Bird's Eye View Perception Strategy for Indoor Autonomous Ground Vehicles Using Only Lidar

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Robotics and Autonomous Systems
May 2025
Luuka Lindgren

Supervisors:
Prof. Tomi Westerlund
MSc. Haichuan Li

UNIVERSITY OF TURKU
Department of Computing

Luuka Lindgren: Bird's Eye View Perception Strategy for Indoor Autonomous Ground Vehicles Using Only Lidar

Master of Science (Tech) Thesis, 65 p.
Robotics and Autonomous Systems
May 2025

The rising popularity of Light Detection and Ranging (lidar) sensors in autonomous driving and robotics have sparked the interest of indoor usages. Previously these sensors were bulky and expensive, but recent technological advancements have broaden their applications. This thesis delves into bird's eye view (BEV) strategy utilization in autonomous ground vehicles (AGV), with a focus on lidar-only systems used indoors. With a help of conducted profound literature review, we aim to follow design science research methodology by creating artefacts relating to the topics.
Our contributions consists of three artefacts: a novel indoor lidar dataset, a BEV perception framework, and experiments and comparisons of these and existing systems. The point clouds for the dataset were collected from kitchen and lounge areas with a 360 degree lidar sensor, and the indoor objects of the chosen 520 frames were annotated by hand. Proposed dataset was formatted to closely follow KITTI datasets structure aiming to be easily adoptable to existing algorithms. The proposed BEV perception framework was conducted from MaskBEV algorithm by adapting it from outdoor vehicle detection to indoor object detection. Original code required multiple modifications, primarily due to our custom indoor object classes. Additionally, we trained a machine learning model with this modified MaskBEV using the custom dataset and conducted experiments and tests with it. Resulting in varying images that were used to conduct qualitative analysis, and compared with the results of model trained with the open-source dataset KITTI.
Even though our research results were not satisfactory in terms of reliability and accuracy of the perception framework, the research highlights the potential of utilizing BEV strategies with AGVs with minimal sensor setup. Our indoor dataset works as a prototype and a motivator for future works. By shifting the focus from outdoor to indoor, we try to broaden the scope of robotics and autonomous driving researches.

Keywords: Lidar, Perception, AGV, BEV, Machine Learning

Contents

List of Figures

List of Tables

1 Introduction

Autonomous ground vehicles (AGVs) rely heavily on accurate perception of their surroundings, making efficient and safe navigation a critical aspect of their design. While significant progress has been made in outdoor environments — driven by large-scale investments and efforts from well-known tech companies such as Tesla and Google — indoor navigation remains relatively understudied. Yet the need for reliable indoor AGV solutions is growing rapidly, with applications in warehouses, hospitals, factories, and commercial buildings. These environments often presents unique constraints, such as limited Global Navigation Satellite System (GNSS) availability, narrow pathways, diverse obstacles, and dynamic layouts.

Bird's eye view (BEV) is a perception strategy that provides a self-referenced, top-down view of the vehicle and its immediate surroundings [1]. This short-range 360° field of view is particularly advantageous for indoor AGVs, which typically navigate in cluttered, dynamic spaces at relatively lower speeds. By providing an overhead perspective, BEV simplifies object detection, path planning, and collision avoidance. It effectively abstracts the complexities of 3D point cloud data into two-dimensional, map-like view that can be more straightforward to process. Despite these advantages, many existing solutions rely on multiple sensors, including cameras and radars in addition to Light Detection and Ranging (lidar), to handle complex scenarios and enhance systems robustness. In contrast, this thesis focuses on a lidar-only approach, exploring how BEV-based solutions can meet the demands of indoor

environments without the added complexity of additional sensors.

Lidar sensors have become increasingly popular in autonomous driving and robotics applications. Originally employed in meteorology and environmental studies [2]–[4], lidar technology is now more widely available and cost-effective. Modern sensors can generate dense point clouds with high resolution at competitive prices, making them suitable for both research and practical implementations in autonomous driving [5]. In indoor settings, lidar sensors are especially beneficial due to their resilience to lightning changes and their accurate range measurements, making them an ideal choice for this research.

However, most existing lidar datasets and perception algorithms are designed for outdoor use, focusing primarily on cars and pedestrians in large-scale urban environments. By contrast, indoor AGVs face smaller-scale obstacles (such as chairs, tables, boxes, or shelves) and operate in tighter spaces with artificial lighting. Publicly available indoor lidar datasets remain scarce, limiting the validation and benchmarking of lidar-based indoor perception systems.

In this thesis, we propose a bird's eye view perception strategy for robust and reliable indoor AGV navigation that relies exclusively on lidar data. To address the shortage of relevant datasets — and as an illustrative example — we collect a novel indoor lidar dataset featuring multiple object classes commonly encountered in real-world indoor environments. We also adapt an existing state-of-the-art machine learning algorithm, originally developed for outdoor scenarios (MaskBEV), to handle the unique challenges of indoor settings. Ultimately, this work demonstrates that an indoor-focused, lidar-only approach can achieve robust 3D object detection.

1.1 Significance and Motivation

Demand for logistics automation is growing rapidly alongside advancements in information technology, making indoor AGVs a compelling research and development

topic. However, this increased focus is still overshadowed by the more commercialized interest in outdoor autonomous vehicles, such as self-driving cars and trucks. Recent developments in lidar sensors have made them a strong choice for indoor perception, and the top-down BEV representation offers multiple benefits for these type of environments.

These considerations lead to the following research questions:

1. Why are BEV strategies widely used in perception?

2. How can BEV strategies with lidar enhance obstacle detection and navigation capabilities of indoors AGVs?

1.2 Related Works

Numerous projects and papers have been published on the use of Bird's Eye View (BEV) strategies in autonomous driving, each presenting its own perspectives and proposals. MaskBEV [6] is one of the most recent works in this domain and serves as a primary inspiration for this thesis. The authors introduce a novel architecture, MaskBEV, which showcases the benefits of using masks instead of bounding boxes for 3D object detection. This approach supports simultaneous detection and object footprint completion, making the detection framework robust to occlusions and missing signals. Its performance was evaluated using the SemanticKITTI and KITTI datasets, reflecting its outdoor focus. Our work builds on these ideas and adapts MaskBEV for indoor settings.

BEV offers multiple advantages over other perception strategies, including an intuitive 2D representation of 3D space and straightforward sensor fusion. A recent review [7] provides practical guidance on maximizing BEV's potential in autonomous driving. This survey presents a comprehensive overview of recent BEV-based perception methods, a detailed literature review, and advice on improving performance

across various BEV perception tasks. In general, BEV is an unified view representation that can be derived from a range of sensors and data types, leading to specialized subcategories such as BEV lidar, BEV camera, and BEV fusion.

Additional research [8] focusing on bev strategies for large industrial AGVs highlights the diverse real-world opportunities for BEV perception. This paper addresses the gap between industrial AGV automation and the more common passenger-car automation on the public roads, making it relatively similar to our research. It proposes a labeled dataset drawn from their own real industrial park environment. The authors also introduce state-of-the-art framework BEVTrajNet for dynamic object classification and trajectory prediction. The BEVTrajNet only uses voxelized 3D point cloud data from lidar, and their experiments rely on the nuScenes dataset for training and validation. The proposed industrial park dataset was also used to validate the diversity of the model. Simulation results demonstrate reliable perception outcomes in terms of computational efficiency and prediction accuracy, further illustrating the power of BEV-based approaches in diverse settings.

1.3 Contribution

This thesis investigates indoor perception using a bird's eye view strategy. We build on existing research by constructing our own indoor dataset and modifying an existing outdoor machine learning algorithm, MaskBEV [6], to make it effective in indoor contexts. We also compare results from our custom dataset with those obtained using a well-know open-source dataset, KITTI [9], thereby evaluating BEV performance across both indoor and outdoor scenarios. The main contributions of this thesis are:

- **Indoor Lidar Dataset:** 520 frames captured with a Livox Mid-360 lidar sensor on a RC car navigating kitchen and lounge spaces. The dataset is

manually annotated with eight relevant indoor object classes and stored in KITTI-compatible format.

- **Indoor BEV Perception Framework:** An indoor-adapted version of MaskBEV algorithm. The framework includes customized preprocessing, augmentation, and training to handle the clutter and short-range sensing typical to indoor environments.

- **Cross-domain evaluation:** Experimental processes comparing our framework on the proposed indoor dataset to the original MaskBEV on the KITTI dataset. Illustrating how BEV-based perception strategies can perform in both indoor and outdoor conditions.

1.4 Structure

This thesis is divided into 6 chapters as follows:

- **Chapter 2: Background** provides detailed explanations of the core technologies and concepts involved in the research.

- **Chapter 3: Design Overview** presents an overall picture of the hardware, software and dataset design employed in this work. It focuses on what was employed and created.

- **Chapter 4: Implementation** describes the practical steps taken during development, including dataset creation, BEV-based machine learning modifications, model training, and testing. It focuses on how things were employed and created.

- **Chapter 5: Results** analyzes the experimental outcomes of the proposed approach, and comparisons with the results of MaskBEV with open-source dataset.

- **Chapter 6: Conclusion and Future Work** summarizes the key findings and contributions of this research and suggests directions for further study.

2 Background

In this chapter, we perform an extensive exploration into the core topics and concepts associated with lidar-only BEV perception strategies and AGV operating in indoor setting. These topics include diving into details of lidar sensors, processing of 3D point clouds, object detection, BEV, and datasets relating to autonomous driving.

2.1 Lidar Sensor

Lidar is a remote sensing technology that plays a critical role in autonomous driving and mobility. Originally, lidar was utilized in various applications outside the automotive industry, including meteorology [2] and environmental studies, such as landslide detection [3] and forest ecosystem monitoring [4]. Lidar sensor outputs data in format called point cloud, example of this is shown in Figure 2.1.

Lidar operates on the principle of time-of-flight (ToF) measurement. It involves transmitting a light pulse toward a target surface and measuring the time it takes for the reflection to return to the sensor [5]. This measurement is used to calculate the distance to the surface, providing precise spatial information about the environment. ToF equation is:

$$R = \frac{1}{2} c \Delta t \tag{2.1}$$

where c is the speed-of-light constant and the Δt is the elapsed time of the light pulse.

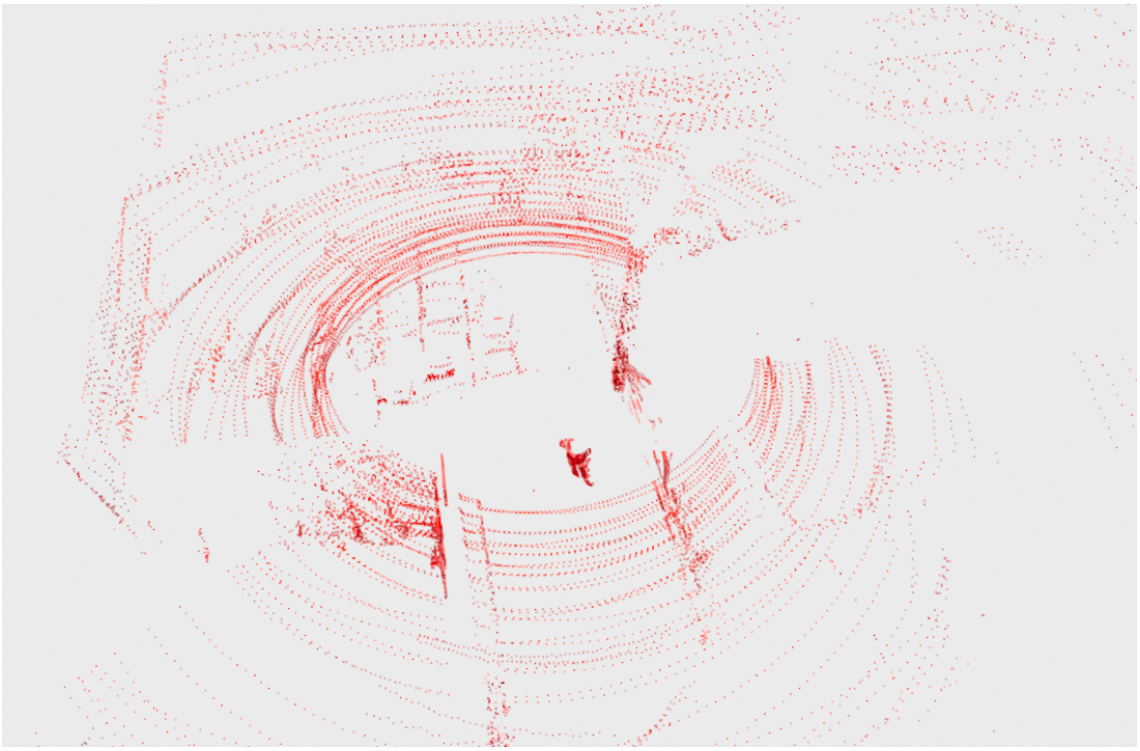

Figure 2.1: Example lidar Point Cloud data visualized with Open3D tool from our custom dataset.

Additionally, lidar sensors can gather spectral information through the analysis of laser return intensity, which is the measure of the surface's reflectance characteristics. This helps recognizing the material of the reflection surface.

A fundamental aspect of lidar technology is its ability to produce one-dimensional distance data between the sensor and the target points [5]. This capability is enhanced by employing various beam deflection techniques, known as scanning methods. These methods allow the lidar system to compile two-dimensional (2D) or three-dimensional (3D) spatial data. The field of view (FOV) of a lidar sensor, which is defined by its scanning methods, determines the extent of the area covered by the sensor. Similarly, the angular resolution specifies the density of the scan lines, thereby affecting the detail and accuracy of the point cloud data captured. In short, lidar emits laser light from a transmitter and after reflection, this returning light is detected by a receiver. The principles of lidar sensor are illustrated in Figure 2.2.

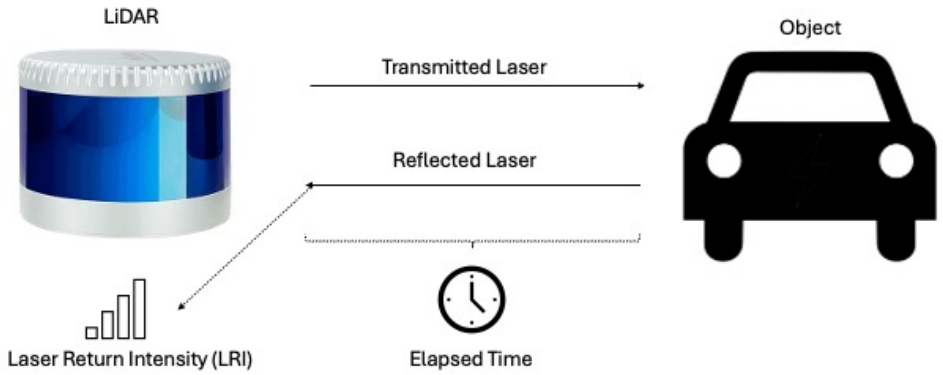

Figure 2.2: Principles of lidar sensor. Elapsed time and intensity of the returning laser are captured and utilized for further tasks.

Autonomous driving applications benefit from lidar due to its versatility, suitability and accuracy [5]. Recent technical advancements have not only enhanced the capabilities of lidar systems but have also made them more affordable and accessible. For instance, improvements in solid-state technology have reduced the size and power consumption of lidar sensors, making them ideal for integration into robotics and autonomous vehicles. However, lidar technology is based on light emission and reflection, thus being susceptible to interference from weather conditions such as rain, snow, fog, and dust. These elements can diffract the laser pulses, reducing the sensor's effectiveness and accuracy [10], making it more suitable for indoor environments where such adverse conditions are absent.

While providing detailed and accurate spatial and spectral data, lidar is often insufficient on its own for robust autonomous navigation. The integration of additional sensors into the system — a process known as sensor fusion — greatly enhances both the accuracy and reliability of the system [10]. Numerous works demonstrate the benefits of sensor fusion [11]–[14], which combines the strengths of

various sensors while mitigating their individual weaknesses. For example, Zhang et al. [11] illustrates in their research that incorporating a stereo camera alongside lidar significantly lowers the false alarm rate in object detection and therefore making the perception system more reliable. Such enhancements are critical in complex environments where the ability to accurately detect dynamic obstacles are crucial.

