8M$
crop	32^3	2.55 s	$1M$
one-shot	$3 \times 128^2 32^3$	0.06 s	$8.8M$

Table 5.4: Inference speed for crop models and their respective feature representation shape.

The one-shot configuration has the fastest inference between all the methods, since the network is only evaluated once for the whole scene. The crops models are not only slower due to the inferring the scene in multiple crops, but also due to the additional overhead of the pre and post-processing steps needed to crop the input point cloud and to stitch the cropped predictions back together. Comparing the inference speed of the crop models with different feature representation shapes, we can see that the inference speed is significantly faster for the model with only planar representations. This is expected, since the number of parameters and the computation footprint of the model is significantly smaller.

5.3.5 Further Analysis

The one-shot models were further analyzed to determine the reason behind their poor performance. Initially, we thought that the poor performance was due to the low resolution of the feature representation. In order to test this hypothesis, we trained a one-shot model with upsampled feature representation. The model is similar to the one-shot model with combined feature representation we mentioned earlier. However, the feature representations are upsampled before being fed into the occupancy prediction head. Two upsampling steps using convolutional layers bring the resolution of the 2D planar feature representations to 512×512, while a single upsampling step was used to bring the resolution of the 3D feature representation to 64^3.

Model	IoU
one-shot (combined)	0.249
one-shot (planes) $+upsampling$	0.69

Table 5.5: Comparison between one-shot models. The model with the combined feature representation has a significantly lower performance than the model with the plane feature representation and upsampling.

Table 5.5 shows the results of the one-shot model with the upsampled feature representation. The model has a significantly higher performance than the one-shot model without upsampling, but further analysis showed that the boost in the performance is due to the reduced number of false positive artifacts.

Another experiment was conducted to determine the impact of the crop size on the performance of the crop models. Initially, we use a crop size of 10m^2 on the xy-plane for the crop models. This translates to 25 crops for the whole scene. The results are shown in Figure 5.16.

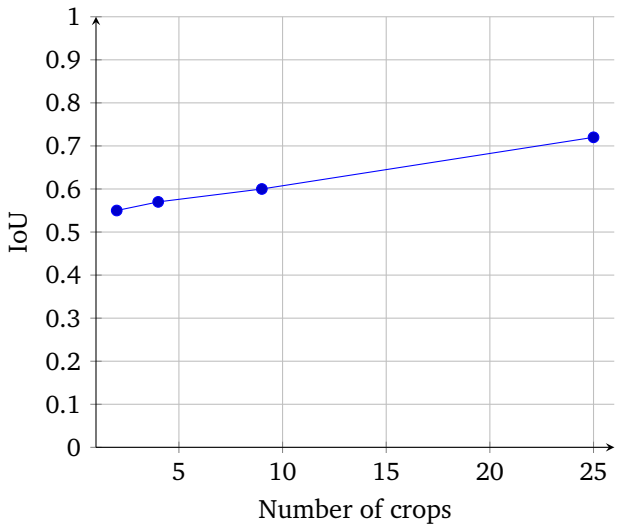

Figure 5.16: Comparison between crop models with different crop sizes. The performance of the model increases with more crops

5.4 Multimodal Methods

In this section, we present the experiments conducted to evaluate the proposed multimodal models based on the approach described in Subsection 4.2.2. In the first subsection, we present the different models and model configurations developed and used in the experiments. We then present the results of these models and configurations in the on-board sensor setup.

The absence of model-based approaches that use camera and LiDAR data for occupancy prediction makes it difficult to compare the performance of the proposed learning-based methods. Additionally, we were not able to find any publicly available methods that use camera and LiDAR data for occupancy prediction. Since we decided to train and test the models on the real-life 3D Occupancy Dataset [Ope23], contrary to the previous experiments, we decided to use the baseline models offered by the dataset authors as a reference.

The authors of the 3D Occupancy Dataset [Ope23] provide a baseline model based on the BEVFormer [Li+e] model proposed for multi-view camera 3D object detection. Additionally, Authors of the BEVDET [Hua+a] camera-based 3D object detection model provide another baseline model for the 3D Occupancy Dataset. The BEVFormer is able to achieve a mIoU of 23.67% on the multi-class occupancy prediction task, while the BEVDET model achieves a mIoU of 23.7%. An improved version of the BEVDet which incorporate tempral information is able to achieve a mIoU of 42.0%.

The ground truth occupancy labels of the 3D Occupancy Dataset are provided in the form of a 3D semantic grid with a cell size of 0.4m, resulting in a resolution of $200 \times 200 \times 16$. It is therefore necessary to for our proposed models to have the same output resolution. However, this resolution increases the memory footprint of the models significantly. In order to reduce the memory propose a model that uses a feature representation with a lower resolution which can be then upsampled to the desired output resolution.

While the authors of UVTR showed that using ResNet-101 as an image backbone yields better performance than ResNet-50, we decided to use ResNet-50 as the image backbone for the proposed models. This is to reduce the memory footprint of the models, and ensure that the models can be trained on fit on the GPU.

The unified feature representation of the UVTR model provides extra flexibility in choos-

ing the modalities used for the occupancy prediction. Since both camera and LiDAR are used to generate their respective feature representations before the fusion module, it is possible to use only one of the modalities for the occupancy prediction.

Initially, the intuition was that the publicly available trained weights of the UVTR model would be a good starting point for the training of the proposed models. To validate this, tested the models on the Nuscenes validation set for the 3D object detection task. The results of the model were unexpectedly poor, with a mean average percision (mAP) of 0.13. This is significantly lower than the advertised mAP of 0.65. We therefore decided to train the UVTR model on the v1.0-mini train set of Nuscenes to validate the performance of the model. Training the model for 20 epochs on the v1.0-mini set resulted in a mAP of 0.3 on the v1.0-mini val set. Although the score was still significantly lower than expected, it was considered sufficient for the purpose of the experiments taking the size difference into account.

Training the models on the complete dataset was challenging, especially in the experimentation phase. Training a model on the v1.0-mini set of Nuscenes takes approximately 8 hours on a single RTX 3090 GPU for 20 epochs. This makes it difficult since the full dataset is around 34× bigger than the v1.0-mini set.

