Informatics

Deep Hough Voting für 3D Objekterkennung und Posenschätzung in LiDAR Punktwolken

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Lukas Reisinger, BSc
Matrikelnummer 01363863

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze
Mitwirkung: Dipl.-Ing. Dr.techn. Csaba Beleznai

Wien, 4. September 2023

Lukas Reisinger	Markus Vincze

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Informatics

Deep Hough Voting based 3D object detection and pose estimation in LiDAR point clouds

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Lukas Reisinger, BSc
Registration Number 01363863

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze
Assistance: Dipl.-Ing. Dr.techn. Csaba Beleznai

Vienna, 4th September, 2023

Lukas Reisinger	Markus Vincze

Erklärung zur Verfassung der Arbeit

Lukas Reisinger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 4. September 2023

Lukas Reisinger

Danksagung

Diese Arbeit wurde in einer Zusammenarbeit des ACIN Instituts der Technischen Universität Wien (TU Wien) und des Instituts für Assistive und Autonomous Systems (AAS) des Austrian Institute of Technology (AIT) realisiert.
Ich möchte mich herzlich bei allen bedanken, die mich im Verlauf dieser Diplomarbeit unterstützt haben. Zurerst möchte ich Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze meinen Dank für seine umfassende Betreuung und Begutachtung dieser Arbeit ausdrücken. Ein besonderer dank ergeht an Dipl.-Ing. Dr.techn. Csaba Beleznai für seine stetige Unterstützung und wertvollen Ratschläge während des gesamten Projekts. Nicht zuletzt möchte ich meiner Familie und Partnerin für die kontinuierliche Unterstützung meinen ausdrücklichen Dank ausprechen.

Acknowledgements

This work was realized in collaboration between the ACIN Institute of the Technical University of Vienna (TU Wien) and the Institute for Assistive and Autonomous Systems (AAS) of the Austrian Institute of Technology (AIT).
I would like to extend my heartfelt gratitude to everyone who supported me throughout the course of this diploma thesis. Firstly, I'd like to express my appreciation to Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze for his comprehensive supervision and review of this work. Special thanks go to Dipl.-Ing. Dr.techn. Csaba Beleznai for his consistent support and invaluable guidance throughout the entire project. Last but certainly not least, I would like to expressly thank my family and partner for their continuous support.

Kurzfassung

Räumliches Denken, insbesondere die Fähigkeit, Objekte in einer 3D-Umgebung zu erkennen und zu identifizieren, ist für Robotersysteme, die in unbekannten Umgebungen navigieren und arbeiten sollen, von entscheidender Bedeutung. In dieser Arbeit stellen wir eine Methode vor, die speziell darauf ausgelegt ist, quaderförmige Objekte in Punktewolken durch einen vote-basierten Mechanismus zu erkennen und ihre 9DoF-Pose zu schätzen. Unser Erkennungsschema wird durch eine Pipeline für die Erzeugung synthetischer Daten ergänzt, die zur Generierung der erforderlichen Trainingsdaten genutzt wird.
Unsere Experimente zeigen, dass unsere Methode robuste Ergebnisse liefert, wenn sie auf echte Daten angewendet wird, obwohl sie ausschließlich auf synthetischen Daten trainiert wurde. Die Untersuchung der sim-to-real gap zeigt nur eine minimale Verschlechterung der Orientierungsschätzung und einen moderaten Rückgang der Detektionsfähigkeit. Wir testen verschiedene Darstellungen der Orientierung und schlagen eine Methode vor, um äquivalente, aber unterschiedliche Orientierungen von Quadern auf eine einzige kanonische Orientierung in einer deterministischen Weise abzubilden.

Abstract

Spatial reasoning, particularly the ability to detect and recognize objects within a 3D environment, is crucial for robotic systems aiming to navigate and function in unfamiliar settings. In this thesis, we introduce a method specifically designed for detecting cuboid-shaped objects within point clouds through a voting-based mechanism and estimating their 9DoF pose. Our detection framework is accompanied by a synthetic data generation pipeline, which is utilized to generate the necessary training data.
Our evaluation reveal that our method exhibits robust performance when applied to real-world data, even though it was exclusively trained on synthetic data. The examination of the sim-to-real gap shows just minimal degradation of orientation estimation and a moderate decline in detection capability. We test different orientation representations and propose a way to map equivalent but distinct orientations of cuboids to a single canonical orientation in a deterministic way.

Contents

CHAPTER 1

Introduction and Motivation

Spatial sensing is a key human ability to perceive objects in a spatial context, and a prerequisite for their manipulation. Transferring this spatial sensing and reasoning ability to machines is thus crucial to enable them executing human-level tasks. In this process, object parameters are commonly described by 3D world space attributes, therefore the related vision tasks are often denoted as 3D object detection and pose estimation. Recent advances in deep learning have resulted in a great increase of representational power and accomplished accuracy, yielding increasingly more accurate percepts across a broad variation of settings, with strong priors directly learned from training data. Simultaneously, depth sensing sensors (stereo, ToF, LiDAR, RADAR) and methodologies (monocular 3D pose, monocular-depth, NERF) provide enhancements in depth accuracy and range, facilitating the advancement of autonomous driving and industrial automation. Despite the great progress, however, still several challenges exist:

- **data problem:** supervised learning is still prevailing, where manual labeling of 3D pose parameters is not straightforward. Unsupervised/self-supervised methodologies here still fail to bridge the quality gap needed for practical applications,

- **robustness with respect to missing/noisy data:** occlusions, self-occlusions and noisy input represent for many pose-aware detection methodologies a substantial problem. Especially, holistic (object-level) representations assume full availability of data, and an exhaustive enumeration of all possible occlusion states during training time.

- **representational constraints in terms of a closed-set condition:** many object detectors can only detect objects seen during training time, not being able to detect objects abstracted as instances based on the composition of some basic entities.

3D object detection is the task of recognizing and localizing objects in 3D space. Typically, objects are detected in the sensor-centered coordinate frame. A transformation into a global or vehicle-centered frame requires information about the relative positioning and rotation between the vehicle and sensor (kinematics, joint states) and between vehicle and the global origin (Global Positioning System *GPS*, Local Positioning System *LPS*). Figure 1.1 illustrates the dependencies between these coordinate frames when a robot interacts with an environment.

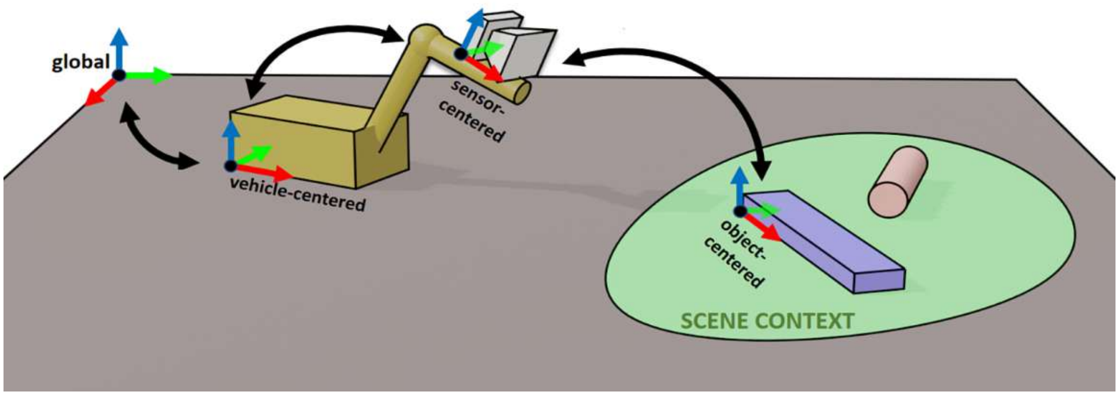


Figure 1.1: Coordinate frames in a scenario where a robot interacts with objects

3D object detection is crucial for autonomous systems navigating in unknown settings. As industrial automation advances, systems that understand and accurately represent their environment become increasingly vital. The challenges in 3D object detection and pose estimation include variation in appearance, multiple object categories, occlusions, and noise. Pose estimation can range in complexity, with 9DoF (degrees of freedom) being the most difficult, considering 3D translation, orientation, and size. Many methods assume objects are flat on the ground and the camera has a limited orientation range, simplifying the task to 7DoF. However, this thesis proposes a method for 9DoF pose estimation to handle varying sensor orientations and object placements.

The choice of sensors used by a detector and representation of sensor data has great implications. When using camera images, objects of the same class in the same spatial configuration can have widely different appearances due to lighting and texturing of the objects. Also, missing depth information and lacking prior knowledge about the object size, the estimations of size and position become ambiguous. This is especially true for generic geometric representations such as cuboids, ellipsoids etc., where object-specific scale and aspect ratios are arbitrary as those objects contain no cues related to the size of the objects. In contrast, specific object categories such as sedan-sized cars have strong priors on their approximate dimensions.

Using LiDAR sensors or stereo cameras performing depth-estimation via triangulation, a scene can be represented by a 3D point cloud rather than an image. This purely geometric data potentially allows for a more accurate estimation of size and position estimation as its primary output intrinsically yields point locations in 3D space. This kind of representation reduced to geometric traits also offers a straightforward way to tackle the sim-to-real gap. It is also beneficial for synthetic data generation, because different appearances of objects due to lighting and texturing do not have to be accounted for.

The data domain of geometric scene/object representations also provides some benefits in the context of deep learning. These benefits are: synthetic data can be produced in large quantities, while the so-called sim-to-real gap is easier to be reduced. Also, geometry is often an invariant property, such as rigid structure, metric size or constrained movement in the geometric space due to the rules of physics.

On sensor side, it can be stated that geometry-aware sensing is increasingly yielding high spatial and temporal resolution at low costs. Active sensing LiDAR devices benefit from the market-pull of automotive industry, while stereo cameras can rely on high quality stereo matching schemes relying on modern learned correlation matchers via deep learning. The Middlebury stereo benchmark also reflects this trend, as each year leading methods are still being replaced by new representational advances and methodologies.

A major drawback of using point clouds is the inherent sparsity and irregularity of the data which often leads to failure modes in terms of less discrimination between categories, low recall in presence of occlusions and low precision in presence of clutter. Several approaches have been developed to deal with the peculiarities of point clouds, such as point feature learning, voxel representation and projections. In Chapter 2, these data-specific characteristics and representational aspects are explained in further detail.

The above scientific challenges motivate my research to understand and enhance a bottom-up Hough-based object detection concept to be used for 3D object detection and 9DoF pose estimation in LiDAR point clouds. The bottom-up voting mechanism in the Hough-based detection pipeline has following advantages:

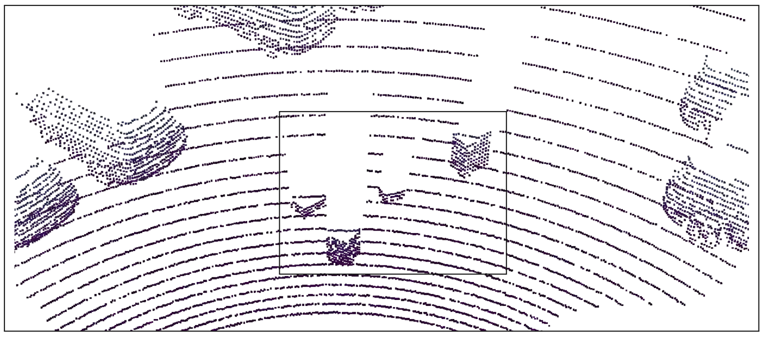
- it can directly exploit the strong spatial-context-encoding ability of the point features created by the backbone (PointNet++)
- based on the scaling of these features it can balance between part-based and object-level representation; a trait which is essential to cope with spatially-varying point densities and missing data caused by self-occlusion, occlusion or corrupted sensory data, and
- the ensemble of votes introduces an added robustness w.r.t. noise and missing data. The Hough-based representation also exhibits a transparent detection mechanism, as votes can be directly visualized. Furthermore it possesses a high representational flexibility, as the voting mechanism could be easily manipulated or reconfigured, such as defining which spatial locations shall be included in the voting mechanism.

