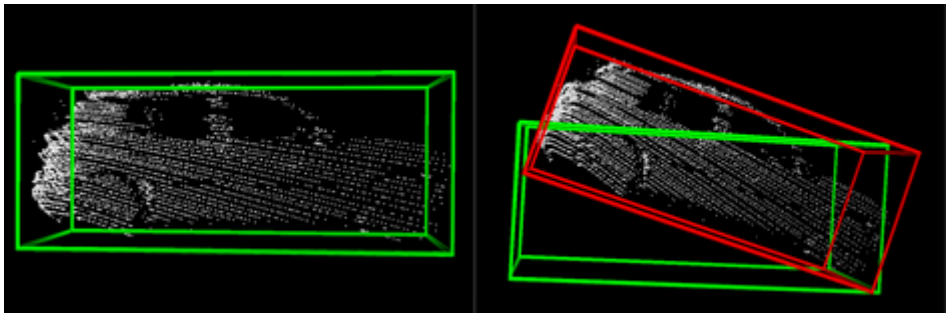
TÉCNICO LISBOA



Data Augmentation for LiDAR-based 3D Vehicle Detection on Sloped Roads

João Miguel de Loureiro Ferreira

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Prof. Teresa Maria Canavarro Menéres Mendes de Almeida
Supervisor: Prof. Alexandre José Malheiro Bernardino
Member of the Committee: Dr. Pedro Daniel dos Santos Miraldo

December 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First of all, I want to thank to Mother, Grandparents and Tomás. To the Mother for the love, strength and sacrifice out of this world that allowed me to have a Home and an Education that so many around us cannot have, even though it is their right. To the Mother for the Father that she was. Thank you for everything! To the Grandparents for all the affection, presence, knowledge. For the walks, the patience, the books. Also for the Parents that they were, especially the Grandfather: my model, my inspiration. You personify altruism, selflessness, sacrifice, the sense of justice and communion. I truly hope that one day I will be a person as good and pure as you. To Tomás for being the brother I wanted so much and asked for. You didn't come when I wanted, but you came when you needed to. I hope I've been a good role model for you and I want you to know that you can count on your older *mano* for anything, always. To you all: now it's my turn to take care of you. It's the least I can do. And I promise to do it the best I know and with the greatest pleasure possible.

To Federica, for the new Home and all that it encapsulates. For the company, the intimacy, the happiness and the constant learning and encouragement to be more and better. For your huge support during this process, even against my will. For the promise that we have all the rest of the time for new experiences, travels and learning (now I'm going to learn to cook like you!). Thank you so much for everything, my love.

I want to thank all my colleagues and professors that I had the opportunity to meet as a student at Instituto Superior Técnico. It was a long phase of my life, full of challenges and doubts, but in the end it turned out to be a positive experience where I graduated as an Engineer and, above all, as a Person.

And if the balance has been positive, it is largely due to the friends the University has given me. For this, I am very grateful to all of you, especially Francisco, Inês and Pedro. Thank you for helping each other, for the confidence, for the sharing, for all the moments, days and long nights spent and for all the years of friendship that we have ahead of us.

Finally, I would like to thank my supervisor Prof. Alexandre Bernardino for guiding me throughout this work. His patience, availability and knowledge sharing were a great support and crucial to the success of this work.

Resumo

Com o intuito de tornar as estradas cada vez mais seguras, os sistemas de condução autónoma pretendem desenvolver carros onde a intervenção humana seja reduzida, ou até mesmo eliminada. No entanto, devido às condições variáveis e diferentes elementos que existem nos ambientes rodoviários, estes sistemas necessitam de constante melhoria. Sabendo que os principais conjuntos de dados disponíveis na literatura não incluem a orientação vertical dos objectos anotados e que nem todas as grandes cidades são planas (onde esta orientação possa ser ignorada), propomos um algoritmo de aumento de dados (aplicado em dados LiDAR) que manipula estes objectos através da aplicação de uma orientação vertical. Para a criação de novos conjuntos de dados, usámos o conjunto de dados KITTI. Obtivemos modelos de detecção por implementação da rede de detecção Complex-YOLO e comparámo-los com o estado da arte para avaliar o impacto da manipulação da orientação vertical no desempenho da regressão da orientação horizontal. Concluímos que a perda de desempenho nas principais classes de detecção da literatura é residual para a tipologia normal do detector de objectos e considerável para a tipologia de menor dimensão, apresentando esta última um desempenho computacional muito superior. Concluímos também que o aumento da variação da orientação vertical pode levar a uma redução generalizada no desempenho dos modelos de detecção. No entanto, os resultados gerais não se revelam um retrocesso, pelo que é disponibilizado um repositório do código desenvolvido para este trabalho com a intenção de que sirva como ponto de partida para futuro trabalho.