Apart from the classification head and the loss, the models for binary and multi-class occupancy prediction are identical. The binary model is trained used the previosly described weighted binary cross entropy loss. The multi-class model is trained using the cross entropy loss, in addition to the geo-scale and semantic scales losses described in Subsection 4.2.3. After several experiments, the following model configurations were decided:

- **Voxel size:** LiDAR point clouds are voxelized into a voxel grid with a size of 0.2m
- **Image backbone:** ResNet-50
- **Feature channel size:** 256
- **Feature grid resolution:** $100 \times 100 \times 8$
- **Upscaling factor:** 2

While testing a new model, it is often advised to first overfit the model on a single frame, to ensure that the model is able to learn. After experimenting with different hyperparameters such as loss functions, optimizers, and learning rates. The overfitting experiment for the binary and multi-class models are shown in Figure 5.17.

Before training on the full dataset, the models were trained on the v1.0-mini set of Nuscenes for 20 epochs. The training losses and validation scores for the binary and multi-class models are shown in Figure **??**.

While the performance of the models is not as good when compared baseline models, we can validate that the models are able to learn. Training on the full dataset was started afterwards. Since training a single model on the full dataset takes around 8 days, we only showcase the results of the binary model trained on 10 epochs. The training losses and validation scores for the binary model are shown in Figure **??**. The quantitative results shown in Figure **??** shows a good performance in areas near the ego vehicle, but the performance drops significantly due to false postives, as the distance from the ego vehicle increases.

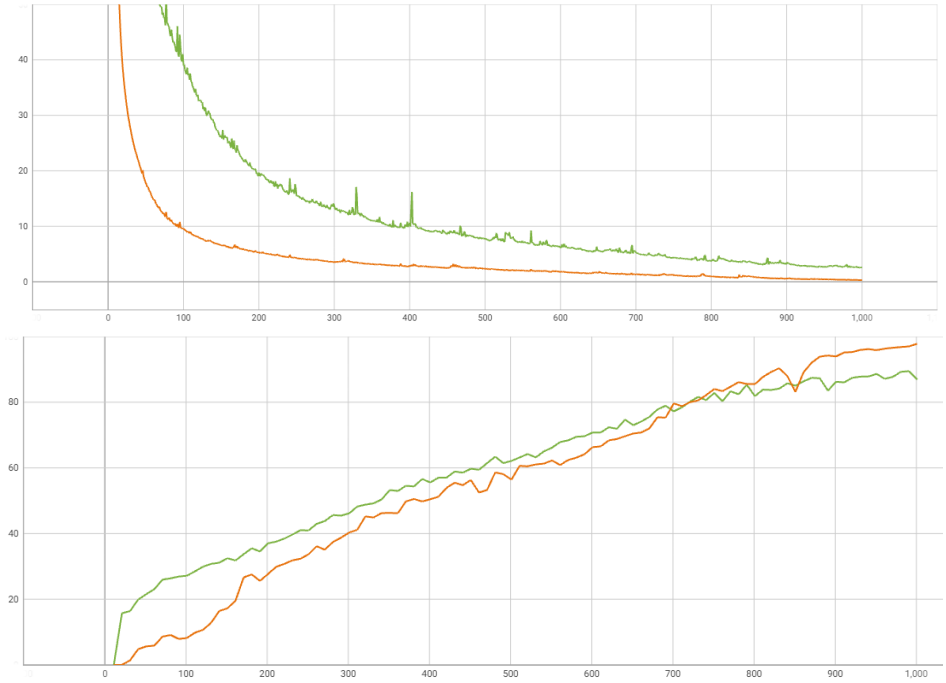

Figure 5.17: Training losses (top) and IoU validation scores (bottom) in the overfitting experiment for the binary (green) and multi-class (orange) models.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

3D occupancy prediction in autonomous driving is an emerging topic that has gained significant attention in the last year. Motivated by this, we present two approaches to predict 3D occupancy in the context of autonomous driving, and showcase their performance, potential and limitations in different scenarios.

Motivated by the fact that data is a decisive factor in the performance of learning-based methods, and the lack of publicly available datasets for 3D occupancy prediction in autonomous driving, at the time of starting this thesis, we propose a new data generation pipeline based on *CARLA* simulator to generate synthetic data for 3D occupancy prediction. We show that the proposed pipeline is capable of generating large-scale, high-quality, and diverse datasets for the task of 3D occupancy prediction in autonomous driving in the infrastructure sensor setup, as well as in the on-board sensor setup. The pipeline is also capable of generating multimodal data, using RGB cameras and LiDAR sensors, while allowing the extension to other sensors.

In the first approach, we extend an existing 3D occupancy prediction developed originally for object and indoor scene reconstruction [Son+20] to autonomous driving domain. In this, we hypothesize that implicit representation of the environment is more suitable for autonomous driving than explicit representation. The presented approach can be used for both, the infrastructure sensor setup, as well as the on-board sensor setup. The methods exhibit competitive performance in both against the baseline models in the respective setups. We also demonstrate that the methods trained on synthetic data generalize well to real-world data by qualitatively evaluating the methods trained in CARLA on the real A9 dataset [Cre+]. Additionally, we study the effect feature representations on the performance of the method, with respect to the choice of feature representation dimension and its resolution.

In the second approach, we propose a novel method for 3D occupancy prediction in the on-board sensor setup, which can be easily extended to the infrastructure sensor setup. We repurpose a model originally developed for 3D object detection [Li+a] and modify it to suite the 3D occupancy prediction task. Although the model did not outperform other state-of-the-art methods, we argue that the model has indeed achieved promising results, that can be further improved in the future.

6.2 Future Work

The work presented in this thesis is a step closer towards understanding the potential of implicit representations for 3D occupancy prediction in autonomous driving. However, based on our results, there are still many open questions and limitations that need to be addressed in the future.

The data generation pipeline presented can be further improved to generate more diverse scenarios, which can help in improving the generalization of the models. Additionally, the pipeline can be extended to generate data from other sensors, such as radar, which can be used along with data from LiDAR and RGB cameras to achieve better performance.

To provide a more insightful analysis of the methods, other metrics such as the *F1 score* can be used in addition to the *IoU* metric, which showed to be biased towards the majority class.