1.1 Problem statement and challenges

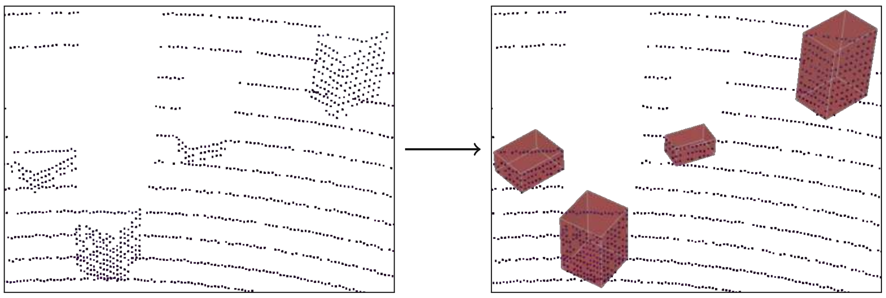
The pursuit of 3D object detection and pose estimation in point clouds pose several challenges, which we have broken down into three categories in the upcoming subsections. In the first subsection, we describe our main objective, which is the detection of and pose estimation of cuboids in point clouds. Next, we shed light on the hurdle of procuring training data for such a task and introduce the idea of generating synthetic data. Finally, we discuss the representational challenge of symmetric objects.

1.1.1 Detector

This thesis addresses the problem of 3D object detection and pose regression of cuboid-like objects in 3D point clouds, as illustrated by Figure 1.2. This should be the first step to a generalized object detection framework capable of detecting many geometric primitives (cuboids, cylinders, ellipsoids, etc.). Cuboid objects, although belonging to a geometric primitive class, are also relevant from a task-oriented point of view. Many man-made functional objects can be described or composed from cuboids (such as containers, crates, construction blocks, furniture).



(a) Scene Overview: A typical LiDAR point cloud within a scene containing a number of different object categories, exhibiting missing data, spatially-varying density, noise and clutter



(b) Left: Point cloud containing objects. Right: Detected objects in point cloud.

Figure 1.2: Illustration of the object detection and pose estimation problem in LiDAR point clouds.

1.1.2 Data

The objective is to detect geometric primitives in varying point cloud densities and estimate the position, size and orientation (which make up the 9 degrees of freedom pose). A large and diverse set of pose annotated data is needed in order to learn representational models such as CNNs or PointNets that task. As of now, no suitable public dataset exists for that objective, so we have to resort to either:

- capturing a large amount of scenes using a LiDAR sensor while accounting for all desired diversities and annotate by hand, or
- programmatically generate the dataset using a simulation framework (Blender [Com18], Nvidia Omniverse [Cor], Unreal Engine [EG]) where the pose information can be generated within the sensor's reference frame and exported.

As the first approach is too labor-intensive and does not scale with an increasing number of classes or learning tasks, we chose the route of creating a synthetic dataset that used for training. A data generation framework was elaborated and used to synthesize thousands of point clouds along with exact annotations. A downside of such a data generation scheme is the introduction of a sim-to-real gap. Synthetic data often exhibits a domain gap in its qualities in comparison to real data, resulting in different performances of the same model when applied to the synthetic and real domains. While methods exist trying to minimize that gap, such as domain randomization [TFR+17] and domain adaptation [THSD17], it remains a hurdle. Because of the sim-to-real gap, we expect our machine learning models to perform poorer on real-world data as they trained exclusively on synthetic data. We attempt to minimize this gap by using purely geometric data and not relying on visual appearances of objects. In order to evaluate the impact of the sim-to-real gap, a validation dataset based on real LiDAR data and a synthetic twin dataset have been created.

1.1.3 Representational challenge due to object symmetry

Because we are dealing with objects that exhibit rotational symmetries, many orientations and sizes might yield equivalent states for a given cuboid in the point cloud. This must be considered in order to avoid penalizing the computed loss in presence of multiple potentially correct states. Even if the model predicts a correct orientation but another possible rotation was the ground truth, a large rotation loss would result incorrectly from that ambiguity and hinders the learning process. This symmetry-driven representational challenge is a well-known effect in the scientific community targeting 3D pose estimation, and numerous methods have been proposed to mitigate it. [XSNF17, TSF17] employ a loss function that measures the offset between each point on the estimated model orientation and the closest point on the ground truth model.

1.2 Contributions

This thesis proposes a purely object-geometry-driven, anchor-free, voting-based, bottom-up, part-based 3D object detection and pose estimation pipeline for point clouds with a focus on cuboids with possible extensions to other geometric classes. This multi-task learning scheme is closely related to VoteNet [QLHG19] and utilizes PointNet++ [QYSG17], an encoder-decoder type network, as a backbone, a Hough voting inspired voting mechanism with vote grouping and aggregation, and finally a prediction head for bounding box estimations. This method, along with the proposed adaptations and a large corpus of synthetic training data, tackles the problems stated in Section 1.1. Key contributions are:

- **Synthetic data generation.** A complex data generation framework - built on top of *Blender* [Com18] and *Blainder* [RNJ21] - has been elaborated and was used to generate a wide variety of scene configurations. This framework allows for generating realistic LiDAR sensor data incorporating a range-dependent noise model along each beam. Data can be generated for arbitrary geometries and with variable LiDAR sensor characteristics. For this thesis, the sensor characteristics of the *Ouster OS-64* [Ous] sensor have been implemented.
- **Extending the proposal module to estimate the 3D orientation of objects**. The model estimates the 9 degrees of freedom (DoF) pose (3 for location, 3 for size, 3 for orientation) for each detected object.
- **Rotational representation**
  - **Enhancing rotational representation to avoid gimbal lock**, which is an ambiguity problem leading to the loss of one degree of freedom, when among the rotational axes (with a predefined order), two axes become parallel.
  - **Introducing a symmetry-aware rotation representation** for cuboids in order to reduce multiple correct orientations to just one. 180° rotations around any of a cuboids local axes results in different orientations while the appearance remains the same due to its symmetric geometry. Similarly, when two or all sides of a cuboid are of the same length, many equivalent orientations exist. Such ambiguities gives rise to conflicting signals for the model to learn, which could lead to a degraded performance. To tackle this problem, we map such equivalent orientations to a canonical orientation using a simple deterministic process.
- **An interactive web-based visualization tool** for point clouds, oriented bounding boxes and votes was elaborated, which can visualize data, percepts and debugging information

1.3 Results preview

In Chapter 4, we evaluate aspects of our proposed cuboid detection and pose estimation method that is purely trained on synthetic data. Since we produced our own validation dataset, that is recorded using a real-world sensor, and a synthetic twin dataset, we can not only evaluate the method on real world data but also highlight differences in performance when evaluating on the synthetic dataset. This difference in performance, the sim-to-real gap, is explored in detail both quantitatively and qualitatively. These results show moderate differences in detection capability between the two validation datasets but also indicates a generalization that allowed for training only on synthetic data and successful real-world inference.
In subsequent experiments we go into detail about the proposed adaptations of representing orientations of objects. We compare the use of Euler angles and the 6D rotation matrix representation and show that orientation estimation improved considerably.
Lastly, we go into detail about the impact of using the proposed symmetry-aware canonical orientation representation which ensures that the model does not receive conflicting signals during training.

1.4 Structure of the thesis, expectations

Chapter 2 explores foundational concepts and reviews relevant existing literature. This encompasses object detection methods, their 3D counterparts, the intricacies of point cloud data and approaches to handling such data.
Chapter 3 is dedicated to the proposed VOCD (Vote-based Oriented Cuboid Detection in point clouds) methodology, presenting the 3D object detection and pose estimation framework with insights into its representation, learning and estimation scheme. Further, details about the synthetic training data generation process are elaborated upon, highlighting the methodologies, tools and parameters used in the creation of the training dataset.
Chapter 4 focuses on the evaluation methodology and conducted experiments, exploring the sim-to-real gap for the model fully trained on synthetic data as well as the differences in performance when using different rotation representations. The outcomes of these experiments are critically assessed, followed by a comprehensive discussion.
In Chapter 5, the thesis concludes by summarizing the key findings and suggesting possible future studies.

CHAPTER 2

Concepts and related work

This chapter gives an overview of relevant concepts along with state-of-the-art methodologies in the domain of 3D object detection. The currently prevailing ideas, representations and methodologies are presented and briefly analyzed in terms of their properties, strengths and weaknesses.

2.1 Object Detection

Object detection is the task of locating instances of objects and identifying the semantic categories they belong to. In the case of 2D object detection, objects must be detected in images by defining bounding boxes around the object instances.
Historically, such methods implemented hand-crafted visual features and by doing so, their capabilities were heavily limited. CNNs eliminated the need for carefully designing visual features by defining those as learnable parameters of a neural network. With the advancements of machine learning techniques, AlexNet [KSH12] was the first method to outperform traditional methods as demonstrated for the image classification task within the ImageNet recognition challenge. Following the success of AlexNet in image classification, R-CNN [GDDM13] and further variants Fast R-CNN [Gir15] and Faster R-CNN [RHGS15] adopted CNNs for object detection, setting the stage for the next generation of object detection methods. Building on these advancements, other architectures were introduced that aimed to further speed up detection while retaining accuracy. YOLO [RDGF16] and SSD [LAE$^+$16] stood out as particularly influential, offering substantial run-time enhancements that paved the way for real-time detection systems.

The focus of this thesis is on 3D object detection. However, as many of these techniques are rooted in the principles of 2D object detection, they remain relevant.

2.2 3D Object Detection

A natural extension of the task of object detection is to detect and locate objects in 3D space. Spatial data can be captured in different formats such as RGB-D images, a fusion of traditional RGB images with an additional channel containing pixel-wise depth information, point clouds or stereo images.

3D object detection methods can be categorized into *region proposal-based* methods and *single shot* methods.

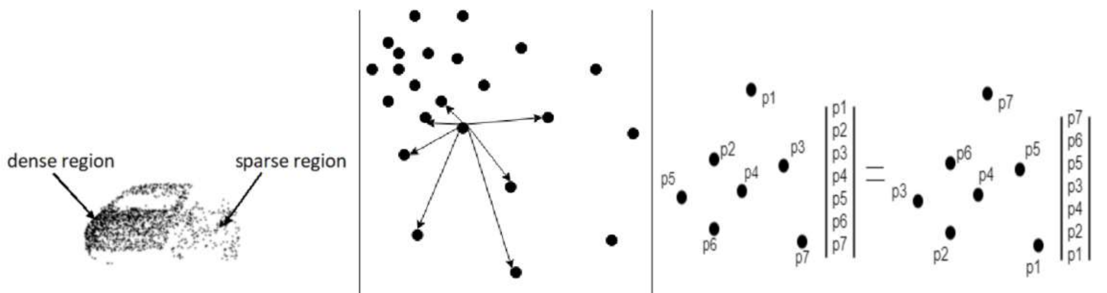
Region Proposal-Based Methods. These approaches initially suggest multiple regions (proposals) that potentially contain objects. Then, for each region, features are extracted and used to determine the properties of each proposal. Different strategies exist to propose regions and aggregate features for such regions:
Segmentation-based methods remove background points by utilizing segmentation methods and then create a large amount proposals on foreground points only.
Frustum-based methods, on the other hand, use existing 2D detection techniques to create candidate 2D regions. For each region, the 3D points in the corresponding 3D frustum are extracted and processed in order to regress properties, such as the object category.

Single Shot Methods. Instead of proposing regions first and then analyzing them, single shot methods integrate both steps into a single stage. The network is designed to simultaneously predict the existence of objects and determine their properties. This streamlining often leads to faster processing times and reduced complexity.

The focus of this thesis is a vote based 3D object detection framework based on VoteNet [QLHG19]. Inspired by generalized Hough transform, VoteNet utilizes PointNet++ [QYSG17], a point-feature learning network, to learn point-wise features and then - for each point - predicts the offset to the respective object center. This effectively brings points, along with their features, closer together and clusters of points are formed. The points of each cluster, along with their features, are then aggregated in order to predict object properties. Considering the stages of VoteNet, this approach bears similarities to region proposal methods but generates those proposals from the point cloud directly using a voting scheme. In the next sections, we discuss the challenges of processing point clouds and how PointNet and PointNet++ tackle those challenges.

2.3 Point cloud data and its challenges

Point clouds are unordered sets and are inherently sparse, because each point in a point cloud is scanned independently and is not placed on a regular grid like pixels in an image. Point clouds are also irregular, as different regions have varying point densities depending on distance, occlusion and sensor characteristics. As a result, the data is unstructured and difficult to process using convolutional neural networks because they expect data that is structured and regular, like images. [BYW$^+$20]
Those challenging characteristics of point clouds are illustrated in Figure 2.1.



(a) Irregular. Sparse and dense regions.

(b) Unstructured. No grid; each point is independent and the distance between neighboring points is not fixed.

(c) Unordered. As a set, point clouds are invariant to permutation.