2.2 3D Point Cloud Preprocessing

Point clouds are data format that represent discrete set of data points in three dimensional space. Each of these data points consist of multiple measurements, making point clouds rich in information, but complex to be effectively utilized.

2.2.1 Introduction to Lidar Data Preprocessing

Lidar generates a vast amount of data to represent the 3D point cloud, which must to be processed before it can be effectively utilized in autonomous driving tasks [15]. A typical lidar dataset may contain millions of discrete points that lack semantic information. While manual processing is tedious and time-consuming, automated data processing is crucial component of the workflow. Efficient data handling methods are therefore essential to extract meaningful insights from the raw point cloud, enabling tasks such as object detection, segmentation, and navigation [16]. Processing 3D point clouds involves analyzing and refining this data to optimize its storage, usability, and quality, using varying computational algorithms. Many of these tasks extend from 1D signal processing and 2D image processing techniques but are more complex due to the additional spatial dimension(s) (x, y, z) involved. A common approach is to convert 3D point clouds into a structured 2D grid, facilitating compatibility with deep neural networks [17].

Himmelsbach et al. [18] identify three primary stages of lidar-based 3D percep-

tion: segmentation, classification, and tracking. This section, however, concentrates on the foundational phase preceding these: data preprocessing. This crucial phase involves essential processes such as noise reduction and filtering, which are necessary to prepare the data for reliable usage in autonomous object detection tasks. Segmentation, classification and tracking are discussed in Section 2.3.

2.2.2 Noise Reduction and Filtering

Lidars are sensors, and every sensor is susceptible to extraneous data, often referred to as noise, which can distort the accurate representation of the environment. Because lidar is based on light emission and sensing technology, it is particularly affected by disadvantageous weather conditions like snow, fog, and rain [19]. While a controlled indoor environment inherently avoids weather disturbances, practical outdoor deployments must instead counter these effects with robust denoising and filtering techniques to improve data accuracy and reliability. Furthermore, sensor noise can result from intrinsic properties of the acquisition device [20] and external factors such as light sources and reflective surfaces [21]. It is therefore essential to conduct varying filtering operations on raw point clouds to acquire data suitable for further processing. Taxonomy of different denoising methods is illustrated in Figure 2.3.

For effective denoising, algorithms such as Statistical Outlier Removal (SOR) and Radius Outlier Removal (ROR) are frequently used to eliminate isolated points that deviate significantly from neighboring points [22]. These methods improve point cloud clarity, enabling a more accurate representation of environmental features. While SOR and ROR perform efficiently in homogeneous data settings, more dynamic versions — Dynamic Statistical Outlier Removal (DSOR) and Dynamic Radius Outlier Removal (DROR) — have been developed to adapt to more irregular environments. These dynamic filters account for point density relative to the

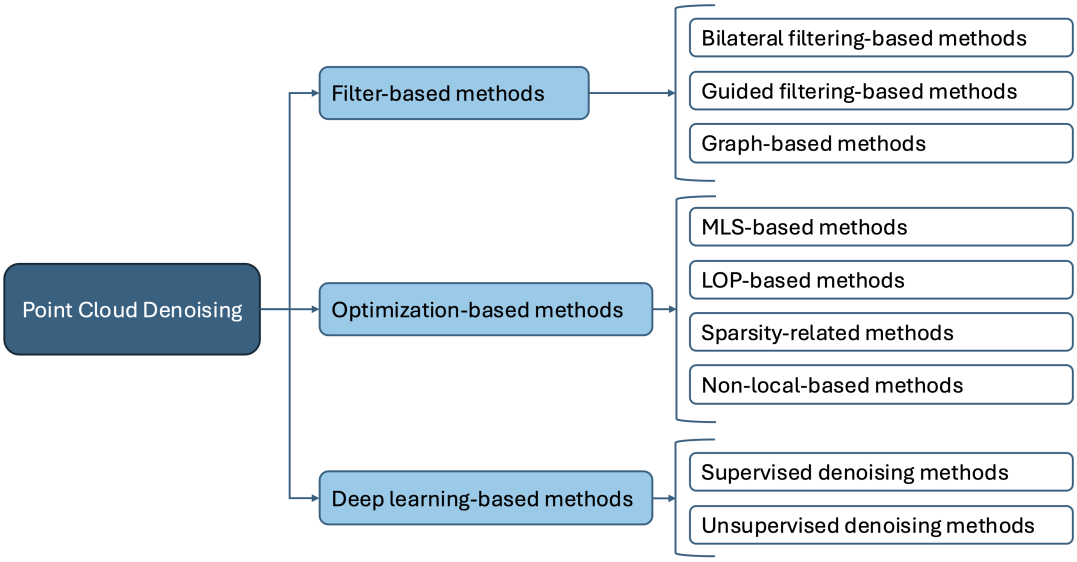

Figure 2.3: Taxonomy of point cloud denoising methods in a broader scope [25].

lidar's distance, providing more robust filtering in challenging scenarios [23], [24]. Figure 2.4 (a) illustrates how these can be used to filter out snowflakes from outdoor lidar data.

The latest innovations in point cloud denoising involve deep learning-based methods, such as the PointNet-based PCPNet [26] and convolution-based GPDNet [27]. These approaches require pre-training with noisy inputs and ground-truth data in an offline setting before they can be implemented in runtime environments. These models provide robust solutions that improve data reliability in complex or densely populated environments [25].

In addition to data denoising, filtering is a critical step in point cloud preprocessing. In autonomous driving settings, most objects of interest are perpendicular to flat ground, making object detection algorithms benefit significantly from ground filtering [28]. Ground filtering involves separating terrain and other ground points from non-ground points, such as buildings and vehicles. By removing ground points from subsequent processing steps, the data size can be significantly reduced, as

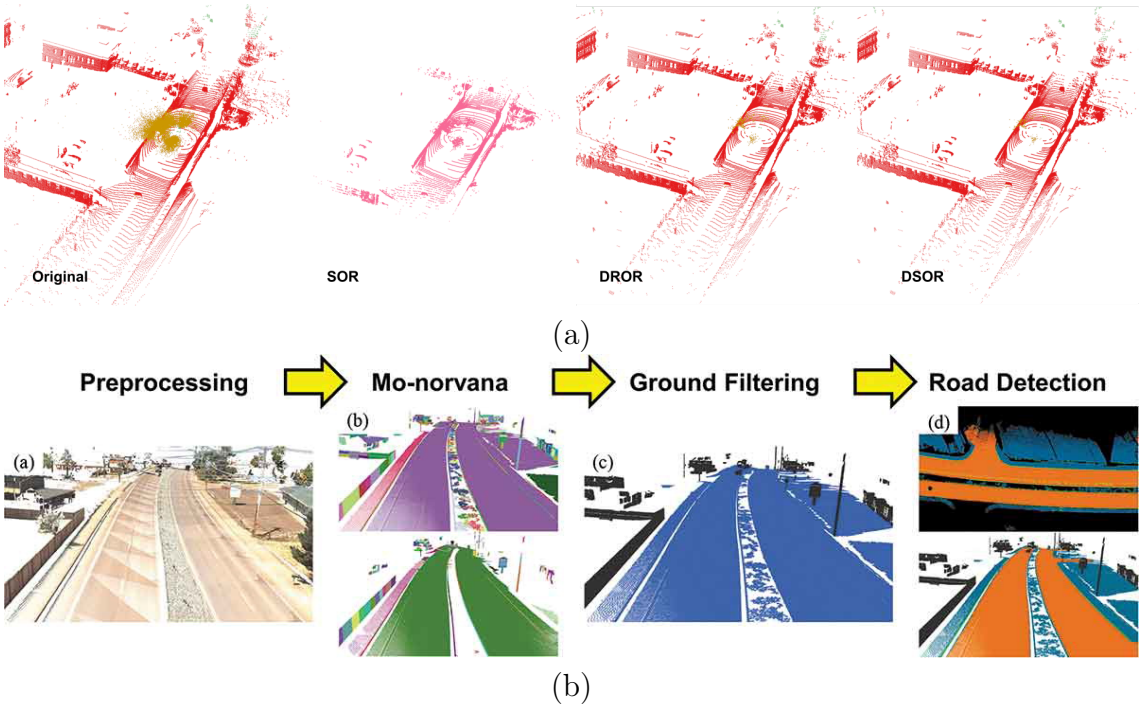

Figure 2.4: Noise reduction and filtering examples with DSOR (a) [23] *(Copyright © 2021, arXiv)*, and Ground filtering (b) [15] *(Copyright © 2021, Taylor & Francis)*. This example of DSOR shows the removal on snowfall from the point clouds and compares effectiveness to other existing methods. This example of ground filtering shows its usability in road detection.

ground points often constitute the majority of the dataset [15].

In dynamic, natural environments, removing all ground points might not always be viable, as subtle changes in terrain could be important for accurate perception. On the contrary, in controlled environments, such as warehouses, where ground surfaces are generally uniform, applying ground filtering can be highly advantageous, leading to considerable computational efficiency and improved algorithm performance. For example, Che et al. [15], [29] proposed an efficient segment-based ground filtering method that utilizes the Mo-norvana framework. This method achieves enhanced accuracy by segmenting the ground into smaller, manageable components, for precise filtering, making it suitable for autonomous driving applications.

2.3 Object Detection

Object detection is a critical part of perception strategy in autonomous driving. By identifying objects from irregular point clouds, object detection lays the foundation for advanced tasks, such as scene understanding and path planning.

2.3.1 Pipeline of Object Detection

After the denoising and filtering processes, the refined 3D point cloud data is almost ready for more complex tasks, such as object detection, which is critical for autonomous vehicle navigation [16]. Object detection can utilize both 2D and 3D methods; however, the focus here is primarily on 3D detection where the input data is in the for of 3D point clouds. In 3D object detection, deep learning models are beneficial due to their ability to learn and improve from experiences automatically without being specifically programmed [30]. Deep learning, a subset of machine learning, involves human brain-like neural networks which consists of multiple layers that can process vast amounts of data.

While some 3D object detection methods may compress the third dimension to convert it into a 2D image format to streamline processing, these are still fundamentally based on 3D data structures and are classified as 3D object detection methods [31]. The use of deep learning in these processes is essential because it enables the efficient handling of large, complex datasets of 3D point clouds [32].

The pipeline of 3D object detection is visualized in Figure 2.5. This process begins with Data Representation, where data can be structured in various forms, including voxel-based, point-based, graph-based, and projection-based methods [31], [33]. Each having their strengths and should be chosen based on the specific requirements of the desired detection algorithm and the available computational resources. The next step, Feature Extraction, which can also be achieved through different ways, such as segment-wise, point-wise, graph-FCNs (Fully Convolutional

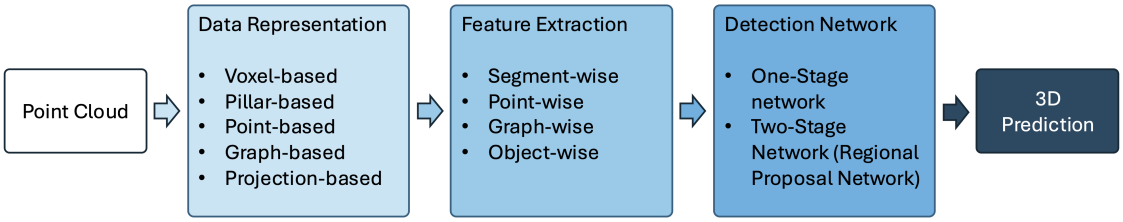

Figure 2.5: Pipeline of object detection. One option from each step is chosen to build object detection framework.

Networks), and object-wise feature extraction [31]. Following feature extraction, the data proceeds to the Detection Network. Depending on the system's needs, this can involve one-stage or two-stage detectors [16], [34]. The latter, which utilizes Regions of Interest (ROI), tend to provide higher accuracy but at the cost of longer inference times.

2.3.2 Data Representation

Raw point cloud data is distributed arbitrarily in 3D space, which makes the data set unordered and irregular. This characteristic emphasizes the importance of structuring the data prior to object detection. There are multiple different approaches to handle the inconsistency of the point cloud data, these methods can be categorized into five classes summarized in the Table 2.1.

3D voxelization is employed to discretize the 3D space into uniform voxel grids, which inherently provide a hierarchical structure [16]. These voxel-based data representation methods offer a regular format for the data that preserves the three-dimensional structure of the raw point cloud. Importantly, voxelization does not alter the points within the 3D space; it results in either a zero voxel or non-zero voxel if at least one point occupies the voxel [31]. This process often leads to a significant number of empty voxels, with the remaining voxels being sparsely pop-

Table 2.1: Summary of Data Representation methods.

Method	Main Ideas	Pros	Cons
Voxel-based	Discretize 3D space into uniform voxel grids. Each voxel can store occupancy or aggregated features.	Preserves 3D structure, facilitates handling occlusion, regular grid enables CNN-based processing	Memory-intensive (many empty voxels), fine resolution means exponential growth in voxel count
Pillar-based	Compress the voxel voxel grid along the vertical axis, forming "pillars" processed by 2D convolutions.	More efficient, than full 3D voxel grids, leverages 2D CNNs, reduced memory footprint	Loses some vertical information, potential decrease in accuracy for tall objects
Point-based	Uses raw or minimally downsampled point clouds directly.	Retains original geometry, flexible sampling strategies	High computational cost, complex feature extraction pipelines
Graph-based	Represents points as nodes in a graph and edges capture neighborhood relationships	Retains local neighborhood structure, can capture fine geometric relationships, robust feature extraction	Complex to implement, can be computationally heavy for large datasets
Projection-based	Project 3D point clouds into 2D pseudo-images (e.g. BEV, front view)	Can leverage mature 2D CNNs, fast inference, easy to fuse with camera data	Loses some 3D geometric detail, complexity can add up, possible distortions or overlaps in projections

ulated and containing irregularly distributed points. The primary advantages of voxelization include reduced dimensionality of the point clouds and improved occlusion handling, as the voxel structure facilitates capturing details about the spatial relationships between objects. However, the main drawback is its memory intensity; computational demands increase exponentially with the presence of empty voxels.

Pillar-based data representation builds on the concept of voxelization but replaces 3D convolutions with a structure of vertical columns known as pillars [31]. This approach compresses the 3D data into 2D format, enabling the use of 2D convolutions which substantially simplify the computational complexity [34]. Drawbacks of this can be the loss of vertical information and accuracy, especially for objects

with great heights. At the same time, this allows for more memory-efficient data processing.

Point-based data representation involve utilizing the raw 3D point cloud data either as-is or after downsampling it to a more compact form through techniques such as random sampling or furthest point sampling (FPS) [31]. The main advantage of this point-based representation are that it is straightforward process and it maintains all, or slightly reduced portion, of the original information from the 3D point cloud to pass the input to next processing steps [16]. However, because of the minimal processing of the input data, extracting features from these representative point clouds is computationally expensive.

Graph-based representations employ a non-Euclidean data structure to model point cloud data with nodes and edges, where nodes represent points and edges denote relationships between neighboring points [33]. Unlike many point-based methods, that process points independently, graph-based data representations encode the local neighborhood structure by linking points through the connecting edges that capture geometric relationships. This enables more robust feature extraction and streamlines the 3D data processing [31].

Projection-based methods differ from others by transforming the 3D point cloud data into one or more 2D image-like representations, such as Front View (FV) or Bird's Eye View (BEV) [31]. FV is a perspective view which has the depth data in Z-axis compressed, and the BEV is a top-down perspective that has the data in Y-axis compressed. In this review [33], Li refers these as View-based methods, mentioning that these utilize traditional well-established convolutional neural networks (CNN) to process the projected 2D views, which can yield near-optimal results compared to other 3D data representations. The efficiency of using established 2D CNN architectures and datasets is a significant advantage in efficiency and practicality. However, this transformation from 3D to 2D projections can lead to the loss

of geometrically-related spatial information, and the redundancy of multiple views increases computational complexity.