In the LiDAR-based approach, the method can be improved by using a more powerful feature extractor, which is more globally aware. The performance of the one-shot model can be improved by constructing the feature representations in a non-uniform resolution grid, which can help in capturing more fine-grained on xy plane, and prevent the loss of information. This, and the use of sparse convolutions can help in improving the performance of the method, by increasing the resolution of the feature representations without increasing the computational and memory requirements.

Additionally, to exploit the full potential of the crop models and allow it to work in real-time, the inference speed can be improved by using batching at inference time.

While the multimodal approach showed promising results, the performance of the method can be improved by using pre-trained backbones and feature extractors to shorten the required training time, in combination with hyperparameter tuning. Additionally, different fusion strategies such as learnable attention mechanisms can be used to increase the performance.

Analogous to recently proposed methods for 3D object detection, both models could benefit from making use of temporal information, which can help in cases of occlusion.

List of Figures

1.1 Motional (formerly NuTonomy) research platform vehicle used in Nuscenes dataset [Cae+]. 2
1.2 Overview of the Providentia++ test bed showcasing the different sensor stations on the highway as well as in urban field in Garching, Germany. 2
1.3 An example where camera-based object detector on a Tesla vehicle failed to detect an object. [Tes22a; Tes22b] 3

2.1 The seeing passenger car (VaMoRs-P), one of the first self-driving vehicles equipped with camera sensors as part of the EUREKA project [E D+94] 7
2.2 The pinhole camera model: Rays from an object in the world pass through the pinhole in the front of the camera and form an image on the back plane (the image plane) [Pri12]. 9
2.3 Pinhole camera model terminology. 9
2.4 Operation principal of LiDAR technology [RCG22]. 10
2.5 Typical spinning 3D LiDAR. 11
2.6 The four common types of 3D representation. (a) Voxels. (b) Point cloud. (c) Explicit representation (mesh). (d) Implicit representation [M M+19]. 12
2.7 2D Object detection and semantic segmentation in autonomous driving as seen in Nuscenes dataset [Cae+]. 14
2.8 The initial sensor setup of the Providentia test bed on the A9 highway [Krä+]. 15
2.9 Top: Overview of the test bed Providentia++. The original stretch highlighted in yellow, and the extension is highlighted in blue [Cre+]. Bottom: Gantry bridge within the s110 intersection in the Providentia++ project. 15
2.10 An example scene from CARLA. 16

3.1 PointNet architecture. Input and feature transformations are applied to n input points, the resulting features are then aggregated using max-pooling [Qi+a]. 18
3.2 PointPillars architecture overview [Lan+] (top). PV-RCNN architecture overview [Shi+] (bottom). 18
3.3 Overview of the SE-SSD model depicting the student-teacher architecture and the formulated losses [Zhe+]. 19
3.4 VISTA module [Den+] (top). MDRNet architecture overview [Hua+22] (bottom). 20
3.5 Depth prediction step in Lift, Splat, Shoot [PF] (top), for each pixel p, a context feature c and a depth distribution α are predicted, features at each point along the ray are determined by the outer product of α and c. Model overview of DETR3D [Wan+] (bottom). 21
3.6 An overview of BeVerse [Zha+b] fusing information from previous frames (top). The framework of the depth-guided BEVDepth [Li+d] (bottom). 22
3.7 Overall structure of 3D-CVF [Yoo+20]. 24
3.8 Stixel representation of 3D environment [HUD09]. 27

3.9 An example of NDT representation compared to a normal occupancy grid [Saa+13a]. (a) The measurement points are placed into grid cells and (b) The distribution parameters are computed for each cell for which there are measurement points. (c) An example of an occupancy grid using the same measurement points. 27

3.10 An overview of the S3CNet architecture [Che+c]. 29

4.1 An example of the output of **cast_ray** method in CARLA. In the top image, a spherical point cloud is sampled around the ego vehicle. The points are used as an initial location for the **cast_ray** method. The end location is set to be the center of the vehicle. The bottom image shows the returned points with the Car label in CARLA. 34

4.2 An example of the using **cast_ray** method in CARLA to determine a point's occupancy in a simplified 2D scenario. The left image shows that the point is labelled as occupied since the ray intersects with the object's boundary from both sides while being within the object's bounding box. The right image shows that the point is labelled as unoccupied, since the ray only intersects with the object's boundary from one side, although being inside the object's bounding box. 35

4.3 Overview of the LiDAR-based occupancy prediction approach [Son+20]. Left: The input point cloud is encoded using a PointNet-based encoder with local pooling and the feature grids are constructed. Right: The feature grids are refined using UNet hourglass networks. The interpolated features for a given query point are then decoded into occupancy probabilities using fully connected layers. 36

4.4 Overview of the UVTR model for 3D object detection [Li+a]. 38

5.1 Example of the generated data for the infrastructure-based sensor setup. Top: The ground-filtered occupancy data is shown in red in the left. The point cloud data from the north and south LiDAR sensors is shown in blue and red respectively. Objects bounding boxes are shown in green. Bottom: Images from the S1 and S2 cameras. 44

5.2 Example of the generated data for the onboard sensor setup. Top: The generated point cloud data from the top-mounted LiDAR sensor is shown in purple, ground-filtered occupancy data is shown in red, and extra sampled points are shown in green. Objects bounding boxes are shown in green. Bottom: The multi-view camera setup. 45

5.3 Example of labelled occupancy grids based on the Nuscenes dataset [Cae+; Ope23]. 46

5.4 Visualization of the meaning of the IoU metric. 46

5.5 Comparison between the 'one-shot' and 'crop' configurations. In the one-shot configuration, the observation point cloud is fed as a whole to the network (top). In the crop configuration, the observation point cloud is divided into smaller crops, which are fed to the network separately (bottom). 48