Figure 2.1: Characteristics of point clouds (Illustration taken from [BYW$^+$20])

Single- and Multi-view projection can be used to transform the point cloud into images so that conventional 2D convolutions can be applied [SMKLM15]. Other methods require point clouds to be discretized into a regular 3D voxel grid. VoxNet [MS15] applies 3D convolutions which - given the inherent sparsity and extra dimension in comparison to 2D images - come with a high computational cost. Authors of Voxel Transformer (VoTr) [MXN$^+$21] tackled the inefficiencies by elaborating a Transformer-based architecture that allows for efficient long-range dependencies.

PointNet [QSMG16] and the subsequent PointNet++ [QYSG17] paper show that features can be learnt for both global and local geometries of point clouds without the need for techniques that introduce spacial quantization artifacts such as voxelization or require unnecessary amounts of resources (e.g. representing empty regions of the observed volume).

2.4 PointNet and PointNet++

Analogously to the 2D representation case, where 2D convolutions capture the local image structure at different scales, volumetric data would also benefit from such a structure-representation approach. However, due to the irregularity and spatially-varying density of point clouds, the transition from 2D to 3D is not straightforward.

2.4.1 PointNet

PointNet [QSMG16] is a neural network architecture, illustrated by Figure 2.2, that processes points in a point cloud, typically retrieved by a LiDAR sensor. Due to the unstructured and unordered nature of point clouds, many researchers resort to discretization of the data, such as quantizing into 3D voxels or projecting onto a 2D planes, so that convolutions or other operations can be applied. In order to avoid quantization artifacts and inflated representation, PointNet abstains from employing such techniques.

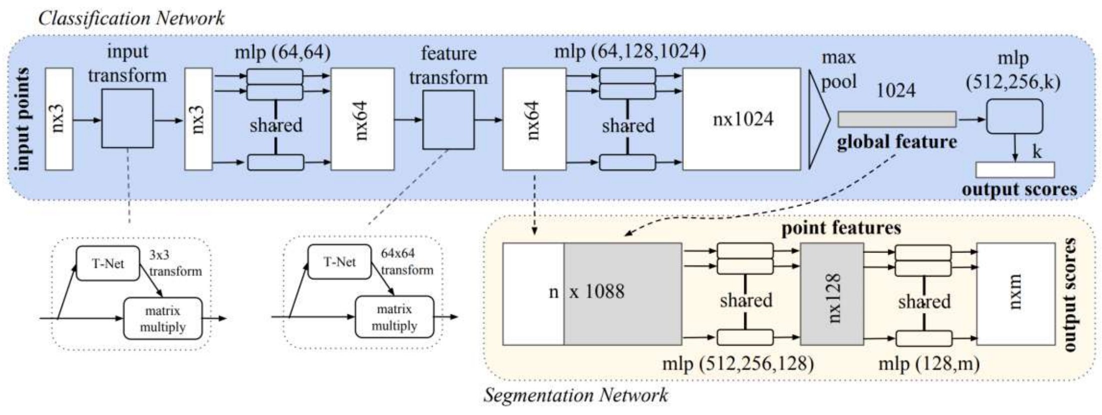


Figure 2.2: Architecture of PointNet (Illustration taken from [QSMG16])

PointNet first transforms the input coordinates into a canonical space by predicting an affine transformation matrix which is applied to the coordinates of the points. This transformation matrix is predicted by a small network referred to as T-Net, which is inspired by [JSZK15]. After this alignment, the coordinates of each point are passed through a shared multi-layer perceptron to generate point-wise features, which are again aligned using a second T-Net. In the next step, max-pool is utilized to aggregate all point features into a global feature vector. The choice such a symmetric function for aggregation facilitates the invariance of input permutation. The resulting global feature vector encodes the whole point cloud and can be used for point cloud classification or other tasks.

2.4.2 PointNet++

PointNet was designed to learn spacial embeddings for each point and aggregate them into a global point cloud embedding that can be used for downstream tasks such as classification or segmentation. The inability to represent the fine structures of local geometries led to an improved method called PointNet++ [QYSG17]. It consists of a number of *set abstraction* (SA) levels where each level consists of a **sampling**, **grouping** and **PointNet** layer. An overview of the network architecture of PointNet++ is given by Figure 2.3.

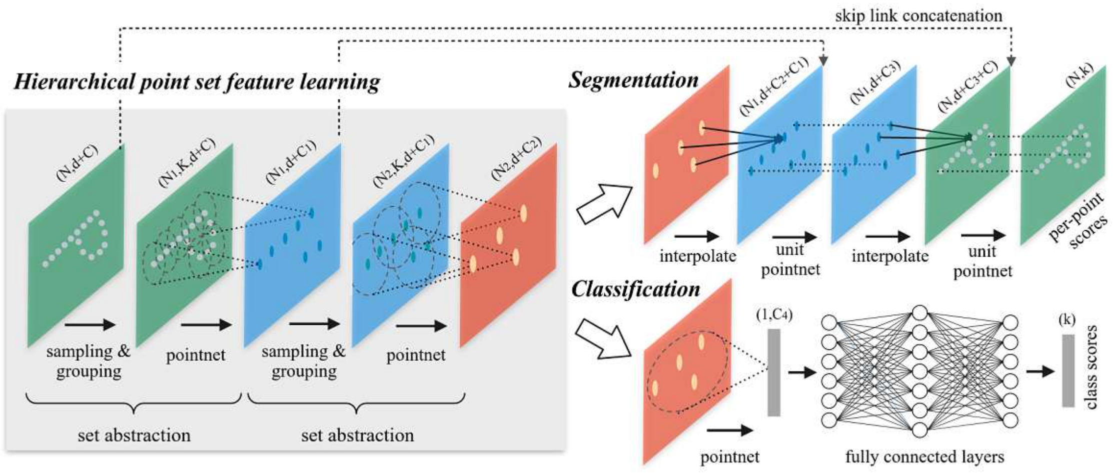


Figure 2.3: Architecture of PointNet++ (Illustration taken from [QYSG17])

The **sampling layer** selects a subset of points from the output of the previous SA (or point cloud if it is the first SA) that become the centroids of the regions for the grouping layer. This is achieved using *iterative farthest point sampling* (FPS) where in each iteration the farthest point that is not yet selected is chosen until a fixed number of points are selected.
In the **grouping layer**, the neighborhood of each centroid is chosen by simply selecting all points inside a given radius, which is referred to as *ball query*.
Then, a reduced version of **PointNet** is used to encode each centroid's local neighborhood, outputting a feature vector for each centroid. By having a fixed scale for each neighborhood and using relative coordinates, feature learning becomes more generalizable. [QYSG17]

2.5 Generalized Hough Transform

The generalized Hough transform [DH72] is a feature extraction technique in computer vision that is able to recognize analytically defined shapes within an image. It tackles the long existing problem of detection using partial observations using a voting mechanism. A typical example would be the task to detect lines in an image. For each pixel in the image, many lines with different parameters could pass through it. This pixel will vote for each of those lines in a parameter space, in which all votes of every pixel are accumulated. Finally, the cells in the parameter space with the most votes would correspond to lines that are detected in the image. Due to the voting mechanism from partial observations, the method is robust to minor deviations of appearance of the instances.

2.6 Deep Hough Voting

Object (foreground) - background separation often implies the following situation: features, such as detected 3D points are often distributed in an uneven manner across the object area or volume. Also, missing detection responses and occlusions are responsible for missing data and objects are often embedded into a background of noise and spurious detections. Therefore the need of robust estimation techniques with respect to object attributes arises. In 2019, inspired by the generalized Hough transform, [QLHG19] proposed VoteNet, end-to-end trainable model working on point clouds using point feature learning and the concept of voting, where local parts of objects cast a vote towards the object center. Those votes can then be clustered and aggregated with minimal contamination of unrelated points, providing a cleaner basis for further processing.

Its bottom-up, parts-based approach introduces a representational advantage by being robust in cluttered scenes with occlusions. It utilizes Pointnet++ [QYSG17] as a backbone network for point feature learning, and aggregates those features in a clever way using a voting mechanism. Then, a prediction head estimates bounding boxes using the aggregated features.

CHAPTER 3

VOCD – Vote-based Oriented Cuboid Detection in point clouds

In this chapter, the proposed methodology for 3D object detection and 9DoF pose estimation for cuboid-like objects in 3D point clouds is presented. Section 3.1 describes the proposed 3D object detection framework, with details about its architecture, backbone network, representations and proposed improvements. Section 3.2 outlines the elaborated data generation pipeline and provides details about the generated dataset.

3.1 3D object detection and pose estimation framework

The proposed 3D detection scheme is fundamentally based on VoteNet [QLHG19], following a bottom-up local geometry-driven estimation scheme of specific object-level attributes such as object center, orientation, and scale. While VoteNet and other previous works [SWL19, CLSJ19, LVC^{+}18] primarily focus on implicitly treating objects on a ground plane with a single "heading" orientation, the presented work targets the estimation of the full set of orientations. Hence, 3D object detection is not limited to objects on a plane, but also applicable for objects lifted or grasped. This extension towards orientation estimation has also an applied impact, as robotic grasping of multiple object categories based on LiDAR or stereo sensors can be conceived based on the methodology. In the following sections, the core components of the representation, learning and estimation scheme are presented.

3.1.1 Architecture

Overview. A point-based feature learning backbone is used for encoding the neighborhoods for a subset of all points. Those so-called seed points are now associated with feature vectors describing local geometries which are useful for the following tasks and are learnt during the training process. A special characteristic of the encoding process is the ability to generate intermediate feature representations which exhibit invariance with respect of missing data, variable density and noise. Using the computed features, each seed point then casts a vote for the center location of the object which it is a part of. This is achieved by passing its feature vector through a multi-layer perceptron, that is shared across all seed points. Those virtual points (votes) have the sole purpose of bringing related points closer together to then be aggregated in the next step. When the votes are clustered, the neighbors' feature vectors are aggregated to form a single feature vector for each cluster. Finally, a shared multi-layer perceptron is used to predict the oriented bounding boxes from the feature vectors of each cluster. As a post-processing step, non-maximum suppression (NMS) is performed to eliminate overlapping bounding boxes. The described architecture is illustrated in Figure 3.1

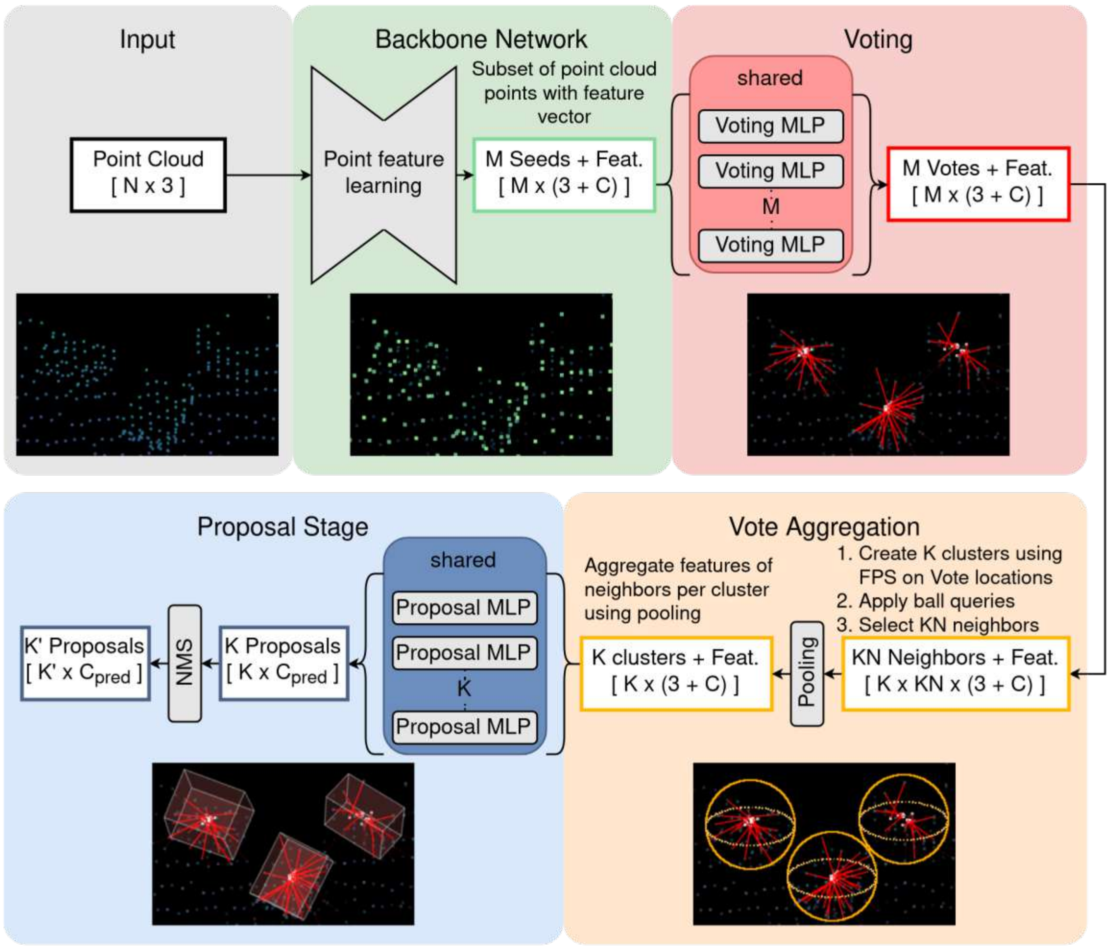

Input
Point Cloud [N x 3]
Backbone Network
Point feature learning
Subset of point cloud points with feature vector
M Seeds + Feat. [M x (3 + C)]
Voting
shared
Voting MLP
Voting MLP
M
Voting MLP
M Votes + Feat. [M x (3 + C)]
Proposal Stage
shared
Proposal MLP
Proposal MLP
K
Proposal MLP
K Proposals [K x Cpred]
NMS
K' Proposals [K' x Cpred]
Vote Aggregation
Aggregate features of neighbors per cluster using pooling
1. Create K clusters using FPS on Vote locations
2. Apply ball queries
3. Select KN neighbors
KN Neighbors + Feat. [K x KN x (3 + C)]
Pooling
K clusters + Feat. [K x (3 + C)]