2.3.3 Feature Extractions

After the input 3D point cloud data is processed into the desired representation format, the next step is to identify and extract important features. The feature extraction method employed depends on the data representation format, as each format requires a specific type of input data. These can also be categorized, and the summaries are displayed in the Table 2.2.

Segment-wise feature extraction methods, such as those used in voxel- and pillar-based approaches, operate on segmented data. These methods iterate through the non-empty voxel cells to perform feature extractions, dividing points into grids with a fixed number of points and subsequently applying point-wise classification to extract features [34]. The Voxel Feature Encoding (VFE) layer from VoxelNet is an example that enables inter-point interactions within a voxel, where point-wise features are combined with locally aggregated features [35]. This strategy is both memory- and computationally efficient way to extract local and global features from voxel grids [31]. Two examples of point cloud object detection methods that use segment-wise feature extractions are VoxelNeXt [36] and PointPillar [37]. VoxelNeXt is post-processing-free and efficient 3D object detector designed to function without anchor proxies, sparse-to-dense point conversions, or other complex components. It employs spatial voxel pruning to select voxels with high feature magnitudes and the applies sparse max pooling layer to extract features. PointPillar, on the other hand, is a fast and versatile 3D object detector that leverages 2D convolutional layers. In its Data Representation phase the features are converted into the pseudo-image format and then extracted via upsampling and concatenation.

To preserve the original structure of point clouds, point-wise feature extractors

Table 2.2: Summary of Feature Extraction methods.

Method	Main Ideas	Pros	Cons
Segment-wise	Extract features within each cell (voxels, pillars), aggregate local features	Efficient for sparse 3D data, local and global features, can leverage 2D and 3D CNNs	Loses fine details with large segments, relies on discretization
Point-wise	Process individual 3D points directly, utilize MLPs and pooling for global features	Preservers original structure, flexible for various point densities	High computational overhead, scalability issues, complex grouping/pooling strategies
Graph-wise	Use graph convolutional networks to propagate and pool features along edges	Encode local neighborhoods effectively, preserves geometry, robust features	High implementation complexity, can be resource-intensive
Object-wise	Use 2D FCNs or other CNN on 2D projections (pseudo-images)	Advanced 2D CNN, efficient on GPUs, easy multi-sensor fusion with cameras	Loss of 3D spatial info, prone to projection distortions

are employed. These methods directly process raw point clouds to extract global features useful for subsequent classification and regression tasks [31]. One of the most notable point-wise architecture is PointNet [38], which does not require any modifications in Data Representation phase, since it takes point clouds as input and outputs either object labels for the point cloud as a whole, or part labels for individual points. Its feature extraction primarily relies on max pooling, which acts as a symmetric function to aggregate information from all points. PointRCNN [39], a two-stage 3D object detection framework, performs feature extraction by segmenting the point could into foreground points and simultaneously generating a small number of bounding box proposals. A region pooling operation then transforms the learned point representations into canonical coordinates, which are combined with the pooled point features. PointNeXt [40] is a relatively new architecture derived from PointNet++. The updated version is designed to be more scalable through two main modifications: adding a multilayer perceptron (MLP) at the beginning of

the network, and introducing an Inverted Residual MLP (InvResMLP) block after self-attention block in each stage. Otherwise, its architecture — and therefore its feature extraction phase — remains the same as in PointNet++ [41]. It works by partitioning the point cloud into overlapping local regions based on the distance metric of the space, and then local features from these regions are iteratively combined into larger units and processed to output higher-level features.

Graph convolutional neural networks (GCNs, also known as Graph-CNNs) are employed for graph-based representations of point cloud data [31]. While they share architectural similarities with traditional CNNs, GCNs are specifically designed to handle data structured as graphs. These can be either naturally sampled on the vertices or edges of a graph, like transportation networks flows, or graphs imposed over unstructured data to capture underlying structure, like 3D point clouds. Architectures such as PointGCN [42] and Graph R-CNN [43] leverage these graph representations for feature extraction in 3D object detection, rather than utilizing voxel grids, uniform grid points, or keypoints. In particular, Graph R-CNN combines 3D point clouds with 2D visual features by dynamical aggregation, forming detailed graph inputs. With Region of Interest (RoI) and iterative graph refinement, it fuses geometric and visual details to produce robust object detection features. On the other hand, PointGCN utilize graph convolutions that are performed directly over local neighborhoods containing geometric relationships. These graph convolutions propagate and aggregate feature information among the connected nodes, preserving fine-grained structure.

Object-wise feature extraction leverages 2D fully convolutional networks (FCNs) to process 2D projections obtained from projection-based data representation methods [31]. These FCNs can utilize one or more of the 2D projections as inputs and apply point-wise classifications on the resulting synthetic images, also known as pseudo-images [32]. Feature extraction is then performed on these pseudo-images to

produce height maps, density, intensity, and/or distance features, depending on the input sensors. Architectures such as MV3D [44] and MaskBEV [6] utilize pseudo-images for 3D object detection. Multi-View 3D (MV3D) uses lidar point clouds to form bird's eye view (BEV) and front view (FV) pseudo-images, as well as an RGB image from a regular camera. Each representation is processed by convolutional layers, but the BEV representation specifically generates 3D object proposals, which are then projected into three views: BEV, FV, and the camera image plane. These views are combined using region-wise features obtained through ROI pooling on each pseudo-image. This outputs fused features that are subsequently used for object class prediction and 3D bounding box regression. MaskBEV, on the other hand, adopts a simpler architecture by using only the BEV representation generated from point cloud obtained with lidar. It employs a Swin Transformer (Swin-T) [45] to extract multi-scale features from the BEV. The BEV pseudo-image is divided into non-overlapping patches, which are processed by Swin Transformer blocks that are based on 'shifted windows' technique. This produces feature maps that are later on upsampled and used for object class and mask predictions.

2.3.4 Detection Network

The final stage of 3D object detection is the detection network, which utilizes the outputs from the feature extraction phase to generate predictions about the objects' classes and locations. The architecture can be either one-stage or two-stage, with the key difference being in the usage of regions of interest (ROI) from feature maps [31]. Two-stage detectors offer a more versatile approach, compatible with a broader range of architectures, and typically provide better accuracy on object proposals. Conversely, one-stage detectors have faster processing speeds due to their more compact structure. Although they have quicker inference times, they face balance issues because they operate directly on the features without the region proposals. In

both cases, the end products of these detectors are object predictions — specifying the class, size and location of the detected objects — used in various applications such as path planning and object tracking.

As the name suggests, two-stage detectors operate in two phases: region proposal generation and ROI pooling. For example, Fast R-CNN [46], a refined version of R-CNN, employs an ROI pooling layer which first extracts a feature vector from each feature map channel. These vectors are then taken through a series of fully connected layers that produce softmax [47] probabilities for object classification and four coordinates for each object class, representing the bounding-box positions. A more recent architecture Sparse Fuse Dense (SFD) [48], introduces a new ROI fusion strategy known as 3D Grid-wise Attentive Fusion (3D-GAF), along with Synchronized Augmenetation (SynAugment) for data augmentation tailored for multi-modal framework, and customized Color Point Convolution (CPConv) for extracting features from pseudo point clouds. This architecture, which integrates data from RGB cameras and lidar, outputs class and bounding box predictions from the fused ROI features during its detection network phase. It also produces separate auxiliary heads from each input stream to regularize the network. As of its 2022 publication, SFD ranked first on the KITTI car 3D object detection leaderboard [49]. By 2025, however, it has fallen to the 21st position, with VirConv-S [50] now leading.

One-stage, also known as single-stage, detectors provide a simpler and faster alternative, designed to minimize inference time without using ROI feature maps. These detectors can also utilize data from multiple sensors and various data representation methods. For instance, FCOS-lidar [51] capitalizes on the range view (RV) of lidar points from projection-based methods, and 3DSSD [52] utilizes a raw point cloud, a result from point-based data representation methods. Both of them utilize only lidar data, but the range view used in FCOS-lidar resembles common RGB image, which minimizes the usage of 3D data processing. It does not need

complicated and expensive voxelization processes, or sparse convolutions, because of the compact range view representation. FCOS-lidar, which employs a feature pyramid network (FPN) for processing spatially downsampled features, attaches classification and regression heads to the feature levels. Each head includes four 3×3 convolutional layers with 64 channels and a prediction convolutional layer. The classification heads generate softmax scores for each class prediction, and the regression heads produce final 3D box prediction, which include coordinates, yaw angle, velocity vector and intersection-over-union (IoU) with the ground-truth. This architecture offers a simple and efficient range view alternative to BEV-based 3D detectors in the projection-based method category. Meanwhile, 3DSSD developed with speed and accuracy in mind, notably manages to remove computationally expensive feature propagation (FP) layers and refinement module, previously considered irreplaceable in point-based methods. After its data representation and feature extraction phase, it has processed the raw point cloud into candidate points that are passed through multilayer perceptron (MLP) to an anchor-free regression head for classification and box prediction of the object. The predictions from 3DSSD's detection network include distance (x, y, z), size (l, w, h) and orientation. Its main advantages being performance and reduced inference times achieved by removing feature propagation layers and refinement modules, and employing an anchor-free regression head.

2.4 Bird's Eye View

The top-down representation that Bird's Eye View (BEV) produces simplifies perception tasks, enhances situational awareness, and supports robust navigation of autonomous vehicles. However this comes with computational overhead, and requires specialized methods to maintain efficiency and accuracy.

Understanding the surrounding environment is crucial for autonomous vehi-

cles, achieved primarily by creating a 3D geometric reconstruction of the physical world [7]. With the increasing variety of sensors equipped on robots and self-driving vehicles, creating an unified presentation of multi-source information becomes critical. The Bird's Eye View is one suggested concept, integrating environmental information from multiple sources into a self-referenced top-down view of the vehicle [1], as illustrated in Figure 2.6. While a single elevated sensor could also achieve this aerial perspective without any data processing, such approach is impractical in environments with spatial constraints, like indoors.

BEV perception can be achieved with many ways, but in autonomous driving research, it can be segmented into three main areas: BEV Camera, BEV lidar and BEV Fusion [7]. This research focuses on the utilization of point cloud data from lidar sensors.

BEV Camera uses multiple cameras with geometric transformations to create BEVs, which are used for object detection and/or segmentation. The features extracted from these views are processed using algorithms such as BEVDepth, BEVStereo, and CaDDN [53]–[55] to generate accurate outputs without distortions [56].

BEV lidar leverages point cloud data instead of regular images to generate Bird's Eye Views, as illustrated in Figure 2.7. The lidar sensor's output can then be

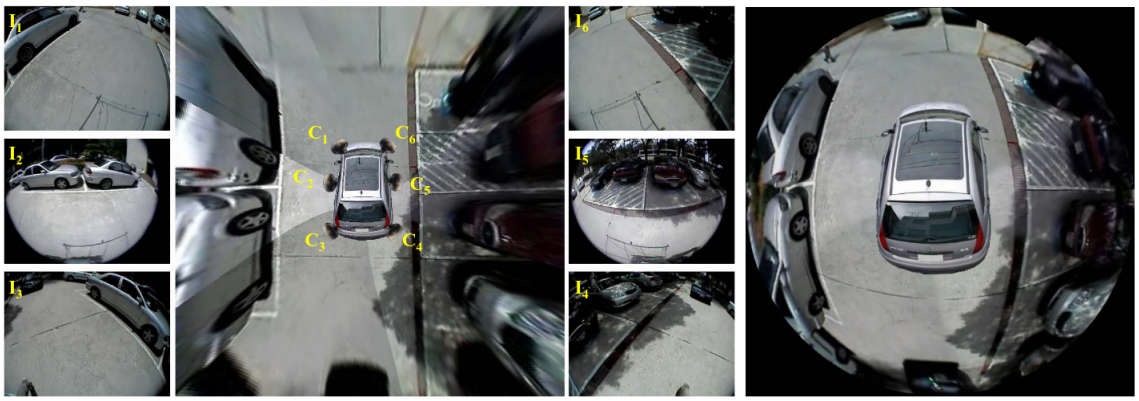

Figure 2.6: Bird's Eye View representation achieved with images from six independent fisheye cameras. [1] *Copyright © 2008, Springer-Verlag Berlin Heidelberg*

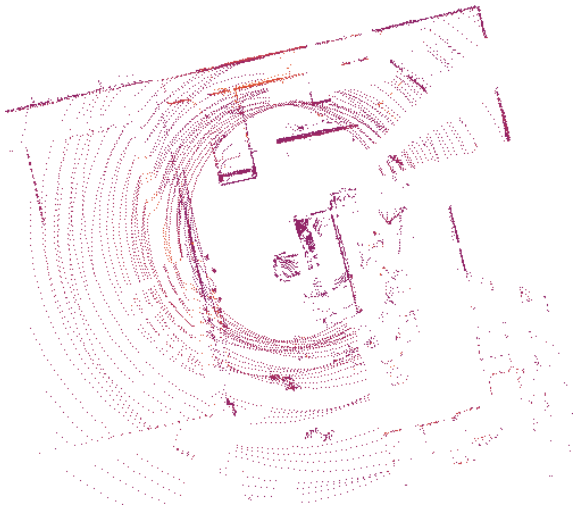

Figure 2.7: Bird's Eye View representation from our proposed lidar-only dataset.

transformed to BEV representation using either pre-BEV or post-BEV methods [7]. The former preserves more 3D information and is suitable for applications where computational resources are not constrained, while the latter is more efficient but provides less 3D detail, making it well-suited for 2D focused tasks. Techniques such as VoxelNet, PointPillars, and Pointformer are commonly used in BEV lidar implementations [35], [37], [57]. A key technique in these methods is voxelization, where the 3D space around the vehicle is divided into a grid of small cells called voxels. These voxels retain point-level features (e.g. coordinates) but are typically collapsed or compressed along the vertical axis to produce the top-down 2D pseudo-image referred as BEV.

BEV Fusion combines data from these and potentially other sensor fusions, enhancing perception capabilities and environmental understanding. The Camera-lidar fusion is most researched option and it merges color and texture data from images with distance and height information from point clouds [56]. Some related algorithms within the BEV Fusion are e.g. TransFusion, BEVFusion, and Fast-BEV [58]–[60]. Fusion-based BEV models generally outperform camera- and lidar-only methods due to their robust, multi-modal approach [61].

When the majority of relevant objects are located on an approximately horizontal plane around the vehicle, top-down orthographic projection via BEV provides standardized format for perception tasks. This format includes both locations and semantic features while remaining unaffected by occlusion, making it promising for various 3D perception tasks in autonomous driving [62]. The standardized BEV projection format also facilitates research and development in 3D perception, offering a consistent foundation for different approaches and methodologies. BEV enhances detection accuracy and robustness in 3D detectors by providing detailed spatial and semantic information [61]. Overall situational awareness is improved through the 360-degrees perception that BEV enables.

Despite these advantages, BEV methods face notable challenges. BEV Camera methods generally lag behind BEV lidar in both effectiveness and efficiency, particularly when it comes to accurately inferring 3D locations from 2D information [63]. Transforming input data into BEV format prior to tasks like object detection adds an additional computational step, which often increases overall processing complexity, requiring more computational resources [56]. Additionally, BEV models have tendency to be more vulnerable to adversarial noise that can be in image data, which can lead to increased errors under suboptimal image conditions [61].

2.5 Datasets in Autonomous Driving

Datasets, which are large collections of specific information, have become increasingly relevant in today's data-driven era. Although data can be gathered from any type of sensor, the most widely used datasets for 3D object detection in autonomous driving primarily focus on cameras, lidars and RADARs. The most commonly used, publicly available datasets in this context are summarized in Table 2.3, including KITTI [64], KITTI-360 [9], nuScenes [65], Waymo Open [66], and ONCE [67]. They differ from one another in various ways, and this section compares them with regard

Table 2.3: Most relevant dataset for autonomous driving and 3D object detection. Class values in parentheses mean the suitable amount of classes for object detection benchmarks.