5.6 Training losses (top) and IoU validation scores (bottom) of the three models trained on the onboard sensor setup. In the loss curves, the number of iterations is shown on the x-axis, while the loss is shown on the y-axis. In the IoU curves, the number of iterations is shown on the x-axis, while the IoU score is shown on the y-axis. The first model (crimson) is a model based on the 'crop' configuration with combined planar and grid feature representations. The second model (turquoise) is a model based on the 'one-shot' configuration with a combined planar and grid feature representation. The third model (blue) is a model based on the crop setting with a grid feature representation only. 49
5.7 Input point clouds as well as images from the multi-view cameras in a sample of the test set. 51
5.8 Qualitative results of the baseline Octomap with single input point cloud (left), and multiple inputs (right). 51
5.9 Qualitative results of the crop (left) and one-shot (right) models with combined feature representation. To better visualize the results, we construct a mesh from the occupied points and render it using Open3d. The meshes are colored based on the height of the points. 52
5.10 The evaluation score on the validation set, the number of iterations is shown on the x-axis, while the IoU score is shown on the y-axis. The first model (grey) is a model based on the 'crop' configuration with combined planar and grid feature representations. The second model based on the crop setting with a grid feature representation only (blue). The model based on the 'one-shot' configuration with a combined planar and grid feature representation (orange). 52
5.11 Input point clouds as well as images from the cameras in a sample of the test set. 53
5.12 Qualitative results of the crop (left) and one-shot (right) models with combined feature representation. To better visualize the results, we construct a mesh from the occupied points and render it using Open3d. The meshes are colored based on the height of the points. 54
5.13 Qualitative results of the baseline Octomap with single input point cloud (left), and multiple inputs (right). 54
5.14 Qualitative results of the crop model with combined feature representation on the A9 dataset. On the left, the image from S1 camera is shown with the input point cloud from the LiDAR sensor projected on it. On the right, the rendered mesh based on the occupancy values of the model is shown. (Note that the view is from an opposite perspective to the one in the left image.) 55
5.15 Flame graphs showing the time spent in each module of the network during inference. The top graph shows the flame graph of the crop model with combined feature representation, the middle graph shows the flame graph of the one-shot model with combined feature representations as well. Finally, the bottom graph shows the flame graph of the crop model with only planar feature representation. 56
5.16 Comparison between crop models with different crop sizes. The performance of the model increases with more crops 58
5.17 Training losses (top) and IoU validation scores (bottom) in the overfitting experiment for the binary (green) and multi-class (orange) models. 60

List of Tables

2.1 Performance matrix of LiDAR and camera sensors in comparison with human driver [Wan21]. 12

5.1 IoU scores of the three models trained on the onboard sensor setup, as well as the Octomap baseline. 50
5.2 IoU scores of the three models trained in the infrastructure setup, as well as the Octomap baseline on the test set. 53
5.3 IoU scores of the three models trained in the infrastructure setup, as well as the Octomap baseline on the test set. 55
5.4 Inference speed for crop models and their respective feature representation shape. 57
5.5 Comparison between one-shot models. The model with the combined feature representation has a significantly lower performance than the model with the plane feature representation and upsampling. 57

Bibliography

[Beh+] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., and Gall, J. *SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences*. URL: https://arxiv.org/pdf/1904.01416.

[BGZuu] Broggi, A., Grisleri, P., and Zani, P. "Sensors technologies for intelligent vehicles perception systems: A comparison between vision and 3D-LIDAR". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE, uuuu-uuuu, pp. 887–892. ISBN: 978-1-4799-2914-6. DOI: 10.1109/ITSC.2013.6728344.

[Bru22] Brucker, M. "Unsupervised LiDAR-based 3D Object Detection Using Infrastructure Sensors". en. MA thesis. Technische Universität München, 2022, p. 78.

[Cae+] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. *nuScenes: A multimodal dataset for autonomous driving*. URL: https://arxiv.org/pdf/1903.11027.

[CC] Cao, A.-Q. and Charette, R. de. *MonoScene: Monocular 3D Semantic Scene Completion*. URL: https://arxiv.org/pdf/2112.00726.

[Car+] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. *End-to-End Object Detection with Transformers*. URL: https://arxiv.org/pdf/2005.12872.

[Che+a] Chen, Y., Liu, J., Zhang, X., Qi, X., and Jia, J. *VoxelNeXt: Fully Sparse VoxelNet for 3D Object Detection and Tracking*. URL: https://arxiv.org/pdf/2303.11301.

[Che+b] Chen, Z., Li, Z., Zhang, S., Fang, L., Jiang, Q., and Zhao, F. *AutoAlignV2: Deformable Feature Aggregation for Dynamic Multi-Modal 3D Object Detection*. URL: https://arxiv.org/pdf/2207.10316.

[Che+c] Cheng, R., Agia, C., Ren, Y., Li, X., and Bingbing, L. *S3CNet: A Sparse Semantic Scene Completion Network for LiDAR Point Clouds*. URL: https://arxiv.org/pdf/2012.09242.

[Cre+] Creß, C., Zimmer, W., Strand, L., Lakshminarasimhan, V., Fortkord, M., Dai, S., and Knoll, A. *A9-Dataset: Multi-Sensor Infrastructure-Based Dataset for Mobility Research*. URL: https://arxiv.org/pdf/2204.06527.

[Den+] Deng, S., Liang, Z., Sun, L., and Jia, K. *VISTA: Boosting 3D Object Detection via Dual Cross-VIew SpaTial Attention*. URL: https://arxiv.org/pdf/2203.09704.

[DWE16] Doherty, K., Wang, J., and Englot, B. "Probabilistic map fusion for fast, incremental occupancy mapping with 3D Hilbert maps". In: *2016 IEEE International Conference on Robotics and Automation (ICRA 2016)*. Piscataway, NJ: IEEE, 2016, pp. 1011–1018. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487233.

[DWE17] Doherty, K., Wang, J., and Englot, B. “Bayesian generalized kernel inference for occupancy map prediction”. In: *ICRA2017*. Danvers (MA): IEEE Computer Society, op. 2017, pp. 3118–3124. ISBN: 978-1-5090-4633-1. DOI: 10.1109/ICRA.2017.7989356.

[Dol+23] Doll, S., Schulz, R., Schneider, L., Benzin, V., Enzweiler, M., and Lensch, H. P. A. “SpatialDETR: Robust Scalable Transformer-Based 3D Object Detection From Multi-view Camera Images With Global Cross-Sensor Attention”. In: *COMPUTER VISION - ECCV 2022*. Ed. by Avidan, S., Brostow, G., Cissé, M., Farinella, G. M., and Hassner, T. Vol. 13699. Lecture Notes in Computer Science. [S.l.]: SPRINGER INTERNATIONAL PU, 2023, pp. 230–245. ISBN: 978-3-031-19841-0. DOI: 10.1007/978-3-031-19842-7{\textunderscore}14.