Figure 3.1: Architecture of proposed object detector (similar to [QLHG19]).
N = #points in point cloud, M = #seed points, C = #features describing local geometry, K = #proposals, KN = #neighboring seed points, selected for aggregation, C_{pred} = #parameters to predict per proposal

Backbone. PointNet++ is used as a backbone network that outputs a fixed amount of sampled points (seed points) with feature vectors. By the nature of the backbone network, each point's feature vector has a limited receptive field, resulting in encoding more local geometries than larger structures. This is useful, as even local geometries have enough clues to approximately point towards object centers.
As discussed in Section 2.4, the strength of PointNet++ comes from the ability to encode local shapes of multiple scales and therefore being able to represent features in varying densities and/or sizes. This property provides a significant advantage as we deal with differently sized cuboids in various scene configurations.

Vote Generation. Using the features of the seed points, votes for object centers are generated using a multi-layer perceptron that is shared across all seed points and its features. The goal is not to perfectly regress the object center location but to bring the seed points closer together by creating those virtual points we call votes.

Vote Aggregation. In this step votes and their features are aggregated in a scheme similar to PointNet++. A subset of votes are selected as cluster centers using iterative farthest point sampling (FPS). Then, ball queries select the neighborhood of each cluster center and the features are subsequently aggregated. This is done by first transforming the feature vectors using a shared multi-layer perceptron and then applying a max-pool operation resulting in one feature vector per cluster.

Proposal Generation. In this step, the feature vector of each cluster is passed through a shared multi-layer perceptron to regress the following target values to form proposals:

- objectness (binary value indicating that the proposal is an object we want to detect)
- category
- 3D object center (regressed as offset from cluster center)
- 3D size
- 3D orientation

Non-Maximum Suppression. In the previous step, a considerable number of overlapping proposed bounding boxes are generated. This happens due to the selection of a predetermined number of vote locations as centers for both vote aggregation and proposal generation, potentially causing nearby vote locations to be chosen for creating proposals. Non-Maximum Suppression (NMS) helps address the issue of overlapping bounding boxes by reducing the number of overlapping proposals and retaining only the most accurate ones. It works by following these steps:

1. Rank the proposed bounding boxes based on their objectness scores.
2. Select the bounding box with the highest score and add it to the final set of detected boxes.

3. Remove any bounding boxes that have a high overlap using Intersection-over-Union (IOU) metric with the chosen box.

4. Repeat steps 2 and 3 until all remaining boxes have been processed.

3.1.2 Orientation representation

Popular examples of orientation representations include *Euler angles*, *rotation matrix* and *quaternion*. None of these representations can inherently handle rotational symmetries, leading to different values describing the orientation of objects with the same appearance. This is especially true when dealing with symmetric objects in point clouds, since no texture is present to break the symmetry.

We briefly describe Euler angles and their limitations, then introduce the alternative representations we utilize: the rotation matrix and a 6D variant of it. Further, a simple method to handle the symmetry of cuboids is described.

Euler Angles

A common choice is to define the orientation utilizing Euler angles, represented by three values for a 3D rotation: yaw, pitch and roll. Those describe consecutive rotations about the 3 major axes. They are intuitive and easy to understand and allow to define a rotation with just 3 parameters. A drawback from using this representation is the phenomenon called *gimbal lock*, which causes the loss of a degree of freedom. This occurs when 2 rotational axes align in parallel, effectively causing one of the axes to become "locked". In this scenario, there are infinite ways to parameterize a single orientation.

Rotation Matrix

In a 3D context, a rotation matrix is parameterized with 9 parameters in the following form:

$$R = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{m}_1 & \mathbf{m}_2 & \mathbf{m}_3 \\ | & | & | \end{bmatrix} \tag{3.1}$$

In addition to the 6 parameters more needed to be estimated, the orthogonality constraint of rotation matrices, where $MM^T = M^TM = 1$ and $det(M) = 1$, has to be enforced in a post processing step. In contrast to Euler angles, the gimbal lock problem is non-existent for rotation matrices.

6D Rotation Matrix

Following [ZBL+20], we choose to use the Gram-Schmidt process to both reduce the number of estimated parameters needed for the construction of the 3×3 rotation matrix to a total of 6 parameters and enforce orthogonality. For this method, the only the first 2 columns of a rotation matrix are regressed using the model. The first predicted column a_1 is normalized, resulting in b_1. The column b_2 is constructed by subtracting the projection of a_2 onto b_1 from itself, ensuring b_1 and b_2 are orthogonal. Since b_1 and b_2 are orthogonal, the third column b_3 of the rotation matrix can be computed as the cross product of b_1 and b_2. Finally, the resulting rotation matrix composed of b_1, b_2 and b_3 is used for loss computation. The Gram-Schmidt process can be described by Equation 3.2 and Equation 3.3, which are taken from [HAAH22].

$$f_{GS}\left(\begin{bmatrix} | & | \\ a_1 & a_2 \\ | & | \end{bmatrix}\right) = \begin{bmatrix} | & | & | \\ b_1 & b_2 & b_3 \\ | & | & | \end{bmatrix} \tag{3.2}$$

$$\begin{aligned} b_1 &= \frac{a_1}{\|a_1\|} \\ b_2 &= \frac{u_2}{\|u_2\|}, u_2 = a_2 - (b_1 \cdot a_2)\, b_1 \\ b_3 &= b_1 \times b_2 \end{aligned} \tag{3.3}$$

Experiments by [ZBL+20] showed improved performance when employing the 6 parameters followed by the application of the Gram-Schmidt process compared to the approach of directly regressing all the values of a rotation matrix.

Canonical Orientation for Cuboids

Rotational symmetric objects, by their very nature, exhibit multiple orientations that result in the same appearance. This is especially noticeable in 3D representations of objects that don't have any textures. Due to this, conflicting signals are attempted to be learnt by the model, which in turn hinders the learning process. Since none of the previously mentioned rotation representations can inherently handle object symmetries, we have to resort to a method to map different but equivalent orientations to a single, canonical one so that there is a predictable one-to-one mapping between appearance and orientation. When a texture-less cuboid is rotated, there exist three additional orientations that would yield the same appearance. These orientations are simply 180° turns around each local axis of the cuboid. In order to map those four orientations to a canonical one in a deterministic way, we calculate the geodesic distance from each possible orientation to the identity orientation and select the one with the smallest distance to it. The exact proposed procedure to obtain a canonical orientation for cuboids is detailed in Algorithm 3.1.

Algorithm 3.1: Canonical Orientation Finding Algorithm

```
Result: Canonical orientation of a cuboid
Function FindCanonicalOrientation(cuboid)
    identityOrientation ← GetIdentityOrientation();
    equivalentOrientations ← GetEquivalentOrientations(cuboid);
    minDistance ← $\infty$;
    canonicalOrientation ← NULL;
    foreach orientation in equivalentOrientations do
        distance ← CalculateGeodesicDistance(orientation,
         identityOrientation);
        if distance < minDistance then
            minDistance ← distance;
            canonicalOrientation ← orientation;
        end
    end
    return canonicalOrientation;
Function GetEquivalentOrientations(cuboid)
    initialOrientation ← cuboid.orientation;
    equivalentOrientations ← [initialOrientation];
    foreach axis in ['x', 'y', 'z'] do
        rotatedCuboid ← RotateCuboidAroundLocalAxis(cuboid, axis, 180);
        equivalentOrientations.append(rotatedCuboid.orientation);
    end
    if AreSideLengthsEqual(cuboid.size.x, cuboid.size.y) then
        rotatedCuboid ← RotateCuboidAroundLocalAxis(cuboid, 'z', 90);
        equivalentOrientations.append(rotatedCuboid.orientation);
        rotatedCuboid ← RotateCuboidAroundLocalAxis(cuboid, 'z', -90);
        equivalentOrientations.append(rotatedCuboid.orientation);
    end
    if AreSideLengthsEqual(cuboid.size.y, cuboid.size.z) then
        rotatedCuboid ← RotateCuboidAroundLocalAxis(cuboid, 'x', 90);
        equivalentOrientations.append(rotatedCuboid.orientation);
        rotatedCuboid ← RotateCuboidAroundLocalAxis(cuboid, 'x', -90);
        equivalentOrientations.append(rotatedCuboid.orientation);
    end
    if AreSideLengthsEqual(cuboid.size.z, cuboid.size.x) then
        rotatedCuboid ← RotateCuboidAroundLocalAxis(cuboid, 'y', 90);
        equivalentOrientations.append(rotatedCuboid.orientation);
        rotatedCuboid ← RotateCuboidAroundLocalAxis(cuboid, 'y', -90);
        equivalentOrientations.append(rotatedCuboid.orientation);
    end
    return equivalentOrientations;
Function AreSideLengthsEqual(a, b)
    return $|a - b| < \epsilon$;
```

3.1.3 Loss

The overall loss $\mathcal{L}$ is calculated by combining the vote loss $\mathcal{L}_{\text{vote}}$ and bounding box loss $\mathcal{L}_{\text{bbox}}$, which are scaled using weighting factors α and β to ensure they are on the same scale:

$$\mathcal{L} = \alpha\mathcal{L}_{vote} + \beta\mathcal{L}_{bbox} \tag{3.4}$$

For every seed point that is part of an object, the regressed object center **vote locations** are supervised using the L1 Loss, introducing the loss component $\mathcal{L}_{vote}$.

The bounding box predictions are supervised using a composite loss function, where each component is individually dedicated to objectness, object center, size and rotation. The weighted sum of those losses make up the bounding box loss $\mathcal{L}_{\text{bbox}}$.

$$\mathcal{L}_{\text{bbox}} = \lambda_1\mathcal{L}_{obj} + \lambda_2\mathcal{L}_{xyz} + \lambda_3\mathcal{L}_{dim} + \lambda_4\mathcal{L}_{rot} \tag{3.5}$$

$\mathcal{L}_{\text{obj}}$ is the loss attributed to the binary objectness classification of a proposal using the Cross-entropy loss. The loss components $\mathcal{L}_{xyz}$, $\mathcal{L}_{dim}$ and $\mathcal{L}_{rot}$ are L1 norms imposed on the differences between ground truth and prediction of the center, the dimensions, and the rotation of the proposed bounding box. The weights $\lambda_1 ... \lambda_4$ are chosen to ensure similar scalings.