Palavras Chave

Sistemas de condução autónoma, aprendizagem profunda, aumento de dados, dados de LiDAR, detecção de objectos 3D

Abstract

With the intention of making roads increasingly safer, autonomous driving systems aim to develop autonomous cars where human intervention is ultimately eliminated. However, due to the complex conditions and varied elements that exist in road environments, these systems need constant improvement. Knowing that the main datasets available in the literature do not present information regarding the vertical orientation (pitch angle) of the annotated objects and that not all large cities are flat, we propose a data augmentation algorithm (applied in LiDAR data) that manipulates the ground truth objects through the application of a pitch angle transformation. For the creation of new datasets, we used the KITTI dataset, as it is one of the most used in the literature. We obtained object detection and classification models and compared them with the state-of-the-art to evaluate the impact of manipulating the vertical orientation on the performance of the regression of the horizontal orientation (yaw angle). We conclude that the loss of detection performance in the main detection classes of the literature is residual for the Normal typology and considerable for the Tiny, even though the latter is far superior in relation to computational performance. We also conclude that the increase in the variation of the vertical orientation can lead to a global reduction in the detection performance. However, the overall results are promising, so a repository of code developed for this work is made available with the intention of serving as a starting point for future suggested work.

Keywords

Autonomous driving systems, deep learning, data augmentation, LiDAR data, 3D object detection

Contents

List of Tables

List of Figures

Acronyms

AOS	Average Orientation Similarity
AP	Average Precision
BEV	Bird's-Eye View
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DoF	Degrees of Freedom
E-RPN	Euler-Region-Proposal Network
FN	False Negative
FoV	Field of View
FP	False Positive
FPS	Frames Per Second
GPU	Graphics Processing Unit
ISR	Institute for Systems and Robotics
IoU	Intersection over Union
MLP	Multi-Layer Perceptron
LiDAR	Light Detection And Ranging
NLP	Natural Language Processing
NMS	Non-Max Suppression
RNN	Recurrent Neural Network
ToF	Time of Flight
TP	True Positive

Nomenclature

The next list defines several nomenclatures that will be later used within the body of the document.

$\mathbf{1}$	Vector of ones with size accordingly to the matrix where it is included
$\mathbf{c}$	Object's centroid vector (x, y, z coordinates)
$\mathbf{R}_y^c(\psi)$	Rotation matrix given by Object's yaw orientation in Camera coordinate system
$\mathbf{R}_y^c(\theta)$	Rotation matrix given by pitch orientation in Camera coordinate system
$\mathbf{T}_y^c(\psi)$	Transformation matrix given by Object's yaw orientation in Camera coordinate system
$\tilde{}$	Homogenised matrix or vector
${}'\mathbf{T}_o(\theta)$	Transformation matrix given by pitch orientation in Object coordinate system
${}^{c''}BBox$	Matrix of Object's bounding box corners' coordinates after applying pitch rotation and translation in Camera coordinate system
${}^{c''}Obj$	Matrix of Object's points after applying pitch rotation and translation in Camera coordinate system
${}^{c'}\mathbf{c}$	Object's new centroid coordinates after applying pitch translation in Camera coordinate system
${}^{c'}BBox$	Matrix of Object's bounding box corners' coordinates after applying pitch orientation in Camera coordinate system
${}^{c'}Obj$	Matrix of Object's points after applying pitch orientation in Camera coordinate system
${}^{c}\mathbf{P}$	Matrix of points belonging to Point Cloud in Camera coordinate system
${}^{c}\mathbf{R}_v$	Rotation matrix from Velodyne to Camera coordinate system
${}^{c}\mathbf{T}_o$	Transformation matrix from Object to Camera coordinate system
${}^{c}\mathbf{T}_v$	Transformation matrix from Velodyne to Camera coordinate system
${}^{c}\mathbf{t}_v$	Translation vector from Velodyne to Camera coordinate system