Dataset name	**Published**	**Frames**	**Box Annotations**	**Classes**	**Night / Rain**
KITTI [64], [68]	2012	15K	200K	8(3)	No/No
nuScenes [65], [69]	2019	40K	1.4M	23(10)	Yes/Yes
Waymo Open [66], [70]	2020	230K	12M	4(3)	Yes/Yes
ONCE [67], [71]	2021	1M	417K	5(5)	Yes/Yes
KITTI-360 [9], [72]	2022	83K	68K	19(2)	No/No

to their size, diversity, and advantages/disadvantages. Each dataset also maintains its own benchmarks and leaderboards, providing a straightforward way to evaluate and rank top-performing machine learning architectures. Often, these benchmarks feature multiple sub-challenges, such as detection, tracking, prediction, and segmentation.

KITTI [64], [68], the oldest and most frequently cited among these datasets, revolutionized research and development in self-driving vehicles. Before its release, visual recognition systems were rarely used in robotic applications such as autonomous driving. The KITTI data were collected using a car equipped with four PointGrey Flea2 video cameras (two grayscale and two RGB) and a Velodyne HDL-64E lidar mounted on the roof. This setup produced approximately 15 000 frames over a total driving distance of 39.2 km in sunny daytime conditions. From these, more than 200 000 3D object annotations were generated across eight class labels; however, only three (Car, Pedestrian, and Cyclist) were ultimately considered suitable for online evaluation. Although KITTI is relatively small and less diverse compared to newer datasets, it remains historically significant for pioneering autonomous driving research. As of February 2025, the top-ranked 3D object detection model on the KITTI leaderboard is VirConv-S [50], which was published in March 2023.

Because KITTI is now over a decade old, a successor dataset called KITTI-

360 [9], [72] was introduced in 2023, with one of the KITTI's original authors of leading its development. The main technical difference of KITTI-360 is that it provides full 360° surround vision in every scene. This was achieved by collecting data with a front-facing 90° stereo camera, a 180° fisheye camera on both sides, and Velodyne HDL-64E and SICK LMS 200 lidars mounted on the roof. The result is a dataset of 332 000 frames consisting four sets of 83 000 2D images and 83 000 3D laser scans, covering a driving distance of 73.3 km, which is almost double the distance of original KITTI. All geolocated 3D and 2D scene elements, both static and dynamic, were annotated using a custom WebGL-based annotation tool. The annotations provide semantic and instance labels for each 2D image pixel and 3D point. For comparison, KITTI-360 contains about 68 000 3D bounding-box annotations, roughly one-third of the original KITTI's count. As a relatively new dataset, not all architectures have been benchmarked on KITTI-360 yet; however, PBEV+SeaBird [73], published in the May 2024, currently leads the 3D object detection leaderboard.

NuScenes [65], [69] is a diverse and extensive dataset known for offering the largest number of object classes among its competitors. It was compiled from 15 manually selected hours of driving data in Boston and Singapore, totaling 242 kilometers in varying traffic scenarios and weather conditions. There were two identical cars, each equipped with six RGB cameras (five 70° FOV in front/sides, and one 110° FOV in the rear), one spinning 360° lidar on the roof, and five RADARs (one in each corner and one in the front). These sensors captured around 1.4 million camera images and 390 000 lidar scans in daylight, nighttime, rain and snow. In total, there are 1.4 million 3D bounding-box annotations spanning 23 object classes, which was the largest collection at that time. These classes go into more detail than other datasets; for example, they include eight variants of pedestrians, ten variants of vehicles, and four types of movable objects. Some of these classes have special

attributes like moving and parked. However, for 3D object detection benchmarking, only ten of these classes are considered compatible. As of February 2025, the leading architecture on nuScenes 3D object detection leaderboard is MV2DFusion-e [74], published in August 2024.

Waymo [66], [70] subsidiary of Alphabet (Google) focused on self-driving cars. In 2020, they introduced a large-scale, high-quality, and diverse dataset called Waymo Open. In most aspects, it surpasses earlier datasets in terms of scale, with the exception of offering only four class labels, three of which are eligible for 3D object detection benchmarking. By contrast, nuScenes — published in the previous year — features 23 classes in total, ten of which are 3D object detection eligible. Waymo's data collection was performed using a car equipped with five lidar sensors and five high-resolution pinhole cameras mounted on the roof, covering 360° around the vehicle. The main lidar on the roof was restricted to 75 meters in range, while the remaining lidars (three in front and one in back) were limited to 20 meters to optimize data density. Altogether, the dataset contains around 12 million labeled 3D lidar objects and 12 million labeled 2D image objects, captured both in suburban and urban areas at four different times of day. The coverage area includes 40 km^2 in Phoenix, and 36 km^2 in Mountain View and San Francisco combined, for a total of 76 km^2. These include sunny and rainy weather conditions, but no snow. Thanks to its large-scale, multimodal nature, the Waymo Open dataset often requires less data augmentation from algorithms to achieve better ML models. While there is no dedicated 3D object detection leaderboard, Waymo does feature a Real-time 3D Detection competition, and algorithm called DetZero [75], published 2023, holds currently the first place.

In addition to the prominent datasets above, many other datasets have been introduced by various companies and research institutions but are not cited as frequently. One example is ONCE [67], [71] dataset, developed by Huawei as a note-

worthy large-scale alternative. Huawei concluded that modern perception models in autonomous driving has become heavily reliant on vast amounts of data. Thus, they created a dataset containing 1 million lidar scenes and 7 million corresponding camera images. These were gathered from multiple undisclosed cities in China under rainy, cloudy and sunny weather conditions. This took around 144 hours of driving and it covered around 210 km^2, giving it 20 times more driving hours and almost three times more covered area than Waymo Open dataset. The sensor configuration includes a 40-beam 360° lidar sensor and seven cameras that also cover 360° horizontal FOV around the vehicle. The captured images and point clouds were manually labelled and double-checked, resulting in 417 000 3D bounding-box annotations across five different eligible classes for 3D object detection.

Although these datasets differ from each another, they share a common objective: to provide a standardized and validated foundation for research and development in autonomous driving. Autonomous driving is a complex field with various subtasks, and different datasets are suited for specific scenarios. Among the discussed datasets, nuScenes offers the most diverse set of object classes, Waymo Open contains the largest number of 3D bounding-box annotations, and ONCE is distinguished by its large raw size and total number of frames. In contrast, KITTI and KITTI-360 are the only datasets that do not include data collected under challenging weather conditions, such as rain or nighttime, and are comparatively limited in the metrics mentioned above. However, these datasets offer different strengths, depending on the evaluation criteria, and specific datasets might be more suitable for certain applications. KITTI, for example, is often used in foundational studies due to its pioneering role in this field.

3 Design Overview

This chapter provides a high-level perspective on both the hardware and software elements that comprise the indoor bird's eye view perception strategy. The main goal is to define and describe the physical platform and the software components without delving into implementation details, which will follow in Chapter 4.

3.1 Overall System Architecture

This research addresses the perception strategy of indoor autonomous driving, which presents unique challenges due to complex layouts, dynamic obstacles, and limited GNSS availability. Compared to an outdoor setting, indoor perception has received far less attention from researchers, and this research aims to help fill that gap. A high-level diagram of our proposed system is illustrated in Figure 3.1.

Bird's Eye View was chosen because it offers several advantages for indoor perception, where distances between objects are shorter and general localization technologies, such as GNSS, may be blocked. BEV provides a consistent representation of the vehicle's pose at all times and also acts as a stable frame of reference for simultaneous localization and mapping (SLAM).

We propose a modified version of existing algorithm MaskBEV [6] that is adapted for an indoor setting. We also create a customized KITTI-like dataset featuring indoor object classes.

The data used in this research was gathered with manually operated robot

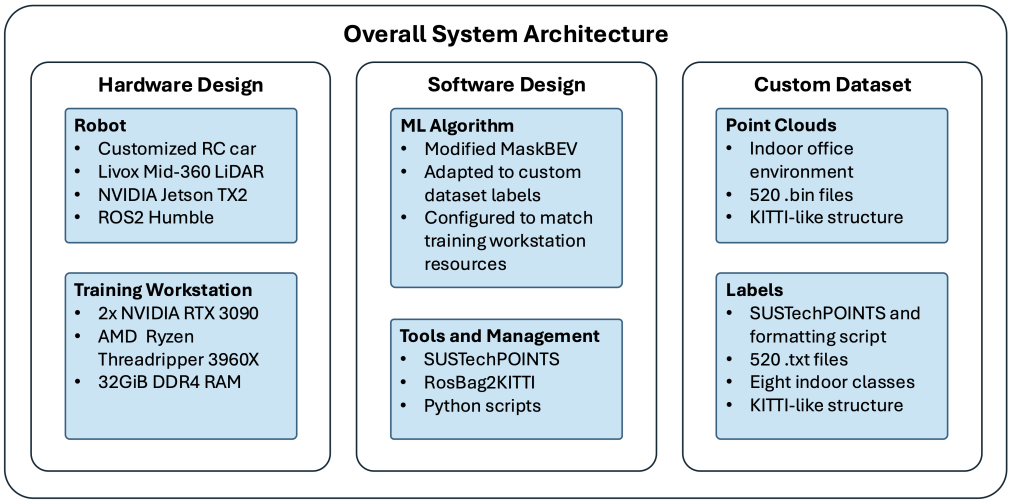

Figure 3.1: High-level overview of the whole system used in this research.

equipped with a single-board computer, a lidar sensor, and four RGB cameras. We plan to use the same robot in future research related to path planning and practical autonomous driving.

The overall architecture consists of physical components (the hardware), diverse set of computer programs (the software), and the created custom dataset. The hardware includes the robot and the workstation employed for training the machine learning model, while the software comprises the robot's software, multiple supporting tools and scripts, and the machine learning algorithm.

3.2 Hardware Design

Although this thesis focuses on software-based perception strategies, practical data collection and experimentation relied on an existing hardware platform. The hardware includes a small ground vehicle with a single-board computer and sensors, mostly designed and developed by previous thesis on this topic. These details are briefly summarized here to provide context for how our software solutions integrate with the hardware.

3.2.1 Robot Hardware

The core of this robot platform is a typical small-scale RC car chassis with front-wheel steering and an electric motor powered by a lithium battery. On top of this chassis, we have an onboard inference computer, multiple sensors, and a separate power system for these components.

The original RC car underwent extensive modifications to accommodate the necessary computational resources. These modifications include 3D-printed housings for the main battery and metallic risers that create additional "floors" on top of the chassis, offering mounting space for the computer and sensors. The battery pack is Gens ace Lipo with a capacity of 5000 mAh and voltage is 14.8 V, which equal to 74 Wh. These adaptations ensure that the robot has sufficient power to drive the motor, steering mechanisms, onboard systems, and sensors, making it both mobile and self-contained.

The robot incorporates several sensors to facilitate perception and navigation. Central to this setup is a Livox Mid-360 [76], a hybrid solid-state lidar providing a full 360° horizontal field of view (FOV) and a 59° vertical FOV. This lidar uses Livox's rotating mirror hybrid-solid technology to achieve omnidirectional coverage, capturing point clouds with a 40-line density. In addition, four Luxonis OAK-D Lite cameras are mounted around the robot. Although these cameras are not utilized in the current study, they could be valuable for future integration of RGB or depth data. Furthermore, the robot includes an MPU-6050 inertial measurement unit (IMU), providing both accelerometer and gyroscope measurements for motion tracking.

To run inference or process sensor data in real-time (or near real-time), the robot requires its own processing unit; our platform uses an NVIDIA Jetson TX2 embedded AI computing device with software based on the ROS2 Humble platform. This relatively small computer features an NVIDIA Pascal architecture GPU

with 256 CUDA cores, a dual-core NVIDIA Denver 2 64-bit CPU, and a quad-core Arm Cortex-A57 MPCore processor. It only consumes around 7.5 watts of power while handling the inference from all of the sensors mentioned above. Livox-SDK, and oakd-ros2-multicam serve as dependencies enabling communication between the sensors and the ROS2.

All hardware components — RC chassis, sensors, and onboard computer — were developed by previous work. Hence, the descriptions in this section are provided solely for context. The contributions of this thesis focus on the software side of indoor lidar perception, for which the existing platform is used to collect data, run experiments, and validate the proposed methodologies.

3.2.2 Training Workstation

The machine learning model must be trained with appropriate data to perform the desired task. This training is done by running a program on a workstation with sufficient computational resources such as memory and processing power. The main specifications of our workstation are listed here:

- **GPUs:** Two NVIDIA GeForce RTX 3090 cards, each with 24 GiB of G6X memory (total 48 GiB VRAM). Driver version: 535.183 and CUDA version: 12.2.
- **CPU:** AMD Ryzen Threadripper 3960X 24-Core Processor with a base clock of 3.8 Ghz.
- **RAM:** Six Crucial 32 GiB DDR4-3200 UDIMM modules (CT32G4DFD832A), totaling 128 GiB of system memory.
- **OS:** Ubuntu 20.04.6 LTS (Ubuntu focal).

This configuration was sufficient for training our relatively small indoor dataset. However, when we attempted to train model on the original KITTI dataset, we

encountered out-of-memory issues, primarily due to the significantly larger data volume. For future scalability, this workstation would benefit from additional resources, particularly data-center-grade GPUs, for example A100 or H100 Tensor Core GPUs. These cards are specifically designed for data-center environments, whereas RTX cards mainly target gaming and rendering tasks.

3.3 Software Design

This section presents a broad overview of the tools and applications employed in our approach. Deeper implementation details are covered in Chapter 4.

3.3.1 High-Level Algorithm Overview

We base our algorithm on MaskBEV, a mask-based object detector neural architecture that uses BEV data representations. Unlike most object detection algorithms, which predict bounding boxes, MaskBEV generates BEV instance masks that capture the footprint of each object. It employs transformer networks to process global information via cross-attention and self-attention across the entire BEV.

Figure 3.2 illustrates the high-level dataflow of the MaskBEV as well as our own version. The architectural pipeline begins with lidar point cloud data, which is transformed into BEV pseudo-images by an encoder. In essence, this encoder performs sparse voxelization, augments each voxel with voxel-relative information, concatenates the returned laser strength, and applies multilayer and layer normalization steps. These operations result in an image of (F $\times$ H $\times$ W), where F is the embedding dimensions, H is the number of voxels along the y-axis, and W is the number of voxels along the x-axis. After encoding, the pseudo-image is processed by a system based on Mask2Former [77]. First, multi-scale feature extraction is performed using the Swin-T [45] backbone, producing feature maps of different sizes.

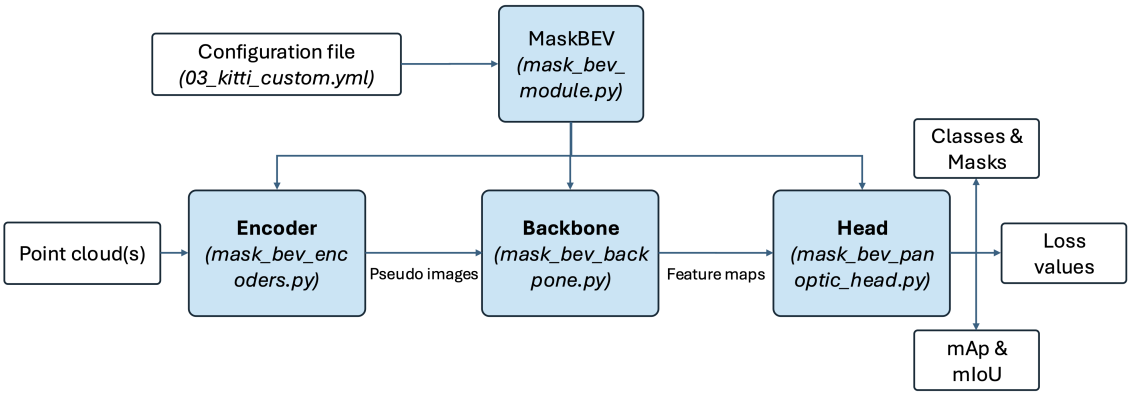

Figure 3.2: Pipeline of MaskBEV based model.