3.2 Training Data Generation

The dataset used for training was generated by a custom data generation framework using the 3D software *Blender* [Com18] and its scripting capabilities. LiDAR scan simulation is achieved by the *Blainder* plugin [RNJ21], where a custom extension supporting the characteristics of the *Ouster OS1-64* [Ous] sensor was implemented. Table 3.1 lists all parameters along with value ranges that are randomly varied throughout the generation process in order to cover a wide variety of object-, scene- and view configurations.
Every scene is made up of objects that can be either cuboids, spheroids or tetrahedrons. The three size parameters are defined in a special way to avoid ambiguities for different size orderings and rotation of an object. For example, when the sizes of the first and second dimension are swapped for a cuboid, a texture-less object appears to have rotated by 90° around the third dimension. To tackle this, we set the size of an object in a descending fashion, so that the first dimension has the highest value, the second the second largest and the third dimension the lowest value of the three.
The positions of objects are randomized inside 10m x 10m area and the orientation of each object is randomized in all degrees of freedom. The sensor's location is randomized by utilizing spherical coordinates, where random values for azimuth, elevation and radius are used.

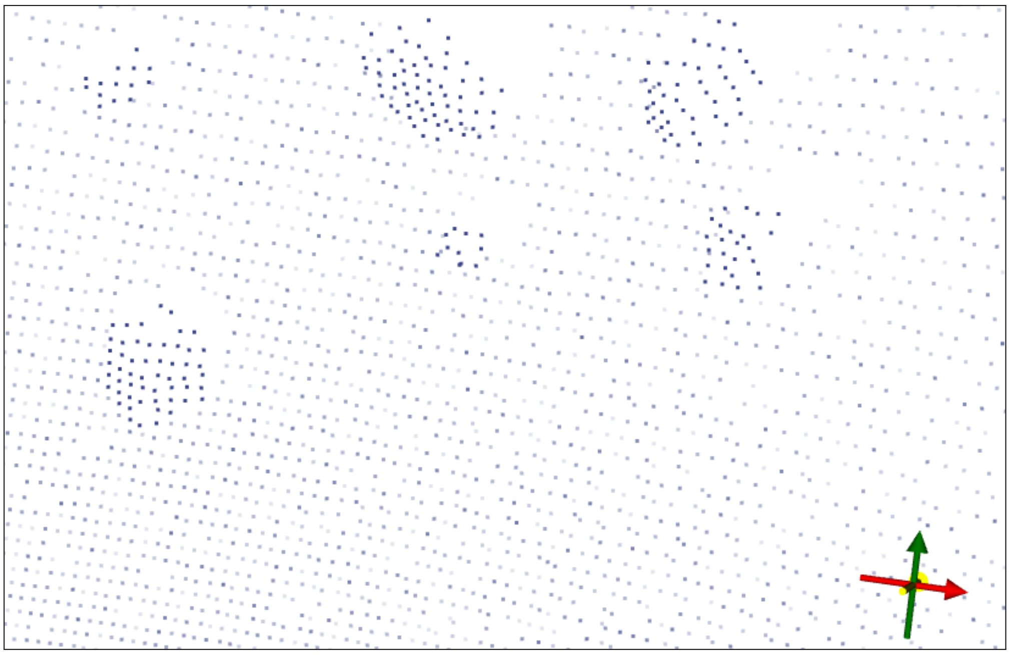
	Parameter	Possible values
scene	# of objects in scene	$[5, 20] \cap \mathbb{Z}$
object	object category	{ cuboid (60%), spheroid (20%), tetrahedron (20%) }
	size	
	X, Y, Z $(X \geq Y \geq Z)$	$[0.25, 4]$
	position	
	X, Y	$[-10, 10]$
	Z	chosen s.t. object is touching the floor
	orientation	3D isotropic rotation
sensor	sensor target	
	X, Y	0
	Z	$[0, 2.5]$
	sensor location	
	azimuth (degrees)	$[0, 360]$
	elevation (degrees)	$[10, 60]$
	radius	$[3, 15]$

Table 3.1: Dataset generation parameters. $[A, B]$ designates an interval of possible values.

For every scene, 8 simulated LiDAR scans were performed using a randomized sensor location and target as defined in Table 3.1. In total, over 30k point clouds along with annotations were generated, forming our synthetic dataset. Figure 3.2, Figure 3.3 and Figure 3.4 show examples of generated point clouds along with their annotations.

Our data generation pipeline outputs the generated point clouds with the following annotations:

- For every object visible by the sensor:
  - category
  - 3DoF position (relative to sensor)
  - 3DoF size
  - 3DoF orientation (relative to sensor)
- For every point in the point cloud:
  - object assignment (used for vote location & objectness of seed points)


(a) Example 1: Point cloud

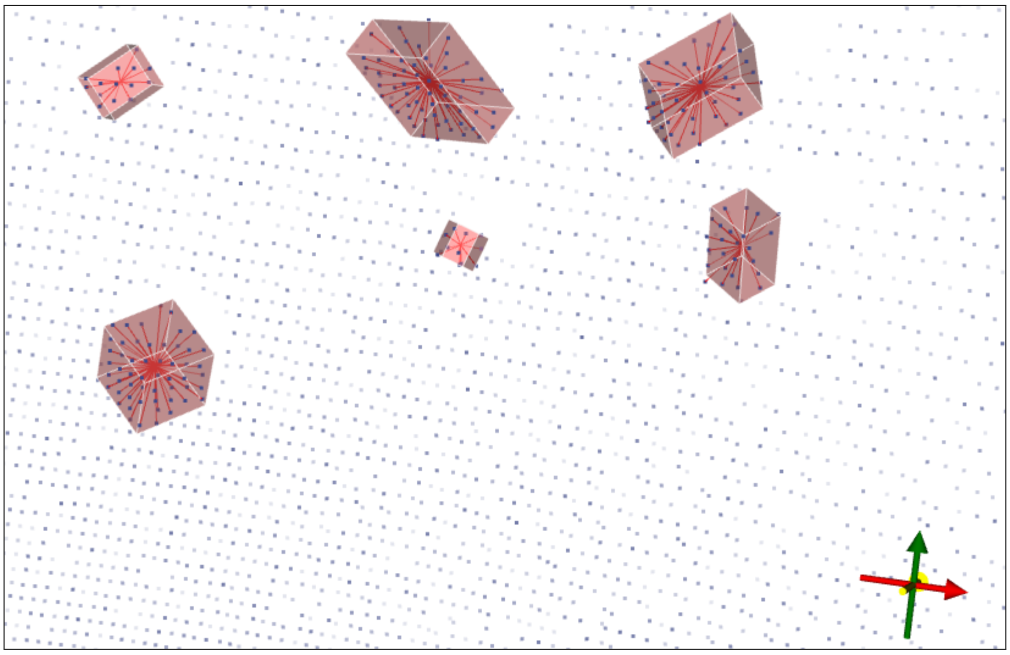
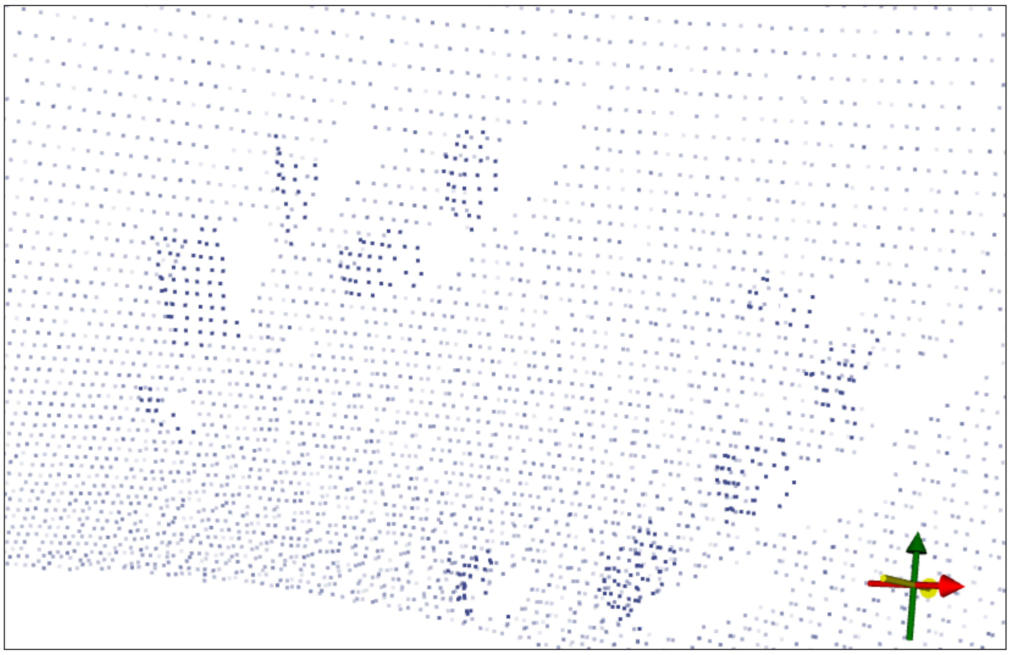

(b) Example 1: Point cloud + annotations

Figure 3.2: Examples of synthetic point clouds along with their annotations. The bounding box color indicates the object category (red: *cuboid*, green: *tetrahedron*, blue: *spheroid*). Object assignments of points are illustrated as red lines to the object centers.


(a) Example 2: Point cloud

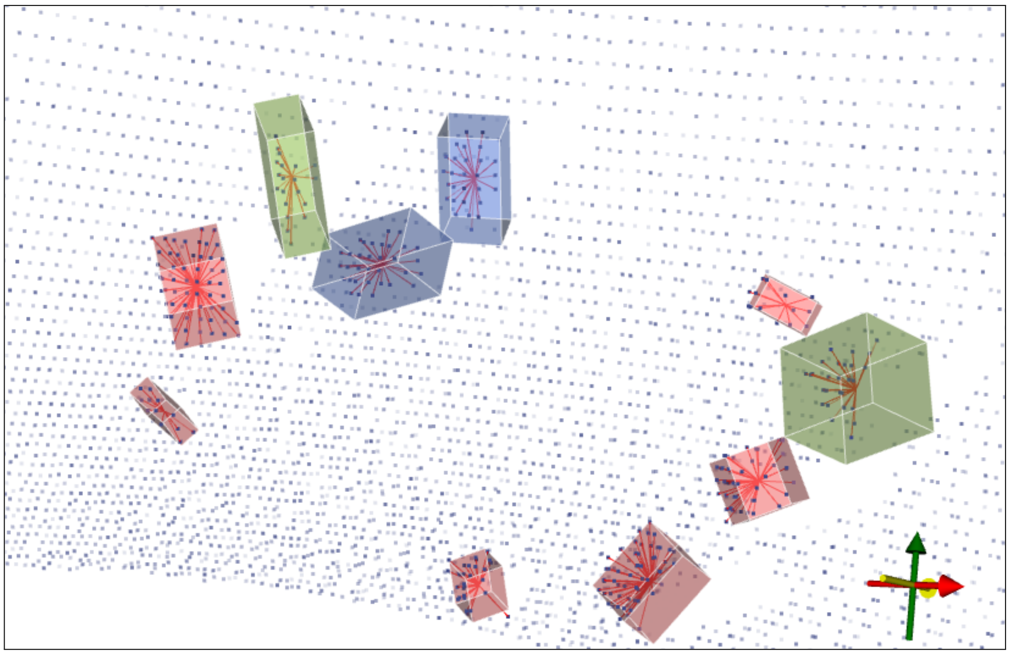
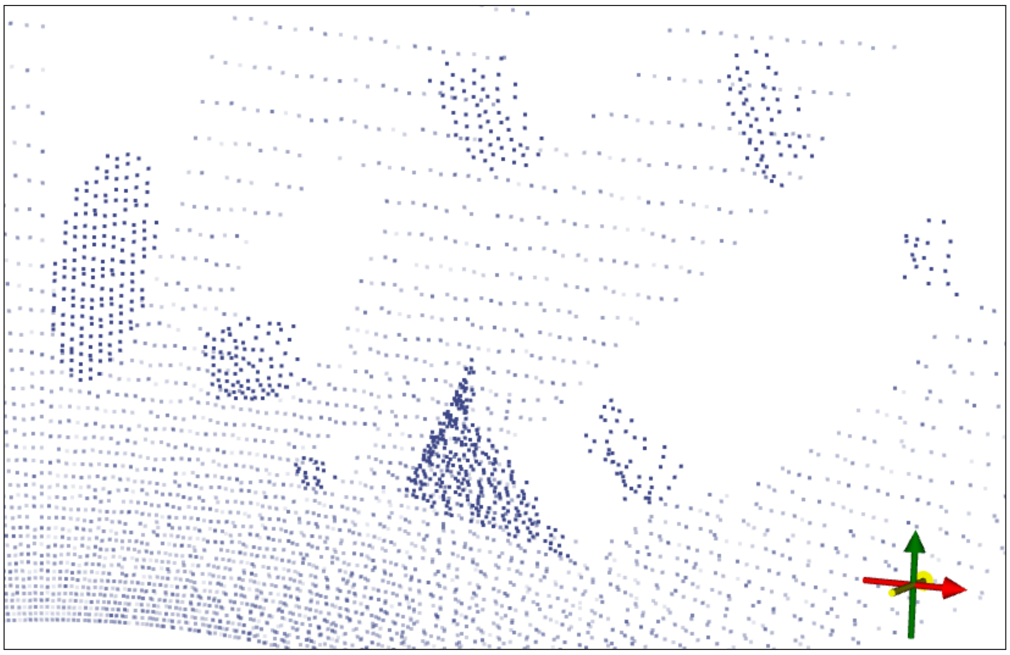
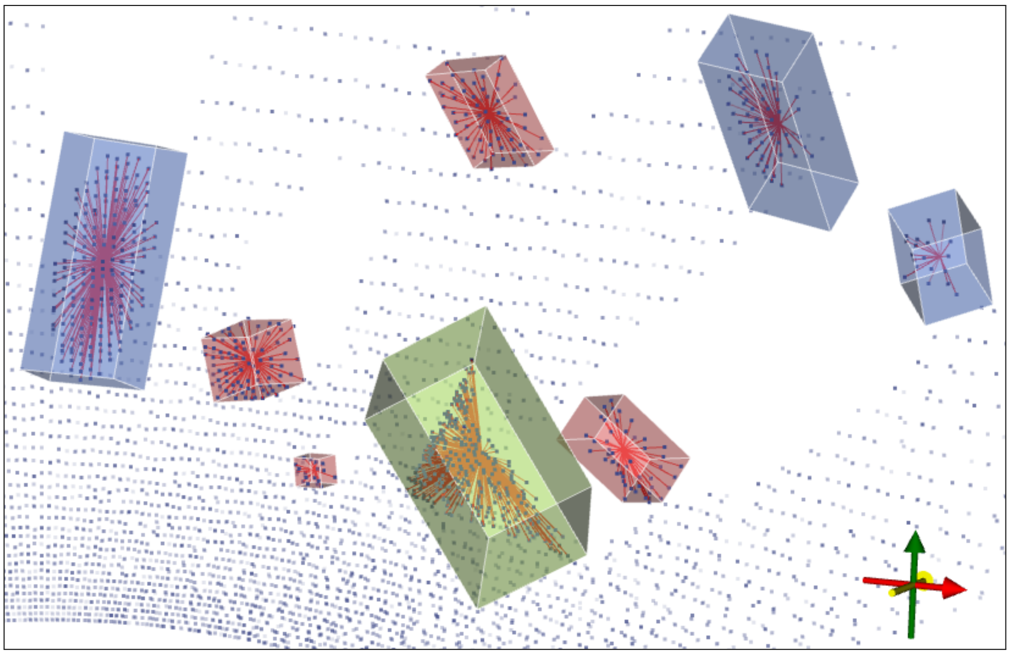