[Dos+] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. *CARLA: An Open Urban Driving Simulator*. URL: https://arxiv.org/pdf/1711.03938.

[Dua+] Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., and Tian, Q. *CenterNet: Keypoint Triplets for Object Detection*. URL: https://arxiv.org/pdf/1904.08189.

[E D+94] E. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrandt, M. Maurer, F. Thomanek, and J. Schiehlen. “The seeing passenger car ’VaMoRs-P’”. In: *Proceedings of the Intelligent Vehicles ’94 Symposium* (1994). URL: https://www.semanticscholar.org/paper/The-seeing-passenger-car-%27VaMoRs-P%27-Dickmanns-Behringer/b7b5a965cadab8f1ae2f364acfa7eb4127a11400.

[FAN23] FANG-MING. *Occupancy Dataset for NuScenes*. 2023. URL: https://github.com/FANG-MING/occupancy-for-nuscenes (visited on 05/07/2023).

[Gan+] Gan, W., Mo, N., Xu, H., and Yokoya, N. *A Simple Attempt for 3D Occupancy Estimation in Autonomous Driving*. URL: https://arxiv.org/pdf/2303.10076.

[Gei+13] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. “Vision meets robotics: The KITTI dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237. ISSN: 0278-3649. DOI: 10.1177/0278364913491297. URL: https://www.researchgate.net/publication/258140919_Vision_meets_robotics_the_KITTI_dataset.

[Gv] Graham, B. and van der Maaten, L. *Submanifold Sparse Convolutional Networks*. URL: https://arxiv.org/pdf/1706.01307.

[GR18] Guizilini, V. and Ramos, F. “Towards real-time 3D continuous occupancy mapping using Hilbert maps”. In: *The International Journal of Robotics Research* 37.6 (2018), pp. 566–584. ISSN: 0278-3649. DOI: 10.1177/0278364918771476.

[GSRuu] Guizilini, V., Senanayake, R., and Ramos, F. “Dynamic Hilbert Maps: Real-Time Occupancy Predictions in Changing Environments”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, uuuu-uuuu, pp. 4091–4097. ISBN: 978-1-5386-6027-0. DOI: 10.1109/ICRA.2019.8793914.

[He+15] He, K., Zhang, X., Ren, S., and Sun, J. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. Vol. 1502. 2015. DOI: 10.1109/ICCV.2015.123. URL: https://www.researchgate.net/publication/272027131_Delving_Deep_into_Rectifiers_Surpassing_Human-Level_Performance_on_ImageNet_Classification.

[He+] He, K., Zhang, X., Ren, S., and Sun, J. *Deep Residual Learning for Image Recognition*. URL: https://arxiv.org/pdf/1512.03385.

[Hec18] Hecht, J. “Lidar for Self-Driving Cars”. In: *Optics and Photonics News* 29.1 (2018), p. 26. ISSN: 1047-6938. DOI: 10.1364/opn.29.1.000026.

[HUD09] Hernán Badino, Uwe Franke, and David Pfeiffer. “The Stixel World - A Compact Medium Level Representation of the 3D-World”. In: *Pattern recognition*. Ed. by Denzler, J. Lecture Notes in Computer Science. Berlin: Springer, 2009, pp. 51–60. ISBN: 978-3-642-03797-9. DOI: 10.1007/978-3-642-03798-6{\textunderscore}6. URL: https://www.researchgate.net/publication/221113643_The_Stixel_World_-_A_Compact_Medium_Level_Representation_of_the_3D-World.

[Hua+22] Huang, D., Chen, Y., Ding, Y., Liao, J., Liu, J., Wu, K., Nie, Q., Liu, Y., Wang, C., and Li, Z. *Rethinking Dimensionality Reduction in Grid-based 3D Object Detection*. 2022. URL: https://arxiv.org/pdf/2209.09464.

[Hua+a] Huang, J., Huang, G., Zhu, Z., Ye, Y., and Du Dalong. *BEVDet: High-performance Multi-camera 3D Object Detection in Bird-Eye-View*. URL: https://arxiv.org/pdf/2112.11790.

[HC] Huang, Y. and Chen, Y. *Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies*. URL: https://arxiv.org/pdf/2006.06091.

[Hua+b] Huang, Y., Zheng, W., Zhang, Y., Zhou, J., and Lu, J. *Tri-Perspective View for Vision-Based 3D Semantic Occupancy Prediction*. URL: https://arxiv.org/pdf/2302.07817.

[IWJ10] I. Dryanovski, W. Morris, and J. Xiao. “Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 1553–1559. ISBN: 2153-0866. DOI: 10.1109/IROS.2010.5652494.

[Kai+10] Kai M. Wurm, A. Hornung, Maren Bennewitz, C. Stachniss, and W. Burgard. “OctoMap : A Probabilistic , Flexible , and Compact 3 D Map Representation for Robotic Systems”. In: (2010). URL: https://www.semanticscholar.org/paper/OctoMap-%3A-A-Probabilistic-%2C-Flexible-%2C-and-Compact-Wurm-Hornung/f3fa1f32535a1725ca3074482aa8304bdd385064.

[Krä+] Krämmer, A., Schöller, C., Gulati, D., Lakshminarasimhan, V., Kurz, F., Rosenbaum, D., Lenz, C., and Knoll, A. *Providentia – A Large-Scale Sensor System for the Assistance of Autonomous Vehicles and Its Evaluation*. URL: https://arxiv.org/pdf/1906.06789.

[Lan+] Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. *PointPillars: Fast Encoders for Object Detection from Point Clouds*. URL: https://arxiv.org/pdf/1812.05784.

[Lee+] Lee, J., Koh, J., Lee, Y., and Choi, J. W. *D-Align: Dual Query Co-attention Network for 3D Object Detection Based on Multi-frame Point Cloud Sequence*. URL: https://arxiv.org/pdf/2210.00087.

[Li+a] Li, Y., Chen, Y., Qi, X., Li, Z., Sun, J., and Jia, J. *Unifying Voxel-based Representation with Transformer for 3D Object Detection*. URL: https://arxiv.org/pdf/2206.00630.