These feature maps are then upsampled by point decoders that produce embeddings for each potential mask detection. In parallel, each feature map is also passed into a transformer decoder that generates query embeddings, which combine with the mask embeddings to produce the binary masks of detected objects.

The main difference between MaskBEV and our implementation lies in the shift from outdoor to indoor environments. Consequently, our object classes differ, focusing on people and indoor structures. To accommodate smaller indoor robots, we introduce a several modifications to MaskBEV to ensure proper functionality, as detailed in Section 4.3.

3.3.2 Tools and Management

This research relies multiple different tools and scripts, ranging from data labeling to version control. Some of these tools are open-source projects used as-is, while others required modifications or had to be developed ourselves from scratch.

Data collection was performed using ROS2 and a deserializing tool called rosbag2_to_pcd [78] to control the robot and capture the 3D point clouds. Then SUSTechPOINTS [79] was used to label the frames, and Python scripts — a custom

one and RosBag2KITTI [80] — were for conversion of label and point cloud files to match KITTI's format.

SUSTechPOINTS was originally developed for labeling KITTI-type datasets, so the source code, written in Python, needed changes to accommodate our custom indoor object labels.

Because this research involves coding, Git and GitHub were used for version control. The project also employed remote access tools such as ZeroTier and AnyDesk. Development of this research took place primarily in Docker containers to facilitate possible future reuse.

3.3.3 Robot's Software

As the hardware of the robot came from contributions of the previous work, so is the robot's software. The onboard computer runs Ubuntu 18.04 operating system, the ROS2 Humble platform, the NVIDIA Cuda toolkit, and sensor drivers. The lidar uses livox_ros_driver2 from Livox-SDK, the cameras require oakd_ros2_multicam, and the IMU relies on the mpu6050 Python package.

All of the robot's software is bundled in a Docker container, making it self-contained and easily transportable.

3.4 Custom Dataset

The pioneering KITTI dataset inspired much of this research, and thus the proposed custom dataset inherits the folder structure and file formats from KITTI. The main difference lies in the customized labels and the contents of the frames. In KITTI [64], there are eight object classes: Car, Van, Truck, Pedestrian, Person sitting, Cyclist, Tram, and Misc. These had to be changed to match our requirements in an indoor setting. We replaced these with classes for most typical indoor elements, resulting

in these eight classes: Person, Table, Chair, Couch, Shelf, ThrashCan, Robot, and Misc.

The finalized dataset, containing point clouds and labels, is only around 166 megabytes in size, whereas the KITTI dataset is almost 150 times larger in size, approximately 24 600 megabytes [68]. The overall structure of the custom dataset is visualized in Figure 3.3, showing three main folders for calibration, label, and point cloud files, each split into testing and training directories. This structure largely mirrors KITTI; we omit the image_02/ directory containing RGB images in .png format, because only lidar point clouds are used. In our case, the testing directories remain empty, and the train/test files split is specified by train.txt (80 %) and val.txt (20 %), which we generate via a script. We also generate samples.pkl, which stores

```
Custom_Dataset/
|
├── data_object_calib/          # Calibration files, one for all, or separate files for each frame
|   └── testing/
|   └── training/
|       └── calib/
|           └── lidar_calib.txt
|           └── dummy_calib.txt
├── data_object_label_2/        # Label files, one file for one frame, one row for one object
|   └── testing/
|   └── training/
|       └── label_2/            # [000001.txt-000520.txt]
|           └── 000001.txt
|           └── 000002.txt
|           └── ...
├── data_object_velodyne/       # Point cloud files, one file for one frame
|   └── testing/
|   └── training/
|       └── velodyne/           # [000001.bin-000520.bin]
|           └── 000001.bin
|           └── 000002.bin
|           └── ...
|                               # Generated files with scripts:
├── train.txt                   # List of point cloud files (.bin) that are used in training
├── val.txt                     # List of point cloud files (.bin) that are used in validation/testing
├── samples.pkl                 # Pre-generated samples for data augmentation
```

Figure 3.3: File structure for our custom datasets. Distinct folders for calibration, label, and point cloud files, and each of these directories are divided into testing and training files. It deviates from the KITTI dataset format with a distinct calibration file for all frames rather than separate file for each frame, and KITTI also includes folder for RGB images.

information about objects to undergo data augmentation during model training, reducing runtime overhead.

The label files in our custom dataset, and also in the KITTI dataset, are text files with a .txt extension, meaning they contain readable ASCII characters. Each label file represent one frame and may contain multiple labels, each occupying a separate line. One object label is defined by 15 space-separated values, starting with the label type (a string) and followed by integers or floats. Figure 3.4 presents an example file from our dataset (00510.txt), which includes four objects: two persons, a couch and a chair. The first seven numbers after type declaration relate to 2D image data. As we only use lidar data, these 2D fields are set to zero. The next three numbers represent the 3D object dimensions in meters (height, width, and length) and the subsequent three values describe the object location in camera coordinates, also in meters. The final number specifies the rotation around the Y-axis, ranging from -π to π (-3.14159–3.14159). Table 3.1 lists these values in detail. In terms of memory size, the label files occupy only a small portion of the dataset, ranging from about 149 bytes to 981 bytes each.

In contrast, point cloud files are stored in binary format .bin, making them unreadable by text editors. The data inside is made up of bit patterns, and does not contain anything in readable text format. This file format is more efficient for machines to understand, and this is important in point cloud files, since they can potentially contain thousands or even millions of 3D points. As with the label files, each frame corresponds to an individual binary file. In our dataset, the smallest

```
Person 0.00 0 0.00 0.00 0.00 0.00 0.00 1.85 0.81 0.68 2.68 0.37 -2.19 -2.58
Person 0.00 0 0.00 0.00 0.00 0.00 0.00 1.85 0.81 0.68 -0.62 0.28 -0.19 -0.18
Couch 0.00 0 0.00 0.00 0.00 0.00 0.00 1.17 1.24 2.23 2.01 0.26 -0.25 -3.20
Chair 0.00 0 0.00 0.00 0.00 0.00 0.00 0.94 0.73 0.67 0.84 0.36 -3.19 -1.70
```

Figure 3.4: Actual label file, 00510.txt, from the custom dataset, containing four objects. The definitions of these values is described in the Table 3.1.

Table 3.1: Description of the 15 values in each row of the KITTI .txt label files. Modified from KITTI dataset readme.txt file.

Values	**Name**	**Description**
1	type	Describes the type of object: 'Person', 'Table', 'Chair', 'Couch', 'Shelf', 'TrashCan', 'Robot', or 'Misc'.
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries.
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded 2 = largely occluded, 3 = unknown.
1	alpha	Observation angle of object, ranging [-pi..pi].
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates.
3	dimensions	3D object dimensions: height, width, length (in meters).
3	location	3D object location x,y,z in camera coordinates (in meters).
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi].
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

point cloud file is about 315 kilobytes and the largest 326 kilobytes, making them substantially larger than the label files. Because we do not use camera images, these point cloud files represent the bulk of our dataset's memory footprint.

The third folder, for calibration files, is also text-based but contains minimal data. The purpose of these calibration files is to synchronize camera images with point clouds, facilitating labeling simultaneously in both the image and point cloud domains. However, our custom dataset only uses point cloud data, which renders extensive calibration data unnecessary. In the original KITTI dataset, there would be individual calibration files per frame and per camera. We simplified this by creating one dummy_calib.txt file that assigns the location of the lidar relative to the robot and also assigns the camera values as zeroes, making it not raise errors with codes that require calibration files. While hacky workaround, this approach was mandatory to ensure compatibility with the rest of our pipeline.

All these elements combine to form a dataset suited for training machine learning models on indoor environments and objects. The 520 frames is relatively small

amount of frames, and will probably only serve as a prototype. For real-world usages, machine learning models require more data applicable for training to produce reliable results.

4 Implementation

In contrast to the design-oriented text of Chapter 3, this chapter provides the concrete steps and technical details involved in constructing the system. It outlines the processes required to build, train, and deploy our BEV perception system.

4.1 Data Collection

Data collection was carried out using the previously described RC car, which was equipped with a Livox Mid-360 hybrid solid-state lidar sensor. The RC car was manually controlled via ROS2 and driven through a kitchen and lounge area in the Turku Intelligent Embedded and Robotic Systems (TIERS) lab's research space in Turku, Finland. These areas contained a diverse set of objects, including chairs, couches, tables, and people. Most of the objects were static, but the people in the environment moved during the data collection.

While the car was controlled around the indoor office environment, the lidar continuously published 3D point cloud data as ROS2 messages to the onboard inference computer. This data stream was captured in real-time and stored in a rosbag file at a rate of 10 Hz. Then a utility tool called rosbag2_to_pcd was used to read and deserialize the data into a PCL format, which produced 520 point cloud frames as .pcd (Point Cloud Data) files. Each of these point cloud files contained data points covering the full 360° field of view around the robot, with 40 laser scan lines ensuring reasonable vertical point cloud density. The results of this procedure is a collection

of 520 raw point cloud files. But these files must to be processed before they can be used for other tasks, such as machine learning model training.

4.2 Point Cloud Processing

After the initial data collection phase, the raw 520 frames of point cloud data required additional processing to become compatible with the machine learning algorithm. The point clouds had to be labeled with object classes and converted from .pcd to .bin format. Similarly, the created label files needed to be converted from .json to .txt. Also, a calibration file had to be constructed with dummy data to finalize the dataset. These steps were mandatory because MaskBEV is configured for datasets like KITTI, KITTI-360, Waymo Open, and SemanticKITTI, and we decided to closely follow KITTI's dataset structure.

The most critical aspect of custom dataset creation involved manually labeling the objects in each point cloud frame. Here, labeling means the process of selecting sets of points and assigning class labels for them. We utilized an open-source software called SUSTechPoints [79] for this task, which provided a semi-automated tool that accelerated the tedious labeling process. Figure 4.1 shows the tool in use, showcasing the process of labeling a Person object in our custom dataset. We iterated through all 520 frames, labeling every object in the scenes with our indoor classes (see Subsection 5.1). We also modified the tool's source code to accommodate our custom labels. Figure 4.2 showcases few included features of the tool, batch-mode editing and interpolation, which proved to be helpful, although the auto-labeling feature produced incorrect volumes in our tests and was thus omitted.

The output of SUSTechPoints consisted of a .json file for each frame, containing one or more labeled objects. Each object included a key-value pair defining its id, type, position, rotation, and scale; last three are represented by the values of x, y, and z. Although these files were suitable for many machine learning frameworks,

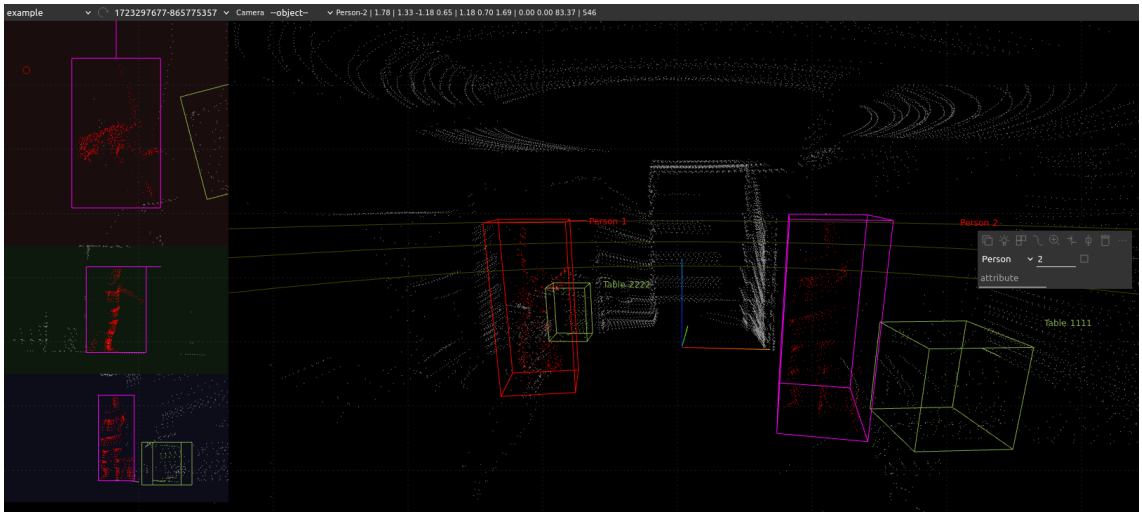

Figure 4.1: The labeling tool (SUSTechPOINTS) in action, displaying the process of defining points that belong to Person 2 object from a frame of our dataset.

MaskBEV requires KITTI-style .txt files for the labels. The main difference between these are, that in the KITTI-style files, each object's data is written in single row in just integers and floats, with one exception being the type as a string. There are also values reserved for information received from camera input, truncation, occlusion, observation angle, and bounding box coordinates. Consequently, we developed custom Python script (see Appendix A) that reads the .json data and reforms it into one-line, space-separated entries, where image related 2D information are set to zero.

Each .pcd file comprised the data captured by the Livox Mid-360 lidar: metadata describing the point cloud structure and the actual points, which included x, y, z coordinates and intensity values. Coordinates were stored as 4-byte floating points, while intensity used 8-bit integers (values between 0 and 255). We employed an open-source GitHub tool called Tools_RosBag2KITTI, which iterated over each row in the .pcd file, retrieved the coordinates and intensity, and wrote them to a binary format .bin file. Repeating this step for all frames produced the point cloud files compatible with KITTI dataset.

Afterward, the point cloud and label data were almost ready for processing as a

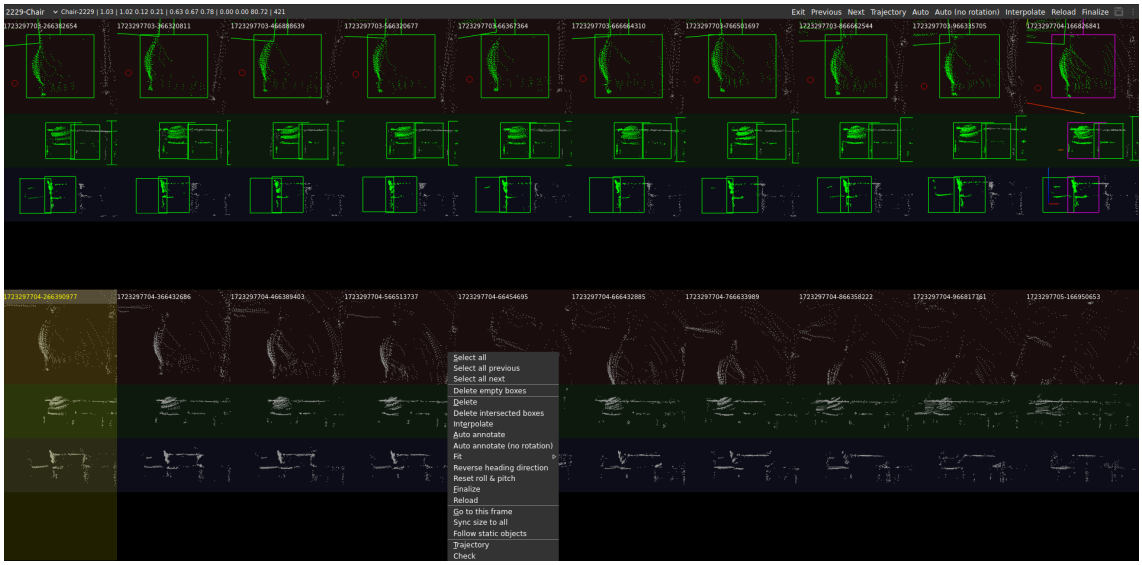

Figure 4.2: Batch editing mode in the SUSTechPoints tool that features also interpolation, that calculates the position of the object in the other selected frames. This shows the labeling of one chair object in 20 frames.