(b) Example 2: Point cloud + annotations

Figure 3.3: Examples of synthetic point clouds along with their annotations. The bounding box color indicates the object category (red: *cuboid*, green: *tetrahedron*, blue: *spheroid*). Object assignments of points are illustrated as red lines to the object centers.



(a) Example 3: Point cloud



(b) Example 3: Point cloud + annotations

Figure 3.4: Examples of synthetic point clouds along with their annotations. The bounding box color indicates the object category (red: *cuboid*, green: *tetrahedron*, blue: *spheroid*). Object assignments of points are illustrated as red lines to the object centers.

3.3 Summary

In this chapter, we introduced VOCD, the proposed 3D object detection and 9DoF pose estimation framework with all its details. Additionally, a data generation framework was described, which was leveraged to craft a diverse and comprehensive collection of 3D point clouds with annotations containing fully-oriented cuboids and clutter objects of varying sizes. The embedded point clouds exhibit diverse densities due to the randomized location and orientation of the LiDAR sensor. Such a rich and varied dataset should provide a solid foundation for the detection method to accurately identify oriented cuboids within point clouds.

CHAPTER 4

Results and discussion

The previously described concepts and enhancements of the Deep Hough Voting method are subjects that we intend to qualitatively and quantitatively analyze. In this endeavor, the following research questions will be addressed:

Will a model trained solely on synthetic data reflect its performance on real data? Is there any degradation in performance? Will the use of rotation matrices as rotation representation improve the performance compared to Euler angles? Does the utilization of the canonical orientation representation have a positive impact of on performance by eliminating conflicting signals for the model to learn?

4.1 Evaluation Methodology

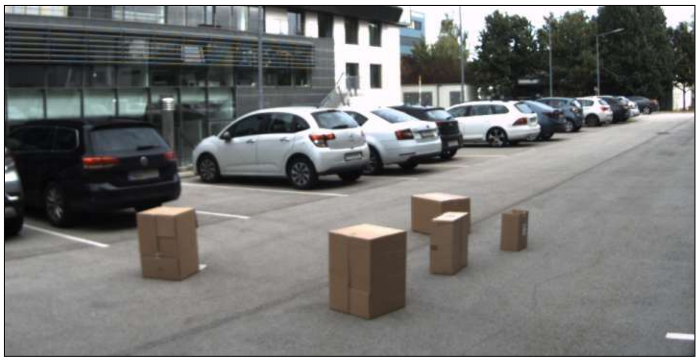
To investigate the performance difference of the model on real versus synthetic data, we create two validation datasets. The first of these is based on real LiDAR data, while the second, a synthetic dataset, has been designed to mirror the same scenes but employs the use of a simulated version of the same LiDAR sensor.

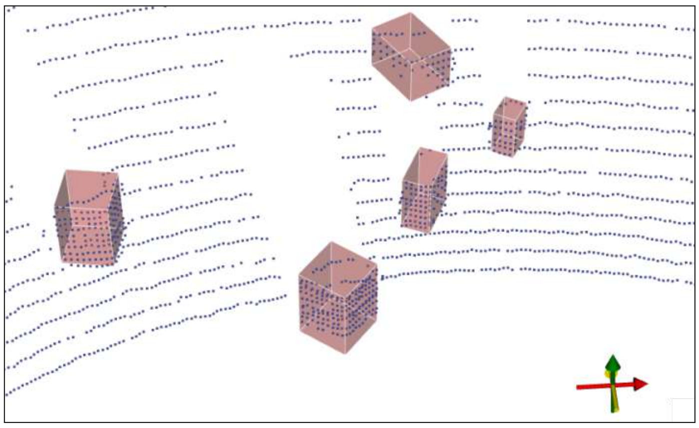
4.1.1 Validation Datasets

Utilizing an *Ouster OS-64* LiDAR sensor [Ous] we successfully recorded 27 different scenes, each of them encompassing between 1 to 6 cuboids. Annotations were manually added to those point clouds using the 3D software *Blender* [Com18], resulting in the real-world validation dataset DS_{real}.

For the synthetic twin dataset DS_{synth}, empty scenes were populated with the annotated boxes. For a fair comparison, the floor and surrounding objects - such as cars and light poles - were also modeled using the real point clouds as a reference. Finally, the scenes were scanned using a simulated version of the same LiDAR sensor, resulting in the

synthetic point cloud and thus synthetic validation dataset.
In Figure 4.1 and Figure 4.2, examples of point clouds with their annotations from both DS_{real} and DS_{synth} are compared.


(a) Example 1: LiDAR sensor's Perspective


(b) Example 1: Real point cloud + GT objects

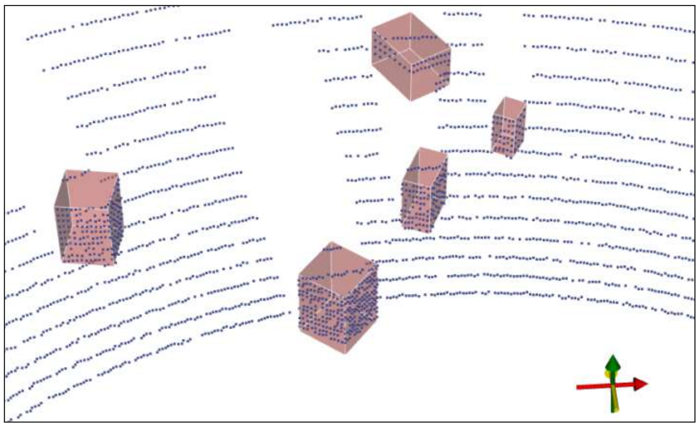
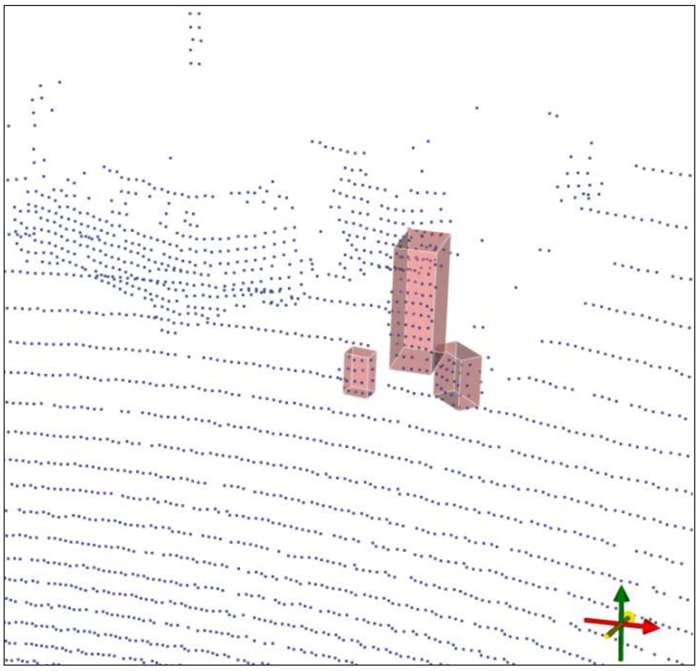

(c) Example 1: Synthetic point cloud + GT objects

Figure 4.1: Examples point clouds from the recorded validation dataset and its synthetic counterparts.


(a) Example 2: Real Point cloud + GT objects

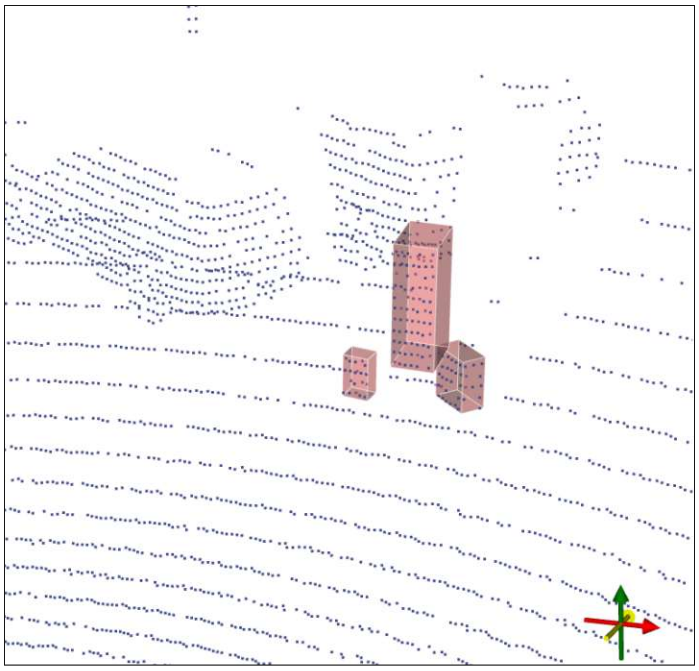

(b) Example 2: Synthetic point cloud + GT objects

Figure 4.2: Examples of point clouds from the recorded validation dataset and its synthetic counterparts.

The synthetic point clouds exhibit point densities and noise patterns that closely mirror their real counterparts. In synthetic point cloud, typically transparent elements like car windows present as opaque. Objects in both datasets feature roughly equivalent point counts on their exteriors and show comparable local geometric configurations.

4.1.2 Evaluation Metrics

Evaluation metrics are essential in assessing the performance of models. In general, they provide measurable values, enabling direct comparisons between different models. In our case, we compare the performance of the same model, trained solely on synthetic data, on the real validation dataset and the synthetic counterpart in order to investigate the difference in performance, also known as the sim-to-real gap.