[Li+22] Li, Y., Ma, D., An, Z., Wang, Z., Zhong, Y., Chen, S., and Feng, C. “V2X-Sim: Multi-Agent Collaborative Perception Dataset and Benchmark for Autonomous Driving”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10914–10921. DOI: 10.1109/LRA.2022.3192802.

[Li+b] Li, Y., Yu, Z., Choy, C., Xiao, C., Alvarez, J. M., Fidler, S., Feng, C., and Anandkumar, A. *VoxFormer: Sparse Voxel Transformer for Camera-based 3D Semantic Scene Completion*. URL: https://arxiv.org/pdf/2302.12251.

[Li+c] Li, Y., Yu, A. W., Meng, T., Caine, B., Ngiam, J., Peng, D., Shen, J., Wu, B., Lu, Y., Zhou, D., Le V, Q., Yuille, A., and Tan, M. *DeepFusion: Lidar-Camera Deep Fusion for Multi-Modal 3D Object Detection*. URL: https://arxiv.org/pdf/2203.08195.

[Li+d] Li, Y., Ge, Z., Yu, G., Yang, J., Wang, Z., Shi, Y., Sun, J., and Li, Z. *BEVDepth: Acquisition of Reliable Depth for Multi-view 3D Object Detection*. URL: https://arxiv.org/pdf/2206.10092.

[LI20] Li, Y. and Ibanez-Guzman, J. "Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems". In: *IEEE Signal Processing Magazine* 37.4 (2020), pp. 50–61. ISSN: 1053-5888. DOI: 10.1109/MSP.2020.2973615.

[Li+e] Li, Z., Wang, W., Li, H., Xie, E., Sima, C., Lu, T., Yu, Q., and Dai, J. *BEVFormer: Learning Bird's-Eye-View Representation from Multi-Camera Images via Spatiotemporal Transformers*. URL: https://arxiv.org/pdf/2203.17270.

[Lia+a] Liang, M., Yang, B., Wang, S., and Urtasun, R. *Deep Continuous Fusion for Multi-Sensor 3D Object Detection*. URL: https://arxiv.org/pdf/2012.10992.

[Lia+b] Liang, T., Xie, H., Yu, K., Xia, Z., Lin, Z., Wang, Y., Tang, T., Wang, B., and Tang, Z. *BEVFusion: A Simple and Robust LiDAR-Camera Fusion Framework*. URL: https://arxiv.org/pdf/2205.13790.

[Lin+] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. *Feature Pyramid Networks for Object Detection*. URL: https://arxiv.org/pdf/1612.03144.

[Lio+] Lionar, S., Emtsev, D., Svilarkovic, D., and Peng, S. *Dynamic Plane Convolutional Occupancy Networks*. URL: https://arxiv.org/pdf/2011.05813.

[Liu+17] Liu, S., Tang, J., Zhang, Z., and Gaudiot, J.-L. "Computer Architectures for Autonomous Driving". In: *Computer* 50.8 (2017), pp. 18–25. ISSN: 0018-9162. DOI: 10.1109/MC.2017.3001256.

[Liu+15] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. *SSD: Single Shot MultiBox Detector*. 2015. DOI: 10.1007/978-3-319-46448-0{\textunderscore}2. URL: https://arxiv.org/pdf/1512.02325.

[Liu+a] Liu, Y., Wang, T., Zhang, X., and Sun, J. *PETR: Position Embedding Transformation for Multi-View 3D Object Detection*. URL: https://arxiv.org/pdf/2203.05625.

[Liu+b] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. URL: https://arxiv.org/pdf/2103.14030.

[Liu+c] Liu, Z., Tang, H., Amini, A., Yang, X., Mao, H., Rus, D., and Han, S. *BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation*. URL: https://arxiv.org/pdf/2205.13542.

[LC87] Lorensen, W. E. and Cline, H. E. "Marching cubes: A high resolution 3D surface construction algorithm". In: *ACM SIGGRAPH Computer Graphics* 21.4 (1987), pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422.

[Lu+] Lu, J., Zhou, Z., Zhu, X., Xu, H., and Zhang, L. *Learning Ego 3D Representation as Ray Tracing*. URL: https://arxiv.org/pdf/2206.04042.

[M M+19] M. Michalkiewicz, J. K. Pontes, D. Jack, M. Baktashmotlagh, and A. Eriksson. "Implicit Surface Representations As Layers in Neural Networks". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4742–4751. ISBN: 2380-7504. DOI: 10.1109/ICCV.2019.00484.

[MHW] Mahmoud, A., Hu, J. S. K., and Waslander, S. L. *Dense Voxel Fusion for 3D Object Detection*. URL: https://arxiv.org/pdf/2203.00871.

[Mar09] Martin Magnusson. *The Three-Dimensional Normal-Distributions Transform — an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. 2009. URL: https://www.researchgate.net/publication/229213868_The_Three-Dimensional_Normal-Distributions_Transform_---_an_Efficient_Representation_for_Registration_Surface_Analysis_and_Loop_Detection.

[McM19] McManamon, P. F. *LiDAR technologies and systems*. Bellingham, Washington, USA: SPIE Press, 2019. ISBN: 9781510625396. DOI: 10.1117/3.2518254. URL: https://www.researchgate.net/publication/334410807_LiDAR_Technologies_and_Systems.

[Mes+] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. *Occupancy Networks: Learning 3D Reconstruction in Function Space*. URL: https://arxiv.org/pdf/1812.03828.

[Mic+] Michalkiewicz, M., Pontes, J. K., Jack, D., Baktashmotlagh, M., and Eriksson, A. *Deep Level Sets: Implicit Surface Representations for 3D Shape Inference*. URL: https://arxiv.org/pdf/1901.06802.

[MMD20] MMDetection3D. *OpenMMLab's Next-generation Platform for General 3D Object Detection*. July 2020. URL: https://github.com/open-mmlab/mmdetection3d.

[Nie+] Niemeyer, M., Mescheder, L., Oechsle, M., and Geiger, A. *Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision*. URL: https://arxiv.org/pdf/1912.07372.

[Ope23] OpenOccupancy. *OpenOccupancy: 3D Occupancy Benchmark for Scene Perception in Autonomous Driving*. Feb. 2023. URL: https://github.com/CVPR2023-Occupancy-Prediction-Challenge/CVPR2023-Occupancy-Prediction-Challenge.