KITTI-format dataset, except file naming needed to match the KITTI convention. The algorithm demands dataset files to be presented with six digits, starting from 000000.bin/.txt. A Python script (see Appendix B) handled this renaming by incrementally numbering each file. This process finalized our custom dataset, making it ready to train the machine learning algorithm.

Last part of forming the raw point clouds into KITTI-like dataset is the calibration files for each included frames. Original KITTI utilizes sensor fusion to enable the usage of images and point clouds in the dataset, meaning it has to have separate calibration file for each frame to accurately place objects in the images and the point clouds. Since we deliberately did not use cameras, we had to create mock-up calibration files to pass through data validation of the algorithm. To simplify the dataset, we created singular calibration file that comprised of the position of the lidar relative to the robots frame, and identity matrix for the lidar-to-camera variable. Other variables that were present in the KITTI's calibration files, such as IMU-to-lidar and camera positions, were set to zero.

4.3 Machine Learning Algorithm

An essential component of autonomous perception is the machine learning algorithm. In this research, we used an existing open-source algorithm called MaskBEV [6] as our foundation. MaskBEV employs a transformer-based architecture that generates binary masks and predicts classes for detected objects using BEV representations. A high-level overview of the algorithm was provided in subsection 3.3.1. Here, we offer more detail on the MaskBEV architecture, training and prediction processes, and describe the specific modifications we made to MaskBEV. Figure 4.3 illustrates the steps included in the MaskBEV system.

The original MaskBEV author provided thorough instructions for using the algorithm with specific datasets. While clear, these instructions did not detail how to incorporate additional features or classes. This posed a challenge because we had to adapt the algorithm to utilize our custom dataset. To begin with, we examined the extensive codebase to locate critical components (dataloaders, encoders, backbones, heads, testing scripts, and configurations) and pinpointed every reference to object classes. We found them scattered over six different Python files, along with hardcoded file paths tied to the original author's setup. This meant that the code was not originally designed well for future works, and needed refactoring in several files to remove these hardcoded dependencies. Dataloader also underwent minor modifications, the original version relies on separate calibration files for each frame because of the sensor fusion, which we are not utilizing in this lidar-only approach. We modified the dataloader to use single dummy_calib.txt file for every frame, and this singular file was filled up with zeroes and an identity matrix to pass through validations in the code. Another obstacle involved advanced features, which were mostly implemented for SemanticKITTI dataset, restricting certain functionalities for KITTI-based training. Ultimately, we succeeded in training a model on our custom dataset, generating predictions and visualizations, which are detailed in

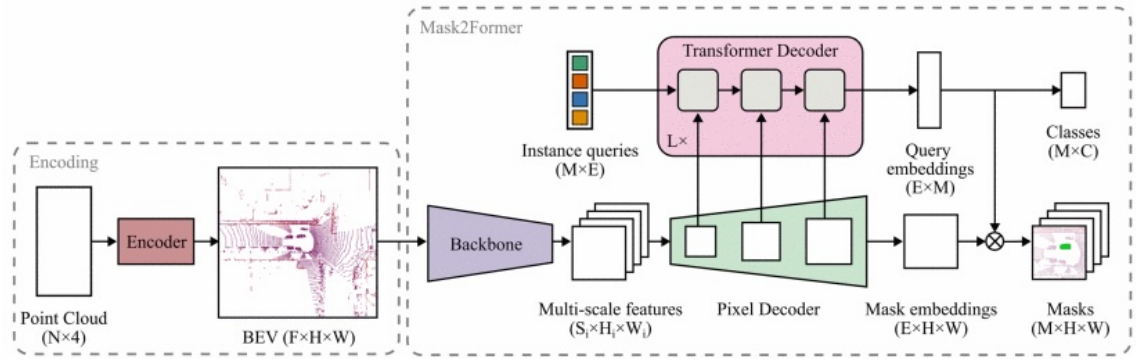

Figure 4.3: Architecture overview of MaskBEV, which is split into the Encoding component and the Mask2Former component. The former produces the BEV pseudo-image, while the latter outputs class and mask predictions. In this scope, our architecture remain same as the original. Image taken from original MaskBEV paper [6]. *Copyright © 2023, IEEE*

Section 4.4.

After configuring the code for our dataset, we needed to determine appropriate settings for model training. We encountered out-of-memory issues on our training workstation subsection 3.2.2 under certain configurations, which required iterative adjustments. The key parameters in our final configuration file included voxelization constraints, dataloader settings, and data augmentation. We set the voxel size to 0.02 meters and defined the perception area as a 10-meter horizontal diameter around the robot, with a vertical range of three meters above and below it. The dataloader used a batch size of 4 with 8 workers. Data augmentation included object noise, flipping, rotation, global noise, point dropout, point shuffling, and jitter. Most other variables at their default values from the example configurations. The complete configuration file used for our final model can be found in Appendix C.

We trained the machine learning model on the previously mentioned workstation for approximately eight hours, running a total of 102 epochs. Training automatically stopped after 30 consecutive epochs without improvement, so the best model was achieved at 72 epochs. We used AdamW [81] optimizer with a learning rate of 5.0×10^{-5} and weight decay of 1.0×10^{-4}. Because the algorithm does not use pretrained

networks, we did not apply learning rate multipliers to the backbone.

Training results in a machine learning model that can be utilized for further tasks such as 3D object detection. The architecture of our implementation is as follows: VoxelNet from mmdetection3d is used to split input point clouds into voxels of size 0.02 meters, within a 10 meter (x and y ranges from -5 to 5 meters) range in the horizontal plane. This yields grids of size H = W = 500. A 3-layer PointNet encoder (PillarFeatureNet, PointPillarScatter, LayerNorm) processes these 500 x 500 resolution voxel grids and outputs bird's eye view representation with F = 128 feature channels. A Swin-T backbone then extracts multi-scale features with S = 192 channels. The pixel decoder outputs E = 256 channels and the transformer decoder uses M = 45 queries. The query embeddings from transformer decoder are projected and correlated with the mask embeddings from pixel decoder to ultimately produce a set of classes and instance masks. All the remaining parameters follow the default parameters of Mask2Former [77].

Architecture of MaskBEV depends on mask supervision in footprint completion and also in object detection learning, meaning that ground truth masks are necessary in all parts of the machine learning algorithm. With KITTI dataset, and thus also with ours, the BEV masks are generated from data found in the label files, from 2D bounding boxes and per-point instance labels to be exact. Orthogonal projection from these are projected onto the ground plane, providing the ground truths of the objects. This ground truth mask generation allows MaskBEV to predict the complete footprint of objects, even when only part of the object is visible, but it relies on image data from the label files of KITTI dataset, while the prediction employs only point cloud data from lidar. Figure 4.4 displays how MaskBEV handles the mask generation.

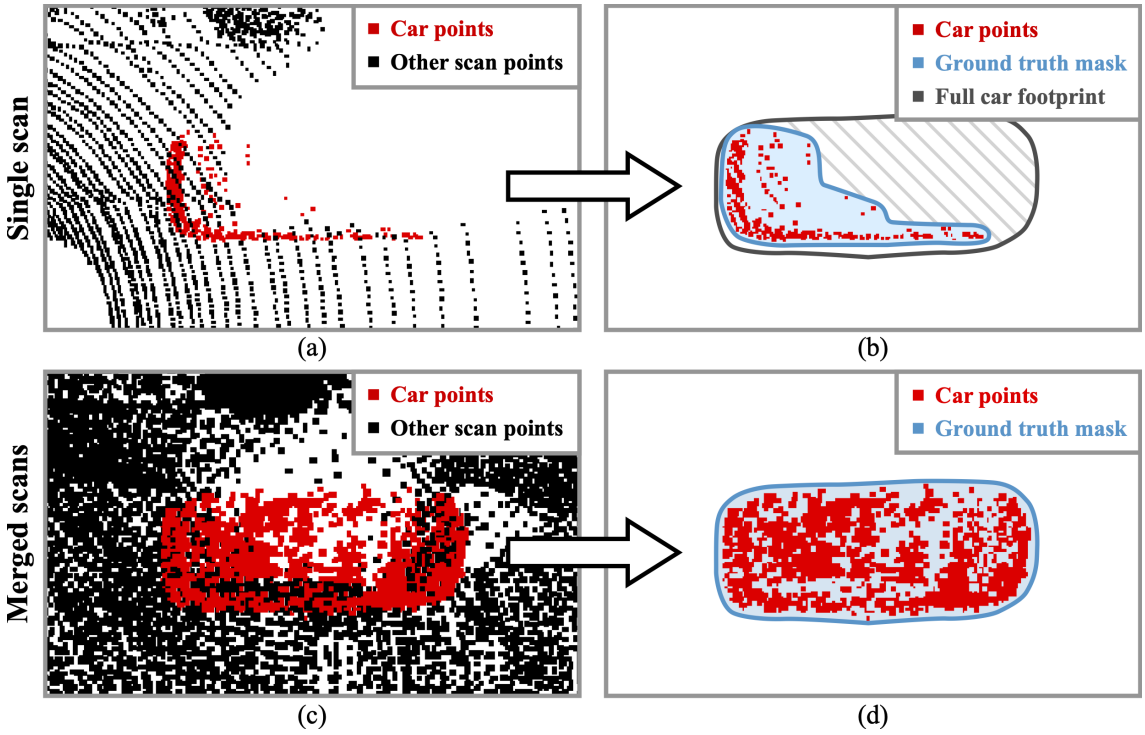

Figure 4.4: Mask generation in MaskBEV. Utilizing merged sequential lidar scans to produce complete masks that represent the whole footprint of the object. Image taken from original MaskBEV paper [6]. *Copyright © 2023, IEEE*

4.4 Testing and Visualization

Our approach in this research was mostly experimental, leveraging existing tests and visualization utilities from the MaskBEV repository. We did not include any quantitative testing on the custom dataset, but our manual labeling process helps ensure the integrity and correctness of the point clouds and labels.

The testing and visualization of our BEV framework used the same workstation as the model training, described in Subsection 3.2.2. We used the best-performing model from 72 epochs, along with the same configuration file. In the original MaskBEV codebase, there are two visualization scripts for KITTI dataset, and we adapted these scripts for our custom dataset. Each script iterates over the frames listed in val.txt file, which differ from the ones used in training (80 % for training and 20 % for validating). For each frame, the script encodes the raw point cloud

into a pseudo-image (BEV), processes it via the backbone for feature extraction, and ultimately produces class predictions and instance masks for each detected object. During this process, the script creates a new directory per frame, where it generates and saves these: An encoded BEV pseudo-image into enc.png, an image combining the ground truth masks of each object into gt.png, and an image showing the predicted masks for the objects into pred.png. The ground truth and prediction images are presented with black background and white masks as the objects. These outputs allow a qualitative evaluation of the model's performance. By comparing gt.png and pred.png, we can see how accurately the model's predictions align with the labeled ground truths.

5 Results

In this chapter, we present the results of our research, focusing on two primary outputs: our custom lidar-only indoor dataset and the performance of two MaskBEV-based approaches: our modified system and the original unmodified. We provide qualitative analyses of both the dataset and the resulting test images of the algorithms.

5.1 Custom Dataset

We first present a qualitative analysis of the new indoor lidar dataset, since we did not conduct comprehensive quantitative evaluation on it. The dataset was recorded with a Livox Mid-360 lidar in a typical indoor environment, the set comprises 520 point cloud frames of a kitchen and lounge area. Each frame is annotated by hand with eight indoor object classes (e.g. chair, couch, table, person).

Figure 5.1 shows two perspectives of frames from the dataset, screenshot from the labeling tool, and a top-down BEV pseudo-image. In the labeling view, there is displayed a frame with multiple chairs, a couch, and a person in the lounge area. The second image visualizes different frame from the dataset, but using the BEV perspective. Most of the objects in these frames are static, but frames also include two persons walking around as dynamic objects. Label quality was carefully controlled by initially employing interpolation feature, followed by manual adjustment and verification to ensure each labeled object aligns with its real-world counterpart.

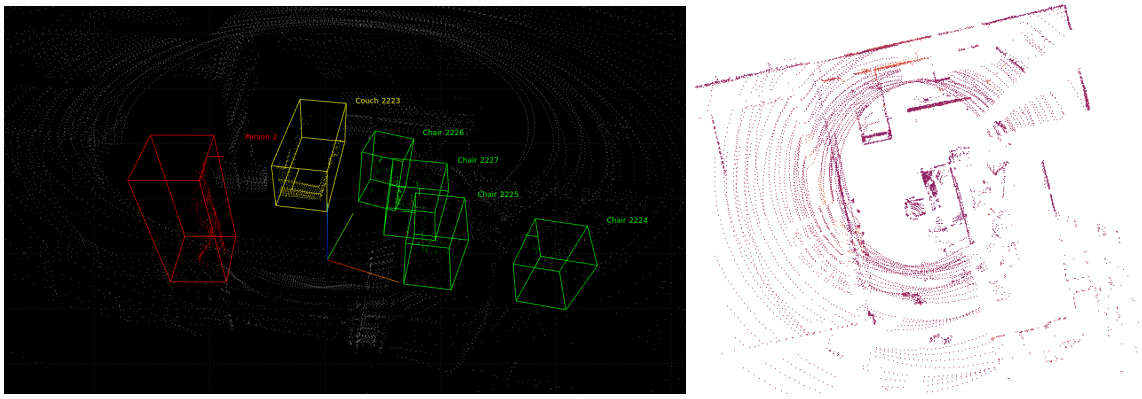

Figure 5.1: Two complementary views of the dataset. The left-hand image is a screenshot from the labeling tool SUSTechPOINTS, which shows a lidar frame overlaid with bounding boxes. The right-hand image displays a separate frame rendered as a BEV pseudo-image that the algorithm test generates. Dataset contains 520 frames like these with variety of objects.

Given that many indoor objects have complex shapes, such as chairs, careful manual correction was mandatory for precise annotations.

While this dataset offers a unique perspective on indoor lidar data, it has certain imperfections. First, it captures only one session and one physical location, which restricts the diversity of environments and objects. Second, since all the frames come from a single environment, there is imbalance of the classes, for example chair instances are way more frequent than persons. In real-world scenarios, indoor AGVs can encounter a broader range of objects in more diverse environments, so larger and more varied dataset is recommended for robust machine learning model training.

This dataset follows closely the KITTI-like structure and file formats for ease of use with existing algorithms. However, it differs in almost every other aspect, including the data gathering sensors (single lidar vs. lidar and four cameras), indoor versus outdoor environments, and the types of labeled objects (small items vs. large vehicles). Consequently, usage of KITTI-based methods to our dataset requires careful modification of data handling, especially since indoor objects tend to have smaller footprints, more complex shapes, and can be heavily occluded.

Despite its limitations, this dataset serves as a proof of concept for lidar-based indoor BEV perception. It demonstrates how a KITTI-structured dataset can be adapted for smaller-scale indoor spaces. Future improvements may include capturing data in multiple locations with different objects, lighting conditions, and more dynamic elements, like other AGVs. Main targets for development would be in the diversity and the sheer amount of data included in the dataset.

5.2 BEV Perception Framework

Having constructed the indoor dataset, we proceeded to adapt MaskBEV so that it could operate in a purely lidar-based indoor scenario. The modifications were limited to two areas. First, the dataloader was revised so that it could parse our KITTI-like label files with the indoor object classes to semantic categories expected by the algorithm. Second, a modest hyperparameter search was conducted to identify configurations suited to the small and complex dimensions of indoor objects. No changes were made to the model architecture, or other functionalities of the MaskBEV.

The experimental procedure described in Section 4.4 yields three images for every test frame: (i) an encoded BEV pseudo-image generated directly from the raw point cloud, (ii) a set of ground truth instance masks produced from the label files, and (iii) the corresponding prediction masks generated by the trained model. Test results from one actual frame is shown in Figure 5.2. All of the three artefacts are in the same coordinate system, so they could be overlaid together to illustrate the frameworks correctness.