Average Precision

The Average Precision (AP) metric is used as the key performance metric for our quantitative evaluation. It is well established in the domain of object detection and reflects the detection capability of a detection model, making it suitable for our use case. We compute the AP by approximating the area under the Precision-Recall curve using the following formula:

$$AP = \sum_{k=1}^{n-1} (R_k - R_{k-1}) * \max_{k' \geq k} P_{k'} \tag{4.1}$$

where n denotes the number of distinct recall values derived from adjusting the confidence threshold of the model, R is the resulting ordered collection of distinct recall values (referred to recall levels), P is the collection of precision values corresponding to the recall levels R. Both R and P are padded with 0 at the beginning and 1 by the end.

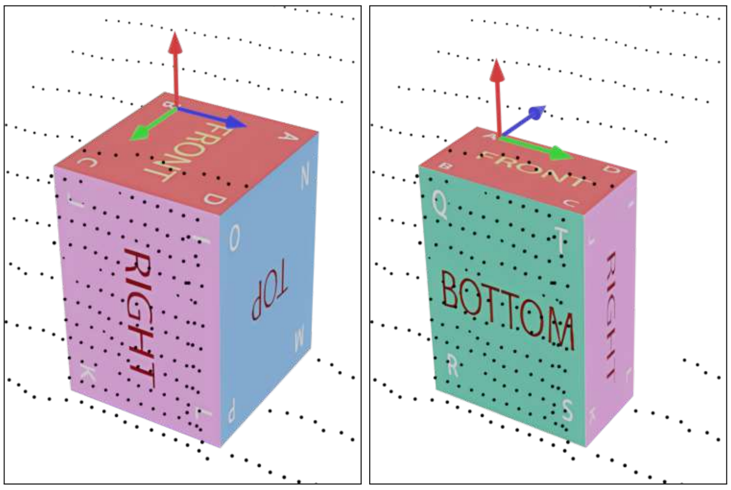
To compute Precision and Recall metrics, associations of predicted bounding boxes to ground truth bounding boxes are needed. This is achieved by the Intersection-Over-Union (IOU) metric, which is a measure of overlap between 2 areas or - in our case - volumes. Going from the most confident to least confident predictions, we compute the IOU with every non-associated ground truth bounding box. We select the ground truth bounding box with the highest IOU and if the IOU value is above a certain threshold, we associate the predicted and ground truth bounding box. The IOU metric is calculated using the following formula:

$$IOU = \frac{\text{Intersection volume}}{\text{Union volume}} \tag{4.2}$$

We report AP for specific IOU thresholds, denoted by a subscript (e.g. AP_{25} for IOU=0.25). A higher IOU threshold would require a better fit of the predicted bounding boxes to the ground truth in order to be associated, impacting both precision and recall values. By doing so, we jointly evaluate rotation, translation and size along with the detection capability.

Orientation Error

Evaluating the estimated orientations in our setup requires special consideration. We define a cuboids side lengths in descending order, which means that the longest side of a cuboid is along its local X-Axis, the second longest is along its Y-Axis and the shortest is along its Z-Axis. Consequently, the orientation of the cuboid is coupled to the dimensions of the cuboid, leading to a flawed orientation error when dimensions are wrongly estimated. Thus, before employing orientation error metrics, it's crucial to address and remedy this concern.



(a) Ground truth bounding box (b) Predicted bounding box

Figure 4.3: An example where a difference in side length estimation can lead to a drastically different predicted orientation, as indicated by the gizmos above the objects.

Using a simple scenario (as illustrated in Figure 4.3) can help explain the issue:

Consider, as an example, an ideal cuboid prediction, perfectly matching the ground truth annotation, as depicted by Figure 4.3a. The Angular Error for such a prediction would be zero. However, if the neural network - due to missing data or estimation errors - estimates even a single dimension of the cuboid shorter than another shorter side length, the predicted orientation would flip by 90°, as shown by Figure 4.3b. This stems from the network's tendency to generate cuboid orientations based on the descending order of its predicted dimensions. As this behavior results in a significant Angular Error of 90° for our case, any naive use of orientation error metrics is flawed.
Given that the network also outputs the cuboid's dimensions in a descending order, incorrectly predicted side lengths cannot be identified with certainty. We therefore apply all orientation error metrics on every of the 90° flipped versions of the predicted

orientations, and report the smallest one. For a symmetry-aware evaluation, we compare those orientations to all equivalent ground truth orientiations. For cuboids, these equivalent orientations are rotations of 180° around each local axis.

Keeping previous considerations in mind, we use the following metrics to evaluate the orientation of the proposals. We adopt the popular Average Orientation Similarity (AOS) metric, first proposed by [GLU12]. It is defined as:

$$AOS = \sum_{k=1}^{n-1} (R_k - R_{k-1}) * \max_{k' \geq k} s(R_{k'}) \tag{4.3}$$

where $s(r)$ is the orientation similarity at recall level r, calculated as a normalization of the cosine similarity:

$$s(r) = \frac{1}{|\mathcal{D}(r)|} \sum_{i \in \mathcal{D}(r)} \frac{1 + \cos \Delta_\theta^{(i)}}{2} \delta_i \tag{4.4}$$

Here, $\mathcal{D}(r)$ are the detections at recall level r and $\Delta_\theta^{(i)}$ is the angular error between the ground truth and predicted orientation of the detection i, which is calculated using the geodesic distance. For detections matched to a ground truth bounding box $\delta_i = 1$, otherwise $\delta_i = 0$. AOS has a range of $[0, 1]$, but as indicated by Equation 4.4, its upper limit is defined by the AP. This means that perfect orientation estimations would result in an AOS score equal to AP.

Additionally we adopt Average Angular Error (AAE), a metric building upon AOS and popularized by [KLR18]. It is a detection normalized measure of average orientation error and is defined as:

$$AAE = \cos^{-1}\left(2\frac{AOS}{AP} - 1\right) \tag{4.5}$$

AAE returns the average angular error in radians in the range of $[0, \pi]$. For easier interpretability of the results we report AAE in degrees with a range of $[0, 180]$.

Similarily to AP, AOS and AAE are calculated for specific IOU thresholds, denoted by a subscript (e.g. AOS$_{25}$, AAE$_{25}$ for IOU=0.25)

4.2 Experiment 1: Exploring the sim-to-real gap

For this experiment, we employ the synthetically trained VOCD model and investigate the sim-to real gap by comparing the performance and prediction characteristics on the real test set and the mimicked synthetic dataset. In this configuration, three Euler angles (yaw, pitch, roll) are regressed in addition to the other 6DoF (location + size), making up the total 9DoF pose estimation.

4.2.1 Quantitative Results

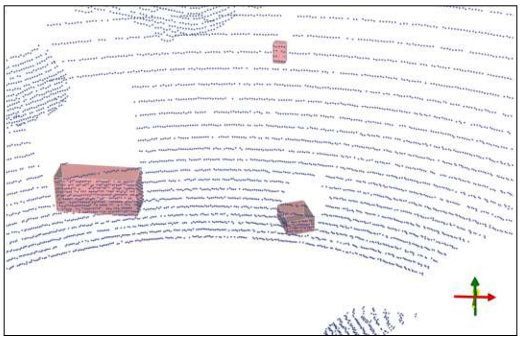
Validation Dataset	AP_{25}	AP_{50}	AOS_{25}	AOS_{50}	AAE_{25}	AAE_{50}
DS_{synth}	**82.2**	**62.8**	**75.6**	**57.6**	**32.9**	**33.4**
DS_{real}	81.4	51.6	74.1	46.3	34.7	37.1

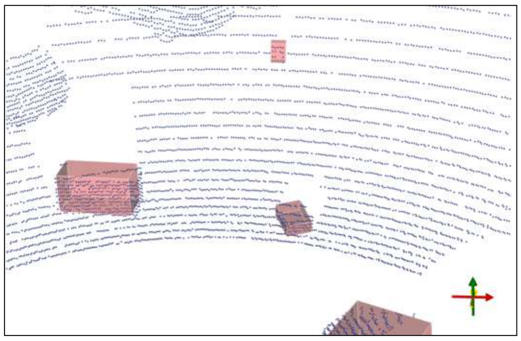
Table 4.1: Performance of the synthetically trained model on the synthetic and real validation datasets.

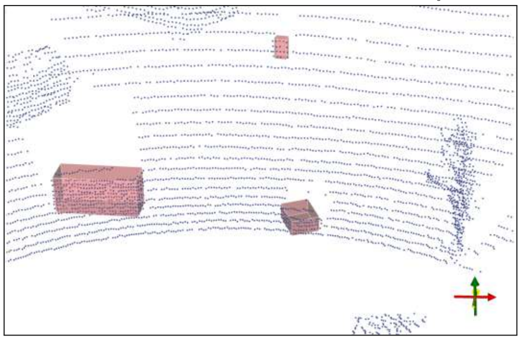
Unsurprisingly, the results in Table 4.1 show a superior performance of the model on DS_{synth} compared to DS_{real}. The slight decline in AP_{25} and moderate decline in AP_{50} on DS_{real} underscores the inherent differences between the two datasets and seemingly unavoidable sim-to-real gap.

4.2.2 Qualitative Results

Figure 4.4 and Figure 4.5 show side-by-side comparisons of predicted objects on the both DS_{synth} and DS_{real}.


(a) Example 1: Ground truth DS_{synth}


(b) Example 1: Predictions on DS_{synth}


(c) Example 1: Ground truth DS_{real}

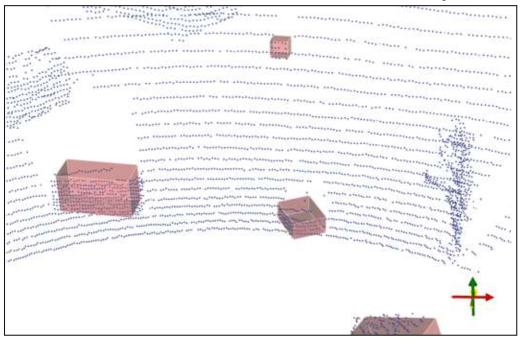
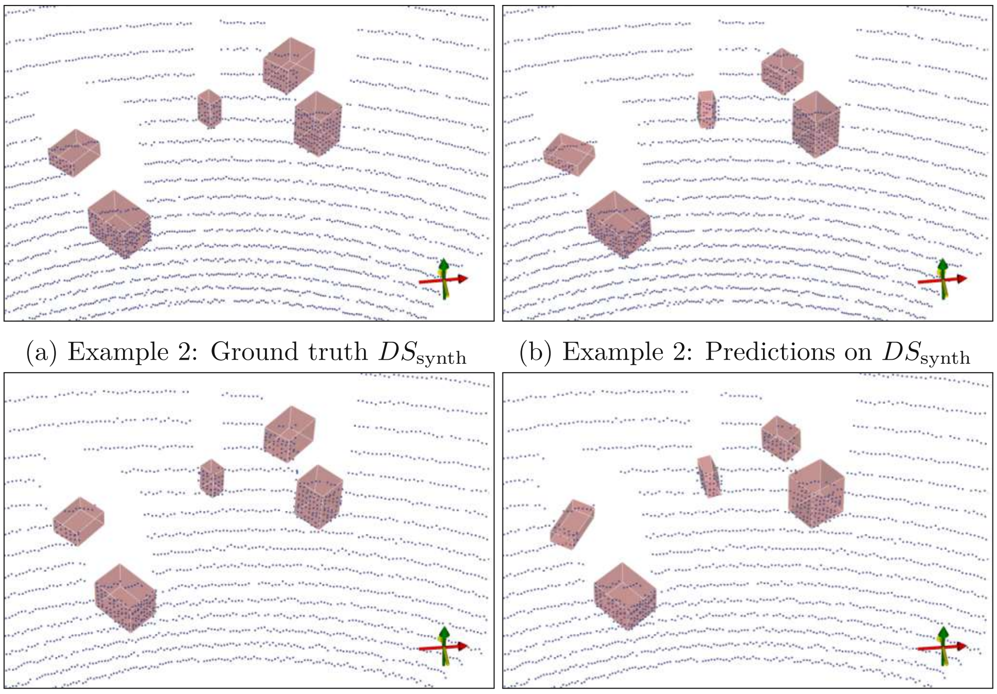

(d) Example 1: Predictions on DS_{real}

Figure 4.4: Comparison of predictions on DS_{synth} and DS_{real}


(a) Example 2: Ground truth DS_{synth} (b) Example 2: Predictions on DS_{synth}

(c) Example 2: Ground truth DS_{real} (d) Example 2: Predictions on DS_{real}

Figure 4.5: Comparison of predictions on DS_{synth} and DS_{real}

In Figure 4.6 we showcase the voting mechanism on the same scene for both validation datasets side-by-side. We can observe an overall robustness of the voting mechanism voting towards the centers of objects but also a slight degradation in center vote accuracy on DS_{real} compared to DS_{synth}.