$^{c}Bbox$ Matrix of Object's bounding box corners' coordinates in Camera coordinate system

$^{c}Bbox_{\psi}$ Matrix of orientated Object's bounding box corners' coordinates in Camera coordinate system

$^{c}BBox_{\psi}\mid_{max\{x\}}$ Maximum X-axis value of Object's orientated bounding box in Camera coordinate system

$^{c}BBox_{\psi}\mid_{max\{y\}}$ Maximum Y-axis value of Object's orientated bounding box in Camera coordinate system

$^{c}BBox_{\psi}\mid_{max\{z\}}$ Maximum Z-axis value of Object's orientated bounding box in Camera coordinate system

$^{c}BBox_{\psi}\mid_{min\{x\}}$ Minimum X-axis value of Object's orientated bounding box in Camera coordinate system

$^{c}BBox_{\psi}\mid_{min\{y\}}$ Minimum Y-axis value of Object's orientated bounding box in Camera coordinate system

$^{c}BBox_{\psi}\mid_{min\{z\}}$ Minimum Z-axis value of Object's orientated bounding box in Camera coordinate system

$^{c}y_{M}$ Maximum Y-axis value in Camera coordinate system

$^{c}y_{offset}$ Vertical offset computed for pitch translation in Camera coordinate system

$^{c}y_{t}$ Threshold value applied in Y-axis in Camera coordinate system

$^{o\prime}BBox$ Matrix of Object's bounding box corners' coordinates after applying pitch orientation in Object coordinate system

$^{o\prime}Obj$ Matrix of Object's points after applying pitch orientation in Object coordinate system

$^{o}\mathbf{T}_{c}$ Transformation matrix from Camera to Object coordinate system

$^{o}Bbox$ Matrix of Object's bounding box corners' coordinates in Object coordinate system (without yaw orientation)

$^{o}Bbox_{\psi}$ Matrix of orientated Object's bounding box corners' coordinates in Object coordinate system

^{o}Obj Matrix of Object's points in Object coordinate system

^{PCL} Refers to Point Cloud

$^{v}\mathbf{P}$ Matrix of points belonging to Point Cloud in Velodyne coordinate system

${}^{v}\mathbf{t}_{c}$	Translation vector from Camera to Velodyne coordinate system
${}^{v}Bbox$	Matrix of Object's bounding box corners' coordinates in Velodyne coordinate system
$_{M}$	Maximum value
$_{m}$	Minimum value
h	Height value
l	Length value
p_x	x coordinate of point (generic)
p_y	y coordinate of point (generic)
p_z	z coordinate of point (generic)
w	Width value

1

Introduction

Contents

In this first chapter, a brief introduction to the proposed study theme will be given. In addition, the objectives to be fulfilled by this work will be defined. Finally, the thesis outline will be presented.

1.1 Motivation

The number of annual road accidents that occur all over the world, which sometimes result in death or serious injury, brings the urgent need to make driving increasingly safer. With this in mind, autonomous driving systems have emerged that aim to develop autonomous cars capable of reducing, or even completely eliminating, the need for human intervention and, thus, increasing road safety. One of the main components of these systems is real-time 3D object detection and classification, where accurate object localisation is essential for route planning as well as collision avoidance.

In recent years, several datasets have emerged in the autonomous driving scientific community that use various sensors for data acquisition. Of these, the Light Detection And Ranging (LiDAR) [1] sensor has proved to be the most used technology, as it is able to map the surrounding environment through 3D points. However, these datasets have been mainly obtained in almost flat cities, so the various deep learning methods that have emerged suggest approaches that consist of vehicle detection and regression of their orientation, but only in the horizontal plane.