Inspection of these results quickly revealed that our system is not working properly. The ground truth and prediction masks were displaced and distorted. Because the annotations were conducted manually, we can confidently state that the output masks are indeed different from the actual objects in the frames. Most of the ob-

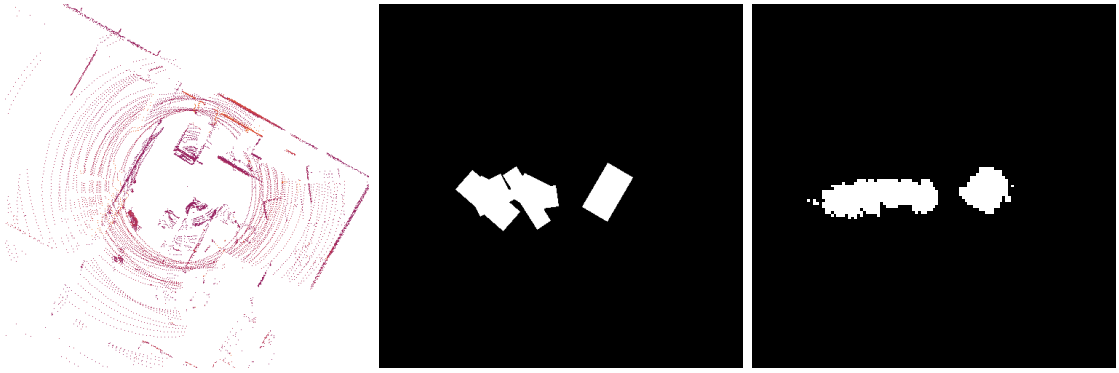

Figure 5.2: Test results from custom model trained with custom dataset. On the left is the encoded BEV pseudo-image, middle image is the ground truths of the objects, and the right is the predictions of the model. These should be compatible to be layered on top of each others, but the ground truths do not pose the actual position of the objects.

ject labels should have almost equal length sides when viewed from the bird's eye projection. Another factor is that there were no overlapping boxes in the labeling phase, and in the results almost every mask is overlapping. Also, the position of the objects are incorrect, results display the masks along the X-axis, even if that is not the case in the actual labels. These faulty results recur in all other frames as well, making every result undesirable.

A detailed debugging of every step in this process revealed that the original MaskBEV implementation constructs its ground truths by projecting camera-centric 2D bounding boxes from the KITTI label files into BEV space. This means that it relies on the presence of camera images. This is not clearly stated in the original paper or in the MaskBEV code repository. While the actual prediction phase is genuinely lidar-only, the training phase is not. And, because our dataset does not contain camera data (see Figure 3.4 and Section 5.1 for details), the ground truth generator received invalid inputs and produced faulty masks. The network then learned to imitate those errors, which leads to messy predictions visible in Figure 5.2.

Even though the positions and dimension of the ground truths would be correct, the predictions with this model would probably be still imprecise and indistinct.

This is combined result of the lack of diverse data in the dataset and the lack of training epochs in the model training.

5.3 Open-Source Dataset Results

Original plan included to train the original MaskBEV model locally using the open-source KITTI dataset. Unfortunately, the total 48 GiB VRAM memory from the two RTX 3090 cards was insufficient for the full-resolution input demanded by the official configuration files and the much larger dataset. Hence, we could not do these test ourselves, and had to include results from the original paper. We trust the original authors and their test results, and give all credits to them.

Figure 5.3 displays the successful predictions that their model achieved. These images are combination of the three images that the tests provide, the ground truths and the prediction masks are layered on top of the encoded BEV pseudo-image to prove its effectiveness. There are some imperfections marked with red arrows, but

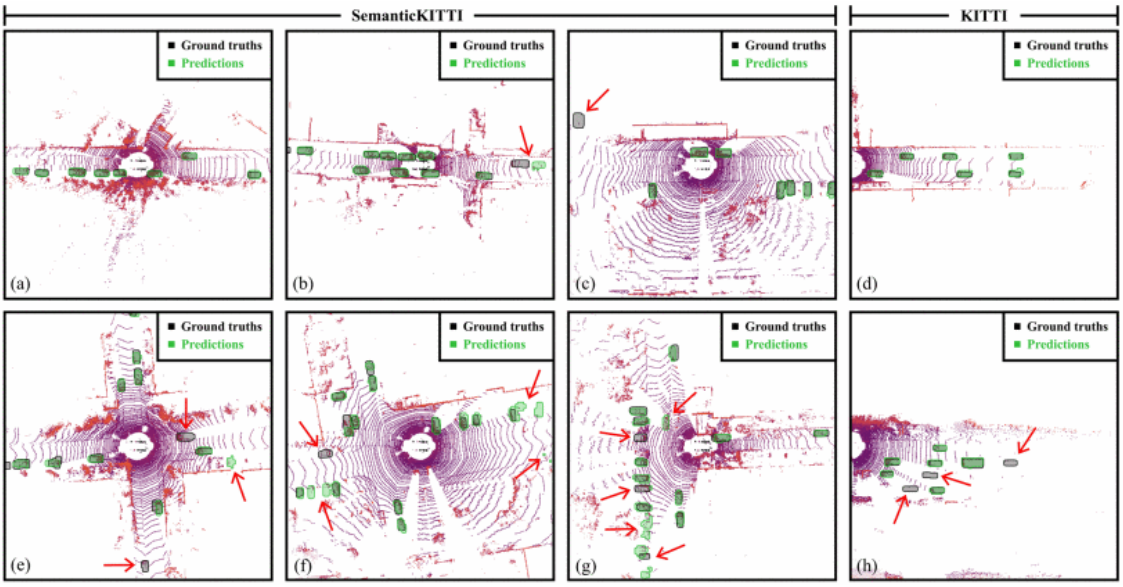

Figure 5.3: Original MaskBEV predictions on SemanticKITTI and KITTI datasets. First row showcases good predictions and second row displays failure cases, which are the results of too complex scenarios. Image taken from the original paper [6]. *Copyright © 2023, IEEE*

with there are not same kind of fundamental flaws with mask positions like with our system.

In the outdoor driving scenarios, MaskBEV performs convincingly, achieving great accuracy on SemanticKITTI and competitive results on KITTI. Importantly, in case of failures, the masks are broadly aligned with the BEV pseudo-images. The main weakness is in complex scenarios, where there are object ambiguity and several detected objects. This does not have the same coordinate errors as our system.

The original MaskBEV focused using the SemanticKITTI dataset, but the tests on the research paper also included some with the use of the KITTI dataset. If we could train the original model locally, we would have emphasized the experiments on the KITTI dataset, which would have made the comparisons more compatible.

6 Conclusion and Future Work

6.1 Conclusion

This research set out to determine if BEV perception strategy could provide reliable object detection for AGVs, using only lidar sensor. The work was guided by three general research questions and produced three concrete contributions.

Regarding the two research questions presented in Chapter 1. The BEV refers to a data representation that projects three-dimensional sensor data onto a two-dimensional plane by compressing the Y-axis, providing a self-referenced view from top to down. For use cases with lidar, each 3D point is mapped to a cell in 2D coordinates, and the height values are encoded as additional channels. Resulting in a pseudo-image that captures the objects across the floor plane while discarding perspective distortions that happen in camera images. BEV is popular in modern perception because it provides a view that aligns naturally with planning and control layers, like blueprints. Also, with minimal modifications the flattened data can be processed with mature 2D methods, such as convolutional or transformer-based networks. In the indoor AGV context, the lidar-only BEV offers three benefits: (i) it is non-dependent of external lights, so it can even function in dark rooms, (ii) it captures fine geometric details, crucial for navigating through complex objects and narrow paths, and (iii) its grid-aligned outputs can be easily employed by lightweight path-planning methods.

While addressing these questions, the thesis produced three main contributions. It delivered a novel indoor lidar dataset comprising 520 carefully annotated point cloud frames from a kitchen-lounge environment. Despite being relatively small in size, the dataset works as an example for future indoor perception studies and demonstrates that KITTI-style structure can be achieved even with different sensors and object classes. To accompany this custom dataset, we adapted the MaskBEV network to be compatible with the new dataset. This adaptation did not succeed as intended, because it turned out that MaskBEV's mask generation relies on 2D camera data from the label files. Also, the original MaskBEV was developed for outdoor driving environments, to detect large objects like cars and trucks, making the adaptation process require modifications to the configurations and core functionalities to make it optimized for our scenarios. Finally, we concluded that the unmodified MaskBEV with the results of the original paper could not be accurately compared with our version due the failed local training of the unmodified model and the faulty results that our system provided. MaskBEV performs reliably in its own scope, but our indoor approach needs different base network, or a thorough reprogramming of the whole codebase.

After these results I have come to the conclusion that MaskBEV does not provide suitable foundation for this indoor AGV perception task. Even though MaskBEV may not appropriate algorithm for this task, that does not mean that the BEV perception strategy is bad for it. Vice versa, there are multiple advantages that support the usage of BEV methods indoors, but this research proved that MaskBEV is not the obvious choice.

6.2 Future Works

Following the conclusions about the utilization of BEV strategies with indoor AGVs, this research could benefit from more in-depth study and development. In this thesis,

we had limited time and resources, and we could not make the adapted MaskBEV work reliability and robustly in the indoor environments.

Future research should consist either seeking for a new base algorithm that is more suitable for this task, or going deeper into refactoring this framework by fine-tuning the weak spots. There are two main things to fix in our approach: (i) the faulty mask generation, and (ii) making the whole perception system more lightweight in terms of memory and processor usage, to be more suitable for platforms with limited computing resources, i.e. small scale AGVs.

Appendix A Label File Python Script

```
import os
import json
import argparse
import math

parser = argparse.ArgumentParser(description='convert label to kitti format')
parser.add_argument('src', type=str,default='./data', help="source data folder")
parser.add_argument('tgt', type=str,default='./data_kitti', help="target folder")
parser.add_argument('--scenes', type=str,default='.*', help="")
parser.add_argument('--frames', type=str,default='.*', help="")

args = parser.parse_args()

scenes = os.listdir(args.src)

for s in scenes:
  labels = os.listdir(os.path.join(args.src, s, 'label'))
  for l in labels:
    with open(os.path.join(args.src, s, 'label', l))  as fin:
```

```
20      label = json.load(fin)
21
22    if 'objs' in label:
23      label = label['objs']
24
25    output_path = os.path.join(args.tgt, s, 'label_kitti')
26    if not os.path.exists(output_path):
27      os.makedirs(output_path)
28
29    with open(os.path.join(output_path, os.path.splitext(l)[0]+".
    txt"), 'w') as fout:
30      for obj in label:
31        line = "{} 0.00 0 0.00 0.00 0.00 0.00 0.00 {:.2f} {:.2f}
    {:.2f} {:.2f} {:.2f} {:.2f} {:.2f}\n".format(
32          obj['obj_type'],
33          float(obj['psr']['scale']['z']), #h
34          float(obj['psr']['scale']['y']), #w
35          float(obj['psr']['scale']['x']), #l
36          -float(obj['psr']['position']['y']), #x
37          -float(obj['psr']['position']['z']) + 0.5*float(obj['psr'
    ]['scale']['z']), #y
38          float(obj['psr']['position']['x']), #z
39          -float(obj['psr']['rotation']['z']) - math.pi/2  #
    rotation_y
40        )
41
42        fout.write(line)
```

Appendix B Rename Python Script

```python
import os
from pathlib import Path

velodyne_path = Path('data/KITTI/data_object_velodyne/training/velodyne')
label_path = Path('data/KITTI/data_object_label_2/training/label_2')

velodyne_files = sorted(velodyne_path.glob('*.bin'))
label_files = sorted(label_path.glob('*.txt'))

for i, (velodyne_file, label_file) in enumerate(zip(velodyne_files, label_files)):
  new_name = f"{i:06d}"
  velodyne_file.rename(velodyne_path / f"{new_name}.bin")

  label_file.rename(label_path / f"{new_name}.txt")

print("Files have been renamed successfully!")
```

Appendix C Configuration File

```yaml
# General Settings
seed: 420
checkpoint:

# Model
optimiser_type: adam_w  # AdamW optimizer
lr: 5.0e-5
weight_decay: 1.0e-4
lr_schedulers_type: plateau
differential_lr: False
differential_lr_scaling: 0.1

# LiDAR Point Cloud Range & Voxelization
x_range: [ -5, 5 ]  # x-range for the LiDAR point cloud
y_range: [ -5, 5 ]  # y-range for the LiDAR point cloud
z_range: [ -3, 3 ]  # Narrowing down the z-range for more relevant
   vertical data
voxel_size: 0.02  # Decrease voxel size for higher granularity
num_queries: 45  # Number of object queries for Mask_bev

# Head Settings
head_reverse_class_weights: False  # Enable reverse class weights
   if class imbalance is an issue

# Encoder Settings
```

```yaml
max_num_points: 32  # Number of points to sample per voxel
encoder_feat_channels: [128, 128, 128]  # Feature channels for the encoder, keeping it consistent
head_feat_channels: 256
head_out_channels: 256

# Backbone Settings
backbone_embed_dim: 192  # Embedding dimensions for the backbone
backbone_use_abs_emb: False  # Use absolute positional embeddings

# Dataset Settings
dataset: kitti  # Ensure dataset is set to KITTI
batch_size: 4
num_workers: 8  # Keep this for optimal data loading
pin_memory: True
remove_unseen: True  # Remove unseen data points from the training process to improve generalization
shuffle_train: True  # Shuffle training data for better convergence
min_num_points: 1  # Minimum number of points to consider when training on objects

augmentations:
  - name: 'object_sample'
    dataset_root: 'data/KITTI'
    num_sample: 15
  - name: 'object_noise'
  - name: 'flip'
    prob_flip_x: 0
    prob_flip_y: 0.5
  - name: 'rotate'
    rotate_prob: 0.1
    rotation_range: 2.5
  - name: 'global_noise'
```

```
54      prob_aug: 0.5
55    - name: 'drop'
56      prob_drop: 0.1
57      per_point_drop_prob: 0.05
58    - name: 'shuffle'
59      prob_shuffle: 0.5
60    - name: 'jitter'
61      prob_jitter: 0.25
62      jitter_std: 0.01
63      intensity_std: 0.01
```

References

[1] Y.-C. Liu, K.-Y. Lin, and Y.-S. Chen, "Bird's-eye view vision system for vehicle surrounding monitoring", in *Robot Vision*, G. Sommer and R. Klette, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 207–218, ISBN: 978-3-540-78157-8. DOI: https://doi.org/10.1007/978-3-540-78157-8_16.

[2] R. T. Brown, "A new lidar for meteorological application", *Journal of Applied Meteorology and Climatology*, vol. 12, no. 4, pp. 698–708, 1973. DOI: 10.1175/1520-0450(1973)012<0698:ANLFMA>2.0.CO;2.

[3] M. Jaboyedoff, T. Oppikofer, A. Abellán, *et al.*, "Use of lidar in landslide investigations: A review", *Natural Hazards*, no. 1, 2012. DOI: 10.1007/s11069-010-9634-2.

[4] K. Lim, P. Treitz, M. Wulder, B. St-Onge, and M. Flood, "Lidar remote sensing of forest structure", *Progress in Physical Geography: Earth and Environment*, vol. 27, no. 1, pp. 88–106, 2003. DOI: 10.1191/0309133303pp360ra.

[5] T. Raj, F. H. Hashim, A. B. Huddin, M. F. Ibrahim, and A. Hussain, "A survey on lidar scanning mechanisms", *Electronics*, vol. 9, no. 5, p. 741, 2020. DOI: https://doi.org/10.3390/electronics9050741.

[6] W. Guimont-Martin, J.-M. Fortin, F. Pomerleau, and P. Giguère, "Maskbev: Joint object detection and footprint completion for bird's-eye view 3d point clouds", in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2023, pp. 5677–5684.