[Par+] Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. URL: https://arxiv.org/pdf/1901.05103.

[PF] Philion, J. and Fidler, S. *Lift, Splat, Shoot: Encoding Images From Arbitrary Camera Rigs by Implicitly Unprojecting to 3D*. URL: https://arxiv.org/pdf/2008.05711.

[Pou+] Poursaeed, O., Fisher, M., Aigerman, N., and Kim, V. G. *Coupling Explicit and Implicit Surface Representations for Generative 3D Modeling*. URL: https://arxiv.org/pdf/2007.10294.

[Pri12] Prince, S. J. D. *Computer vision: Models, learning, and inference*. [Cambridge]: Cambridge University Press, 2012. ISBN: 9781139516709. URL: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=944720.

[Qi+a] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. URL: https://arxiv.org/pdf/1612.00593.

[Qi+b] Qi, C. R., Yi, L., Su, H., and Guibas, L. J. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. URL: https://arxiv.org/pdf/1706.02413.

[Raj+20] Raj, T., Hashim, F. H., Huddin, A. B., Ibrahim, M. F., and Hussain, A. "A Survey on LiDAR Scanning Mechanisms". In: *Electronics* 9.5 (2020), p. 741. ISSN: 2079-9292. DOI: 10.3390/electronics9050741. URL: https://www.mdpi.com/2079-9292/9/5/741.

[Ren+] Ren, S., He, K., Girshick, R., and Sun, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. URL: https://arxiv.org/pdf/1506.01497.

[Ris+] Rist, C. B., Emmerichs, D., Enzweiler, M., and Gavrila, D. M. *Semantic Scene Completion using Local Deep Implicit Functions on LiDAR Data*. URL: https://arxiv.org/pdf/2011.09141.

[RCV] Roldão, L., Charette, R. de, and Verroust-Blondet, A. *LMSCNet: Lightweight Multiscale 3D Semantic Completion*. URL: https://arxiv.org/pdf/2008.10559.

[RFB] Ronneberger, O., Fischer, P., and Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. URL: https://arxiv.org/pdf/1505.04597.

[RCG22] Roriz, R., Cabral, J., and Gomes, T. "Automotive LiDAR Technology: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (2022), pp. 6282–6297. ISSN: 1558-0016. DOI: 10.1109/TITS.2021.3086804.

[RVK] Rukhovich, D., Vorontsova, A., and Konushin, A. *ImVoxelNet: Image to Voxels Projection for Monocular and Multi-View General-Purpose 3D Object Detection*. URL: https://arxiv.org/pdf/2106.01178.

[Saa+13a] Saarinen, J., Andreasson, H., Stoyanov, T., Ala-Luhtala, J., and Lilienthal, A. J. "Normal Distributions Transform Occupancy Maps: Application to large-scale online 3D mapping". In: *ICRA 2013*. [Piscataway, N.J.]: IEEE, 2013, pp. 2233–2238. ISBN: 978-1-4673-5643-5. DOI: 10.1109/ICRA.2013.6630878.

[Saa+uu] Saarinen, J., Stoyanov, T., Andreasson, H., and Lilienthal, A. J. "Fast 3D mapping in highly dynamic environments using normal distributions transform occupancy maps". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, uuuu-uuuu, pp. 4694–4701. ISBN: 978-1-4673-6358-7. DOI: 10.1109/IROS.2013.6697032.

[Saa+13b] Saarinen, J. P., Andreasson, H., Stoyanov, T., and Lilienthal, A. J. "3D normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments". In: *The International Journal of Robotics Research* 32.14 (2013), pp. 1627–1644. ISSN: 0278-3649. DOI: 10.1177/0278364913499415.

[Shi+] Shi, S., Guo, C., Yang, J., and Li, H. *PV-RCNN: The Top-Performing LiDAR-only Solutions for 3D Detection / 3D Tracking / Domain Adaptation of Waymo Open Dataset Challenges*. URL: https://arxiv.org/pdf/2008.12599.

[Sie+19] Siegfried Seebacher, Bernd Datler, Jacqueline Erhart, Gerhard Greiner, and Martin Ullrich. "Infrastructure data fusion for validation and future enhancements of autonomous vehicles' perception on Austrian motorways". In: 2019, pp. 1–7. DOI: 10.1109/ICCVE45908.2019.8965142. URL: https://www.researchgate.net/publication/338800835_Infrastructure_data_fusion_for_validation_and_future_enhancements_of_autonomous_vehicles'_perception_on_Austrian_motorways.

[SZT] Sindagi, V. A., Zhou, Y., and Tuzel, O. "MVX-Net: Multimodal VoxelNet for 3D Object Detection". In: *International Conference on Robotics and Automation (ICRA)* (). URL: https://arxiv.org/pdf/1904.01649.

[Son+] Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. *Semantic Scene Completion from a Single Depth Image*. URL: https://arxiv.org/pdf/1611.08974.

[Son+20] Songyou Peng, Michael Niemeyer, Lars M. Mescheder, M. Pollefeys, and Andreas Geiger. "Convolutional Occupancy Networks". In: *undefined* (2020). URL: https://www.semanticscholar.org/paper/Convolutional-Occupancy-Networks-Peng-Niemeyer/5a6732513a1dc0bea059543f208a7556e3e31067.

[Sun+] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhao, S., Cheng, S., Zhang, Y., Shlens, J., Chen, Z., and Anguelov, D. *Scalability in Perception for Autonomous Driving: Waymo Open Dataset*. URL: https://arxiv.org/pdf/1912.04838.

[Tes22a] Tesla. *Tesla AI day 2022 Occupancy Networks for 3D Perception*. 2022. URL: https://www.youtube.com/watch?v=ODSJsviD_SU&t=4331s&ab_channel=Tesla (visited on 05/07/2023).

[Tes22b] Tesla. *Tesla CVPR'22 Keynote Occupancy Networks for 3D Perception*. 2022. URL: https://www.youtube.com/watch?v=jPCV4GKX9Dw&ab_channel=WADatCVPR (visited on 05/07/2023).