The fact that cities like Lisboa (Portugal) [5] or San Francisco (California, USA) [6] are not flat also poses a great challenge for the detection of objects. Even though the objects will be in a similar plane the closer they are and not presenting different pitch orientations, this might not happen when the distance between them is greater and it will difficult the trajectory prediction and reaction anticipation of these systems and, ultimately, reduce the safeness.

During the research of state-of-the-art methods focused on the task of detecting and classifying objects and taking into account the aforementioned challenges, the Complex-YOLO network [3] proved to be a very interesting and promising work within the real-time 3D object detection and classification by introducing a complex angle regression of the objects' horizontal orientation (yaw angle). Therefore, this work will take this detection network as a starting point for a study on the impact of implementing a variation of the vertical orientation (pitch angle) of objects that allows to bring an even more precise, knowledgeable and relevant description of objects, ultimately making autonomous driving safer.

1.2 Objectives

The main objective of this dissertation is to create new datasets where the ground truths can present pitch orientation with variable values and to train models that can detect vehicles presenting a significant pitch angle, as can be the case on sloped roads.

Currently, object detection models are not able to deal with these cases, as they are trained with datasets acquired in plain terrains where most of the vehicles are captured without considerable vertical orientations. Since there are not datasets considering sloped roads, we had to develop an algorithm that allows manipulating existing datasets in order to modify the representation of vehicles in point clouds by varying their vertical orientation, more precisely through the pitch angle. For this, the KITTI dataset [4] will be used as a starting point, as this is the most used benchmark dataset in the literature.

In order to validate the proposed approach, object detection models will be trained with Complex-YOLO [3], a state-of-the-art real-time 3D object detection network, that uses YOLOv2 [7], a fast 2D standard object detector for RGB images. This network will be implemented by adapting an available code repository [8] that uses a better version of this object detection system, YOLOv3 [9]. These models will be compared between themselves and the original model, so that the impact of the implementation of a vertical orientation in objects can be studied.

Ultimately, it is also expected from this dissertation to make available a repository of code developed during its execution so it can be used for dataset creation and future work.

1.3 Thesis Outline

This document is organised in six chapters. Chapter 1 is an introduction to the research to be carried out that covers the motivation and relevance of working on the subject of 3D real-time object detection and classification applied to autonomous driving systems; exposes some limitations that this topic contains; and defines objectives to be achieved with a view to solving the identified problem.

Chapter 2 starts by providing a theoretical overview of LiDAR and deep learning, respectively the technology and technique employed to execute this work. Afterwards, presents state-of-the-art methods for data augmentation and LiDAR-based 3D object detection and classification applied to autonomous driving systems.

Chapter 3 focuses on the presentation of the KITTI dataset [4] and of the data augmentation algorithm to be applied on this dataset, creating new ground truths. In relation to the algorithm, this chapter presents a thorough explanation, as well as an overview and a pseudo-code structure.

Chapter 4 defines the experimental setup used to validate the implemented algorithm that allows the implementation of the objectives of this work, more specifically the computational infrastructure and how the training and testing tasks were performed. This chapter also describes the performance metrics used to evaluate and compare the obtained models with the ones present in the benchmark.

Chapter 5 presents a thorough analysis and discussion on the results obtained for the data augmentation algorithm through the proposal of three studies that focuses on the following analysis: vehicle detection accuracy, computational performance and class-specific detection.

Chapter 6 sums up the main contributions and achievements throughout the work developed on this thesis. Moreover, possible work to be done in the future is suggested.

2

Background

Contents

In this chapter is firstly given a theoretical overview of LiDAR and Deep Learning. Afterwards, are presented state-of-the-art methods for data augmentation and LiDAR-based 3D object detection and classification. In the latter, one of the methods in the literature, Complex-YOLO [3], is described in more detail, since it is used to carry out the studies presented in Section 5.1.

2.1 Theoretical Overview

2.1.1 LiDAR

LiDAR, as described in [1], is an active sensor that illuminates the surroundings by emitting laser beams. This sensor outputs 3D point clouds that are comprised by points described by its 3D coordinates (X, Y and Z) and its intensity value representing the reflected laser energies. This last value also takes into account the travelling time of the laser between the sensor and the target in both directions, *i.e.*, Time of Flight (ToF), and its processing allows to precisely obtain depth ranges that consist of the distance between each point and the subject.