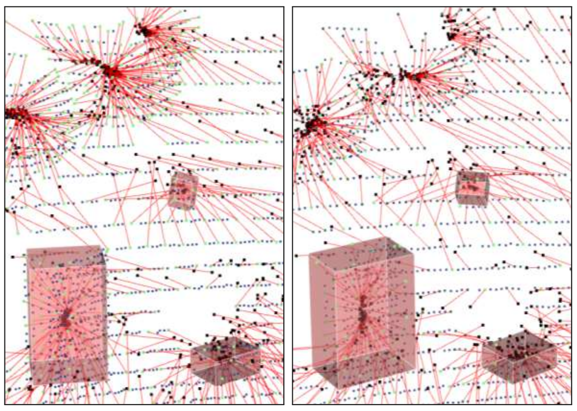
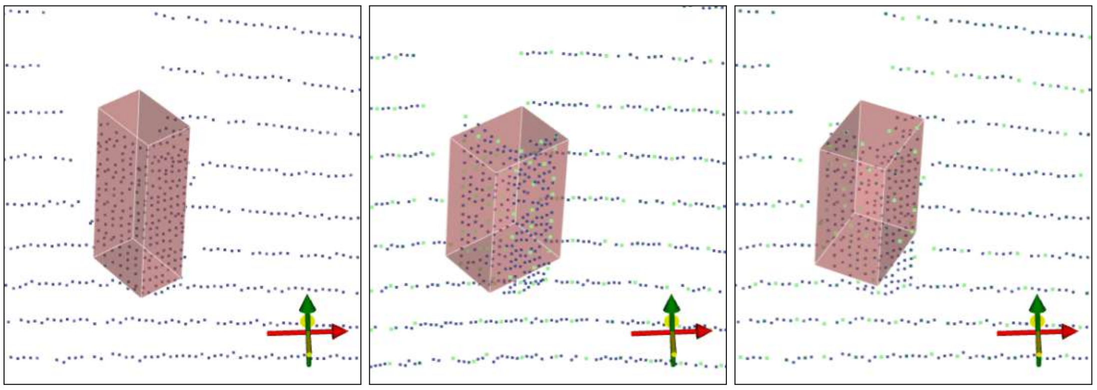

(a) Votes on DS_{synth} (b) Votes on DS_{real}

Figure 4.6: Comparison of generated votes on DS_{synth} and DS_{real}. Vote vectors are illustrated as red lines, each capped with a black dot at the tip.

In Figure 4.5 typical failure case is demonstrated, showing the weaknesses of our proposed method.



(a) Ground truth (b) Prediction DS_{real} (c) Prediction DS_{real}

Figure 4.7: An example of a failure case, where the orientations of predicted bounding boxes on DS_{synth} and DS_{real} are incorrectly estimated.

4.3 Experiment 2: Euler angles vs. 6D Rotation Matrix

In this experiment, we analyze the impact on the model's performance when employing the 6D rotation matrix representation instead of Euler angles for orientation estimation. For one model (OR=EulerOR=Euler), we regressed the Euler angles of the orientations of objects, and for another model (OR=6DOR=6D), we estimated the six parameters needed for the construction of the rotation matrix using the Gram-Schmidt process.

4.3.1 Quantitative Results

Dataset	OR	AP_{25}	AP_{50}	AOS_{25}	AOS_{50}	AAE_{25}	AAE_{50}
DS_{synth}	Euler	82.2	**62.8**	75.6	**57.6**	32.9	33.4
	6D	**85.7**	57.4	**83.4**	55.8	**18.8**	**18.9**
DS_{real}	Euler	**81.4**	**51.6**	74.1	**46.3**	34.7	37.1
	6D	78.6	42.9	**76.1**	41.6	**20.6**	**19.5**

Table 4.2: Performance of the synthetically trained model on the synthetic and real validation datasets. The employed orientation representation (OR) is indicated by the column OR.

As seen in Table 4.2, the 6D orientation representation enabled the model to significantly improve orientation estimation in both DS_{synth} and DS_{real}. We can see an improvement of AAE of around 14° for DS_{synth} and 14°-17° for DS_{real}. While it is important to note that we calculate those orientation errors by comparing a predicted orientation with the closest 90° flipped ground truth orientation, we can still clearly see an improved

alignment of the predictions and ground truth orientations. Orientation estimations that are off by 90° in any of the objects three local axes would contribute to those metrics as being a perfect estimation. We explained in the reasoning for this evaluation method in detail in Section 4.1.2.
While the model employing the 6D rotation representation has improved on the estimation of orientations of the cuboids, we observed a slight decline in AP_{50} by 5.4 points on DS_{synth} and a larger decline by 8.7 points on DS_{real}.

4.4 Experiment 3: Impact of Canonical Rotation Representation

If an object is rotationally symmetric, multiple correct orientations exist that describe the geometric appearance of that object. Due to this ambiguity, contradicting ground truths exist that might hinder the learning process. We therefore introduce a canonical orientation for such objects, explained in Section 3.1.2, that tries to resolve this issue. In this experiment, we measure the impact of using such a symmetry-aware orientation representation in the training process.

4.4.1 Quantitative Results

Dataset	OR	CO	AP_{25}	AP_{50}	AOS_{25}	AOS_{50}	AAE_{25}	AAE_{50}
DS_{synth}	Euler	✓	**85.5**	62.6	**81.1**	**59.7**	**26.3**	**24.9**
	Euler	✗	82.2	**62.8**	75.6	57.6	32.9	33.4
	6D	✓	84.2	**65.9**	80.8	**63.4**	23.1	22.3
	6D	✗	**85.7**	57.4	**83.4**	55.8	**18.8**	**18.9**
DS_{real}	Euler	✓	78.7	44.2	**74.2**	41.8	**27.9**	**26.8**
	Euler	✗	**81.4**	**51.6**	74.1	**46.3**	34.7	37.1
	6D	✓	69.3	**45.0**	66.1	**43.2**	24.7	22.6
	6D	✗	**78.6**	42.9	**76.1**	41.6	**20.6**	**19.5**

Table 4.3: Performance comparison of models employing different orientation representations (OR) when employing or not employing the proposed canonical orientation (indicated by ✓ and ✗ in column CO)

Given the inherent symmetry in cuboids, introducing a canonical orientation to account for the 180° flips around local axes is an intuitive solution to avoid ambiguities. For the models utilizing Euler angle representation, the orientation estimation measured by AAE improves significantly for both validation datasets, as evidenced in Table 4.3. On DS_{real}, we saw a decline in AP_{25} by almost 3 points and in AP_{50} by 7.4 points. Interestingly, we see a different behaviour when the 6D orientation representation is employed: The AAE metrics worsened with the use of the canonical orientation representation by around 3-4° for DS_{synth} and DS_{real}.

Still, taking the best configuration in terms of orientation estimation for models employing the 6D and Euler angles representations, the model utilizing the 6D representation outperforms the other by a large margin for orientation estimation and has a slight underperformance of 1.3 points in AP_{50} on DS_{real}.

4.5 Summary and Discussion

The first experiment confirmed the viability of utilizing a synthetically trained model on real world data. Given that success, it also demonstrates that the sim-to-real gap of learning orientations in our setting is very slim, while the detection capability suffered a little more degradation. This suggests, our synthetic point clouds still exhibit different characteristics or qualities compared to the real-world ones.
In our next experiment, we explored the influence different orientation representations and we conclude that the utilization of the 6D orientation representation has outperformed Euler angles in AAE by 14° on the synthetic dataset and by 14°-17° on the real-world dataset.
In the third and last experiment, we showed the impact the proposed canonical orientation. We trained the same model configurations again but this time we map the orientations of objects to our proposed canonical orientation. While we saw a significant improvement of orientation estimation for models employing the Euler angles representation, a decline in detection performance (AP_{50}) on DS_{real} was observed. Interestingly, the models employing the 6D orientation representation showed different results, favoring the raw orientation without canonicalizing it. We cannot explain the phenomenon, but we suspect the discontinuity of the rotation representation that is introduced with the canonicalization process might be the reason.

CHAPTER 5

Conclusion and future work

In this chapter, we consolidate and discuss key findings of our research. We provide a comprehensive overview of the strengths and weaknesses of the proposed methodology by evaluating the results of our experiments and discussing drawn conclusions. Further, we provide an outlook on potential further investigations.

5.1 Conclusions

The primary motivation of this thesis is the detection and pose estimation of geometric primitives in a 3D scene. We elaborated a vote-based bottom-up method, building upon [QLHG19], to detect cuboids in point clouds and estimate their pose. A huge dataset with point clouds containing objects with annotated 9DoF pose is required to train such a model. Since no publicly available dataset exists with the necessary annotations, we resort to generating synthetic data using our proposed data generation pipeline. A downside of such a data generation scheme is the introduction of a sim-to-real gap. This phenomenon is hard to overcome, and we tried to minimize this gap using purely geometric data and not relying on visual appearance of objects. Our experiments demonstrate a very slim sim-to-real gap in terms of estimating orientations of objects and a moderate gap for the detection capability in general.
Further, we analyzed the impact of using different orientation representations and come to the conclusion that the 6D orientation representation certainly benefits the orientation estimation capability in comparison the using Euler angles. Still, the utilization of Euler angles allowed the method to achieve higher or close scores on the AP metrics on the real-world validation dataset.
Lastly, an interesting conclusion is that the proposed mapping to a canonical orientation representation is not always beneficial. Our experiments show, that only models utilizing the Euler angles representation benefit from this and models using the 6D orientation representation performed better in terms of orientation estimation when learning the

unprocessed orientations. We suspect that the discontinuity we introduced by canonicalizing the orientation affected the model to learn in this case. Overall, the experiments show promising results, indicating the applicability of our proposed method in the real world and room for improvement on the representation of orientations.

5.2 Future work

As stated in the conclusion, the proposed canonical orientation representation to eliminate mixed signals did not have the expected effects when employing the 6D orientation representation. Upcoming work can be dedicated to finding another way to reduce the many equivalent orientation of symmetric objects into a single canonical and continuous representation. Alternatively, methods to address the presence of multiple valid states could be explored. Further, an extension of this method to multiple geometric primitives or even generally symmetric objects could be a possible avenue for research.

List of Figures

List of Tables

List of Algorithms

Bibliography

[BYW$^+$20] Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. Deep learning on 3d point clouds. *Remote Sensing*, 12(11):1729, 2020.

[CLSJ19] Yilun Chen, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Fast point R-CNN. *CoRR*, abs/1908.02990, 2019.

[Com18] Blender Online Community. Blender - a 3d modelling and rendering package. `https://www.blender.org`, 2018.

[Cor] NVIDIA Corporation. Nvidia omniverse platform. `https://developer.nvidia.com/nvidia-omniverse-platform`.

[DH72] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, jan 1972.

[EG] Inc. Epic Games. Unreal engine. `https://www.unrealengine.com/`.

[GDDM13] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[Gir15] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[HAAH22] Thorsten Hempel, Ahmed A. Abdelrahman, and Ayoub Al-Hamadi. 6d rotation representation for unconstrained head pose estimation. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 2496–2500, 2022.

[JSZK15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.

[KLR18] Abhijit Kundu, Yin Li, and James M. Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3559–3568, 2018.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[LAE$^+$16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing.

[LVC$^+$18] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *CoRR*, abs/1812.05784, 2018.

[MS15] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Ieee/rsj International Conference on Intelligent Robots and Systems*, pages 922–928, 2015.

[MXN$^+$21] Jiageng Mao, Yujing Xue, Minzhe Niu, et al. Voxel transformer for 3d object detection. *ICCV*, 2021.

[Ous] Inc. Ouster. Ouster os-64 lidar sensor. `https://www.ouster.com/`.

[QLHG19] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[QSMG16] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[QYSG17] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[RNJ21] Stefan Reitmann, Lorenzo Neumann, and Bernhard Jung. Blainder—a blender ai add-on for generation of semantically labeled depth-sensing data. *Sensors*, 21(6), 2021.

[SMKLM15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[SWL19] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[TFR$^+$17] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017.

[THSD17] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *CoRR*, abs/1702.05464, 2017.

[TSF17] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. *CoRR*, abs/1711.08848, 2017.

[XSNF17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *CoRR*, abs/1711.00199, 2017.

[ZBL$^+$20] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks, 2020.