[7] H. Li, C. Sima, J. Dai, *et al.*, "Delving into the devils of bird's-eye-view perception: A review, evaluation and recipe", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 4, pp. 2151–2170, 2024. DOI: 10.1109/TPAMI.2023.3333838.

[8] C. Li, L. Wang, X. Zhang, X. Zhang, J. Yi, and H. Ye, "Bevtrajnet: Bird's eye view trajectory prediction in factory scenarios based on lidar data", in *2024 4th Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, 2024, pp. 295–300. DOI: 10.1109/ACCTCS61748.2024.00059.

[9] Y. Liao, J. Xie, and A. Geiger, "Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 3292–3310, 2023. DOI: 10.1109/TPAMI.2022.3179507.

[10] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research", *Sensors*, vol. 19, no. 3, p. 648, 2019. DOI: https://doi.org/10.3390/s19030648.

[11] F. Zhang, D. Clarke, and A. Knoll, "Vehicle detection based on lidar and camera fusion", in *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2014, pp. 1620–1625. DOI: 10.1109/ITSC.2014.6957925.

[12] D. Göhring, M. Wang, M. Schnürmacher, and T. Ganjineh, "Radar/lidar sensor fusion for car-following on highways", in *The 5th International Conference on Automation, Robotics and Applications*, IEEE, 2011, pp. 407–412. DOI: 10.1109/ICARA.2011.6144918.

[13] F. Garcia, D. Martin, A. De La Escalera, and J. M. Armingol, "Sensor fusion methodology for vehicle detection", *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 123–133, 2017. DOI: 10.1109/MITS.2016.2620398.

[14] O. L. Junior, D. Delgado, V. Gonçalves, and U. Nunes, "Trainable classifier-fusion schemes: An application to pedestrian detection", in *2009 12Th international IEEE conference on intelligent transportation systems*, IEEE, 2009, pp. 1–6. DOI: 10.1109/ITSC.2009.5309700.

[15] M. J. O. Erzhuo Che and J. Jung, "Efficient segment-based ground filtering and adaptive road detection from mobile light detection and ranging (lidar) data", *International Journal of Remote Sensing*, vol. 42, no. 10, pp. 3633–3659, 2021. DOI: 10.1080/01431161.2020.1871095.

[16] S. Chen, B. Liu, C. Feng, C. Vallespi-Gonzalez, and C. Wellington, "3d point cloud processing and learning for autonomous driving: Impacting map creation, localization, and perception", *IEEE Signal Processing Magazine*, vol. 38, no. 1, pp. 68–86, 2021. DOI: 10.1109/MSP.2020.2984780.

[17] S. A. Bello, S. Yu, C. Wang, J. M. Adam, and J. Li, "Deep learning on 3d point clouds", *Remote Sensing*, vol. 12, no. 11, p. 1729, 2020.

[18] M. Himmelsbach, A. Mueller, T. Lüttel, and H.-J. Wünsche, "Lidar-based 3d object perception", in *Proceedings of 1st international workshop on cognition for technical systems*, vol. 1, 2008. [Online]. Available: https://www.researchgate.net/profile/Thorsten-Luettel/publication/229018428_LIDAR-based_3D_object_perception/links/56ab282f08aed814bde7ac1f/LIDAR-based-3D-object-perception.pdf (visited on 10/12/2024).

[19] M. Jokela, M. Kutila, and P. Pyykönen, "Testing and validation of automotive point-cloud sensors in adverse weather conditions", *Applied Sciences*, vol. 9, no. 11, p. 2341, 2019. DOI: https://doi.org/10.3390/app9112341.

[20] E. A. L. Narváez and N. E. L. Narváez, "Point cloud denoising using robust principal component analysis", in *International Conference on Computer Graphics Theory and Applications*, SCITEPRESS, vol. 2, 2006, pp. 51–58. DOI: 10.5220/0001358900510058.

[21] F. Zaman, Y. P. Wong, and B. Y. Ng, "Density-based denoising of point cloud", in *9th International Conference on Robotic, Vision, Signal Processing and Power Applications*, H. Ibrahim, S. Iqbal, S. S. Teoh, and M. T. Mustaffa, Eds., Singapore: Springer Singapore, 2017, pp. 287–295, ISBN: 978-981-10-1721-6. DOI: https://doi.org/10.1007/978-981-10-1721-6_31.

[22] M.-H. Le, C.-H. Cheng, and D.-G. Liu, "An efficient adaptive noise removal filter on range images for lidar point clouds", *Electronics*, vol. 12, no. 9, p. 2150, 2023. DOI: https://doi.org/10.3390/electronics12092150.

[23] A. Kurup and J. Bos, "Dsor: A scalable statistical filter for removing falling snow from lidar point clouds in severe winter weather", *arXiv preprint arXiv:2109.07078*, 2021. DOI: https://doi.org/10.48550/arXiv.2109.07078.

[24] N. Charron, S. Phillips, and S. L. Waslander, "De-noising of lidar point clouds corrupted by snowfall", in *2018 15th Conference on Computer and Robot Vision (CRV)*, IEEE, 2018, pp. 254–261. DOI: 10.1109/CRV.2018.00043.

[25] L. Zhou, G. Sun, Y. Li, W. Li, and Z. Su, "Point cloud denoising review: From classical to deep learning-based approaches", *Graphical Models*, vol. 121, p. 101 140, 2022, ISSN: 1524-0703. DOI: https://doi.org/10.1016/j.gmod.2022.101140.

[26] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra, "Pcpnet learning local shape properties from raw point clouds", in *Computer graphics forum*, Wiley Online Library, vol. 37, 2018, pp. 75–85. DOI: https://doi.org/10.1111/cgf.13343.

[27] F. Pistilli, G. Fracastoro, D. Valsesia, and E. Magli, "Learning graph-convolutional representations for point cloud denoising", in *European conference on computer vision*, Springer, 2020, pp. 103–118. DOI: https://doi.org/10.1007/978-3-030-58565-5_7.

[28] Y. Li and J. Ibanez-Guzman, "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems", *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020. DOI: 10.1109/MSP.2020.2973615.

[29] E. Che and M. J. Olsen, "An efficient framework for mobile lidar trajectory reconstruction and mo-norvana segmentation", *Remote Sensing*, vol. 11, no. 7, p. 836, 2019. DOI: https://doi.org/10.3390/rs11070836.

[30] J. Chen and X. Ran, "Deep learning with edge computing: A review", *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019. DOI: 10.1109/JPROC.2019.2921977.

[31] N. H. H. Aung, P. Sangwongngam, R. Jintamethasawat, S. Shah, and L. Wuttisittikulkij, "A review of lidar-based 3d object detection via deep learning approaches towards robust connected and autonomous vehicles", *IEEE Transactions on Intelligent Vehicles*, 2024. DOI: 10.1109/TIV.2024.3415771.

[32] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4338–4364, 2021. DOI: 10.1109/TPAMI.2020.3005434.

[33] Y. Li, L. Ma, Z. Zhong, *et al.*, "Deep learning for lidar point clouds in autonomous driving: A review", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, 2020. DOI: 10.1109/TNNLS.2020.3015992.

[34] Y. Wu, Y. Wang, S. Zhang, and H. Ogai, "Deep 3d object detection networks using lidar data: A review", *IEEE Sensors Journal*, vol. 21, no. 2, pp. 1152–1171, 2020. DOI: 10.1109/JSEN.2020.3020626.

[35] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499. DOI: https://doi.org/10.48550/arXiv.1711.06396.

[36] Y. Chen, J. Liu, X. Zhang, X. Qi, and J. Jia, "Voxelnext: Fully sparse voxelnet for 3d object detection and tracking", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 21 674–21 683. DOI: 10.1109/CVPR52729.2023.02076.

[37] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 12 697–12 705. DOI: 10.1109/CVPR.2019.01298.

[38] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660. DOI: https://doi.org/10.48550/arXiv.1612.00593.

[39] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 770–779. DOI: 10.1109/CVPR.2019.00086.

[40] G. Qian, Y. Li, H. Peng, *et al.*, "Pointnext: Revisiting pointnet++ with improved training and scaling strategies", *Advances in neural information pro-*

cessing systems, vol. 35, pp. 23 192–23 204, 2022. DOI: https://doi.org/10.48550/arXiv.2206.04670.

[41] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space", *Advances in neural information processing systems*, vol. 30, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf (visited on 01/12/2025).

[42] Y. Zhang and M. Rabbat, "A graph-cnn for 3d point cloud classification", in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 6279–6283. DOI: 10.1109/ICASSP.2018.8462291.

[43] H. Yang, Z. Liu, X. Wu, *et al.*, "Graph r-cnn: Towards accurate 3d object detection with semantic-decorated local graph", in *European Conference on Computer Vision*, Springer, 2022, pp. 662–679. DOI: https://doi.org/10.1007/978-3-031-20074-8_38.

[44] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving", in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915. DOI: 10.1109/CVPR.2017.691.

[45] Z. Liu, Y. Lin, Y. Cao, *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows", in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022. DOI: 10.1109/ICCV48922.2021.00986.

[46] R. Girshick, "Fast r-cnn", in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[47] W. Liu, Y. Wen, Z. Yu, and M. Yang, *Large-margin softmax loss for convolutional neural networks*, 2017. DOI: https://doi.org/10.48550/arXiv.1612.02295. arXiv: 1612.02295 [stat.ML].

[48] X. Wu, L. Peng, H. Yang, *et al.*, "Sparse fuse dense: Towards high quality 3d detection with depth completion", in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 5408–5417. DOI: 10.1109/CVPR52688.2022.00534.

[49] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. [Online]. Available: https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d.

[50] H. Wu, C. Wen, S. Shi, X. Li, and C. Wang, "Virtual sparse convolution for multimodal 3d object detection", in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 21 653–21 662. DOI: 10.1109/CVPR52729.2023.02074.

[51] Z. Tian, X. Chu, X. Wang, X. Wei, and C. Shen, "Fully convolutional one-stage 3d object detection on lidar range images", in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 34 899–34 911. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/e1f418450107c4a0ddc16d008d131573-Paper-Conference.pdf.

[52] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3dssd: Point-based 3d single stage object detector", in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. DOI: 10.1109/CVPR42600.2020.01105.

[53] Y. Li, Z. Ge, G. Yu, *et al.*, "Bevdepth: Acquisition of reliable depth for multi-view 3d object detection", in *Proceedings of the AAAI Conference on Artificial*

Intelligence, vol. 37, 2023, pp. 1477–1485. DOI: https://doi.org/10.1609/aaai.v37i2.25233.

[54] Y. Li, H. Bao, Z. Ge, J. Yang, J. Sun, and Z. Li, "Bevstereo: Enhancing depth estimation in multi-view 3d object detection with temporal stereo", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 1486–1494. DOI: https://doi.org/10.1609/aaai.v37i2.25234.

[55] C. Reading, A. Harakeh, J. Chae, and S. L. Waslander, "Categorical depth distribution network for monocular 3d object detection", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8555–8564. DOI: https://doi.org/10.48550/arXiv.2103.01100.

[56] J. Zhao, J. Shi, and L. Zhuo, "Bev perception for autonomous driving: State of the art and future perspectives", *Expert Systems with Applications*, vol. 258, p. 125 103, 2024, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2024.125103.

[57] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang, "3d object detection with pointformer", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 7463–7472. DOI: 10.1109/CVPR46437.2021.00738.

[58] X. Bai, Z. Hu, X. Zhu, *et al.*, "Transfusion: Robust lidar-camera fusion for 3d object detection with transformers", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 1090–1099. DOI: 10.1109/CVPR52688.2022.00116.

[59] T. Liang, H. Xie, K. Yu, *et al.*, "Bevfusion: A simple and robust lidar-camera fusion framework", *Advances in Neural Information Processing Systems*, vol. 35, pp. 10 421–10 434, 2022. DOI: https://doi.org/10.48550/arXiv.2205.13790.

[60] Y. Li, B. Huang, Z. Chen, *et al.*, "Fast-bev: A fast and strong bird's-eye view perception baseline", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. DOI: 10.1109/TPAMI.2024.3414835.

[61] Z. Zhu, Y. Zhang, H. Chen, *et al.*, "Understanding the robustness of 3d object detection with bird's-eye-view representations in autonomous driving", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 21 600–21 610. DOI: https://doi.ieeecomputersociety.org/10.1109/CVPR52729.2023.02069.

[62] M. H. Ng, K. Radia, J. Chen, D. Wang, I. Gog, and J. E. Gonzalez, "Bev-seg: Bird's eye view semantic segmentation using geometry and semantic point cloud", *arXiv preprint arXiv:2006.11436*, 2020. DOI: https://doi.org/10.48550/arXiv.2006.11436.

[63] Z. Li, W. Wang, H. Li, *et al.*, "Bevformer: Learning bird's-eye-view representation from lidar-camera via spatiotemporal transformers", 2024, pp. 1–18. DOI: 10.1109/TPAMI.2024.3515454.

[64] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[65] H. Caesar, V. Bankiti, A. H. Lang, *et al.*, "Nuscenes: A multimodal dataset for autonomous driving", in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. DOI: 10.1109/CVPR42600.2020.01164.

[66] P. Sun, H. Kretzschmar, X. Dotiwalla, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset", in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. DOI: 10.1109/CVPR42600.2020.00252.

[67] J. Mao, M. Niu, C. Jiang, *et al.*, *One million scenes for autonomous driving: Once dataset*, 2021. DOI: https://doi.org/10.48550/arXiv.2106.11037. arXiv: 2106.11037 [cs.CV].

[68] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. [Online]. Available: https://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d (visited on 02/20/2025).

[69] Motional, "Nuscenes detection task", 2024. [Online]. Available: https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any (visited on 02/24/2025).

[70] Waymo, *Open waymo datase*, 2025. [Online]. Available: https://waymo.com/open/data/perception/ (visited on 02/25/2025).

[71] Huawei, *Once documentation*, 2025. [Online]. Available: https://once-for-auto-driving.github.io/documentation.html (visited on 02/26/2025).

[72] Y. Liao, J. Xie, and A. Geiger, *Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d*, 2022. [Online]. Available: https://www.cvlibs.net/datasets/kitti-360/index.php (visited on 02/20/2025).

[73] A. Kumar, Y. Guo, X. Huang, L. Ren, and X. Liu, "Seabird: Segmentation in bird's view with dice loss improves monocular 3d detection of large objects", *arXiv preprint arXiv:2403.20318*, 2024. DOI: https://doi.org/10.48550/arXiv.2403.20318.

[74] Z. Wang, Z. Huang, Y. Gao, N. Wang, and S. Liu, *Mv2dfusion: Leveraging modality-specific object semantics for multi-modal 3d detection*, 2024. DOI: https://doi.org/10.48550/arXiv.2408.05945. arXiv: 2408.05945 [cs.CV].

[75] T. Ma, X. Yang, H. Zhou, *et al.*, "Detzero: Rethinking offboard 3d object detection with long-term sequential point clouds", in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. DOI: 10.1109/ICCV51070.2023.00620.

[76] Livox, *Livox mid-360*, 2025. [Online]. Available: https://www.livoxtech.com/mid-360 (visited on 03/20/2025).

[77] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, "Masked-attention mask transformer for universal image segmentation", in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 1280–1289. DOI: 10.1109/CVPR52688.2022.00135.

[78] xmfcx, *Rosbag2_to_pcd*, 2023. [Online]. Available: https://github.com/xmfcx/rosbag2_to_pcd/tree/main (visited on 04/10/2025).

[79] E. Li, S. Wang, C. Li, D. Li, X. Wu, and Q. Hao, "Sustech points: A portable 3d point cloud interactive annotation platform system", in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 1108–1115. DOI: 10.1109/IV47402.2020.9304562.

[80] leofansq, *Tools_rosbag2kitti*, 2018. [Online]. Available: https://github.com/leofansq/Tools_RosBag2KITTI (visited on 04/10/2025).

[81] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019. DOI: https://doi.org/10.48550/arXiv.1711.05101. arXiv: 1711.05101 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.05101.