[Tia+] Tian, X., Jiang, T., Yun, L., Wang, Y., Wang, Y., and Zhao, H. *Occ3D: A Large-Scale 3D Occupancy Prediction Benchmark for Autonomous Driving*. URL: https://arxiv.org/pdf/2304.14365.

[Tob+18] Tobias Fleck, Karam Daaboul, Michael Weber, Philip Schoerner, and J. Marius Zöllner. "Towards Large Scale Urban Traffic Reference Data: Smart Infrastructure in the Test Area Autonomous Driving Baden-Württemberg". In: 2018. URL: https://www.researchgate.net/publication/327449884_Towards_Large_Scale_Urban_Traffic_Reference_Data_Smart_Infrastructure_in_the_Test_Area_Autonomous_Driving_Baden-Wurttemberg.

[Tse+] Tseng, C.-Y., Chen, Y.-R., Lee, H.-Y., Wu, T.-H., Chen, W.-C., and Hsu, W. H. *CrossDTR: Cross-view and Depth-guided Transformers for 3D Object Detection*. URL: https://arxiv.org/pdf/2209.13507.

[Vas+] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. *Attention Is All You Need*. URL: https://arxiv.org/pdf/1706.03762.

[WE16] Wang, J. and Englot, B. "Fast, accurate gaussian process occupancy maps via test-data octrees and nested Bayesian fusion". In: *2016 IEEE International Conference on Robotics and Automation (ICRA 2016)*. Piscataway, NJ: IEEE, 2016, pp. 1003–1010. ISBN: 978-1-4673-8026-3. DOI: 10.1109/ICRA.2016.7487232.

[Wan21] Wang, P. "Research on Comparison of LiDAR and Camera in Autonomous Driving". In: *Journal of Physics: Conference Series* 2093.1 (2021), p. 012032. ISSN: 1742-6596. DOI: 10.1088/1742-6596/2093/1/012032. URL: https://iopscience.iop.org/article/10.1088/1742-6596/2093/1/012032/meta.

[Wan+] Wang, Y., Guizilini, V., Zhang, T., Wang, Y., Zhao, H., and Solomon, J. *DETR3D: 3D Object Detection from Multi-view Images via 3D-to-2D Queries*. URL: https://arxiv.org/pdf/2110.06922.

[War19] Warren, M. E. "Automotive LIDAR Technology". In: *2019 Symposium on VLSI Circuits*. [Place of publication not identified]: IEEE, 2019, pp. C254–C255. ISBN: 978-4-86348-720-8. DOI: 10.23919/VLSIC.2019.8777993.

[Wei+] Wei, Y., Zhao, L., Zheng, W., Zhu, Z., Zhou, J., and Lu, J. *SurroundOcc: Multi-Camera 3D Occupancy Prediction for Autonomous Driving*. URL: https://arxiv.org/pdf/2303.09551.

[Wu+] Wu, X., Peng, L., Yang, H., Xie, L., Huang, C., Deng, C., Liu, H., and Cai, D. *Sparse Fuse Dense: Towards High Quality 3D Detection with Depth Completion*. URL: https://arxiv.org/pdf/2203.09780.

[Yan+a] Yan, X., Gao, J., Li, J., Zhang, R., Li, Z., Huang, R., and Cui, S. *Sparse Single Sweep LiDAR Point Cloud Segmentation via Learning Contextual Shape Priors from Scene Completion*. URL: https://arxiv.org/pdf/2012.03762.

[YYB18] Yan Yan, Yuxing Mao, and Bo Li. "SECOND: Sparsely Embedded Convolutional Detection". In: *Sensors (Basel, Switzerland)* (2018). URL: https://www.semanticscholar.org/paper/SECOND%3A-Sparsely-Embedded-Convolutional-Detection-Yan-Mao/5125a16039cabc6320c908a4764f32596e018ad3.

[Yan+b] Yang, Z., Sun, Y., Liu, S., Shen, X., and Jia, J. *STD: Sparse-to-Dense 3D Object Detector for Point Cloud*. URL: https://arxiv.org/pdf/1907.10471.

[YZKa] Yin, T., Zhou, X., and Krähenbühl, P. "Center-based 3D Object Detection and Tracking". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (). URL: https://arxiv.org/pdf/2006.11275.

[YZKb] Yin, T., Zhou, X., and Krähenbühl, P. *Multimodal Virtual Point 3D Detection*. URL: https://arxiv.org/pdf/2111.06881.

[YSA18] Yiyi Liao, S. Donné, and Andreas Geiger. "Deep Marching Cubes: Learning Explicit Surface Representations". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018). URL: https://www.semanticscholar.org/paper/Deep-Marching-Cubes%3A-Learning-Explicit-Surface-Liao-Donn%C3%A9/7b183c11f1ba13f05db2050ab1abb1fa9f52c9a8.

[Yoo+20] Yoo, J. H., Kim, Y., Kim, J., and Choi, J. W. *3D-CVF: Generating Joint Camera and LiDAR Features Using Cross-view Spatial Feature Fusion for 3D Object Detection*. 2020. DOI: 10.1007/978-3-030-58583-9{\textunderscore}43. URL: https://arxiv.org/pdf/2004.12636.

[Zha+a] Zhang, R., Qiu, H., Wang, T., Guo, Z., Xu, X., Qiao, Y., Gao, P., and Li, H. *MonoDETR: Depth-guided Transformer for Monocular 3D Object Detection*. URL: https://arxiv.org/pdf/2203.13310.

[ZZD] Zhang, Y., Zhu, Z., and Du Dalong. *OccFormer: Dual-path Transformer for Vision-based 3D Semantic Occupancy Prediction*. URL: https://arxiv.org/pdf/2304.05316.

[Zha+b] Zhang, Y., Zhu, Z., Zheng, W., Huang, J., Huang, G., Zhou, J., and Lu, J. *BEVerse: Unified Perception and Prediction in Birds-Eye-View for Vision-Centric Autonomous Driving*. URL: https://arxiv.org/pdf/2205.09743.

[Zhe+] Zheng, W., Tang, W., Jiang, L., and Fu, C.-W. *SE-SSD: Self-Ensembling Single-Stage Object Detector From Point Cloud*. URL: https://arxiv.org/pdf/2104.09804.

[ZT] Zhou, Y. and Tuzel, O. *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. URL: https://arxiv.org/pdf/1711.06396.