A system including LiDAR sensors can be divided into two subsystems named laser rangefinder and scanner:

- The first subsystem is a complex one, containing a laser transmitter that fires pulses of laser lights into the surrounding environment and targets; a photo-detector where electronic signals are generated from the reflected beams; a set of optics that both collimate the emitted laser and focus the reflected signal into the photo-detector; a set of signal processing electronics that estimate the distance between the laser source and the target;
- The latter subsystem steers laser beams at different azimuths and vertical angles that determines the direction at which they are fired.

Figure 2.1, borrowed from [1], exemplifies a LiDAR system.

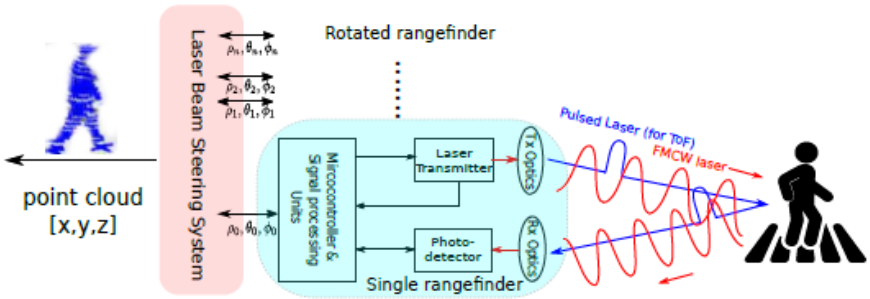


Figure 2.1: Example of a LiDAR system. [1]

2.1.2 Deep Learning

Deep learning is a machine learning technique based on neural networks that try to mirror the way humans acquire certain types of knowledge: learn by example. This technique is relevant because it makes the processes of collecting, analysing and interpreting large data sets quick and easy. Therefore, deep learning has many applications, such as automatic speech recognition, image recognition, object detection and Natural Language Processing (NLP).

In a brief historical context authored by Zhao *et al.* [10], it is indicated that deep learning was explored again in the beginning of the XXI century due to several factors of which the following stand out: the rapid development of parallel computing systems, such as Graphics Processing Unit (GPU) clusters; and the significant advances in the design of network structures and training strategies (such as dropout and data augmentation techniques) that reduced overfitting problems.

Deep Neural Network (DNN)s are the networks used in deep learning and the most popular types are Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), which can be described as follows:

- MLP: the most basic DNN that is composed of a series of fully connected layers, each one consisting of a set of non-linear functions with a weighted sum of all outputs coming from the previous one;
- CNN: the most typically used network in the area of Computer Vision (where this dissertation can also be considered) and consist of one or several convolutional layers that extract simple features from the input through the execution of convolutional operations;
- RNN: this model uses sequential data feeding where a given input consists of itself and the previous samples, and therefore the connections between nodes form a direct graph along a temporal sequence.

2.2 State-of-the-Art methods

2.2.1 Data augmentation

As mentioned in the previous section, deep learning models have proved to be very relevant in the field of object detection studies. However, this type of models depends on large amounts of data so that training and subsequent results are consistent. Since it is not always possible to have large datasets, the need arose to develop and apply data augmentation techniques. These techniques have as main objectives to extend the number of data available in a dataset and, in some cases, to improve the quality of the data under study.

In the case of 3D object detection, the characteristics of LiDAR data make datasets easily occupy a large size, sometimes making it unbearable to store datasets with the required number of samples. Thus, data augmentation techniques prove to be extremely important and useful for the studies that have to be done in this field.

Hahner *et al.* [2] published a work exploring the most commonly used data augmentation techniques in the task of detecting 3D objects through LiDAR. Of these techniques, two possible categories stand out, although similar, which can be defined below:

- Global augmentations, where modifications are applied simultaneously to all points of the point cloud and to the annotations contained therein;
- Local augmentations, where independent and different modifications are applied to each annotation and to the points of the point cloud contained therein, respectively. The novel technique (pitch rotation data augmentation and ground truth manipulation) proposed in this work falls into this category.

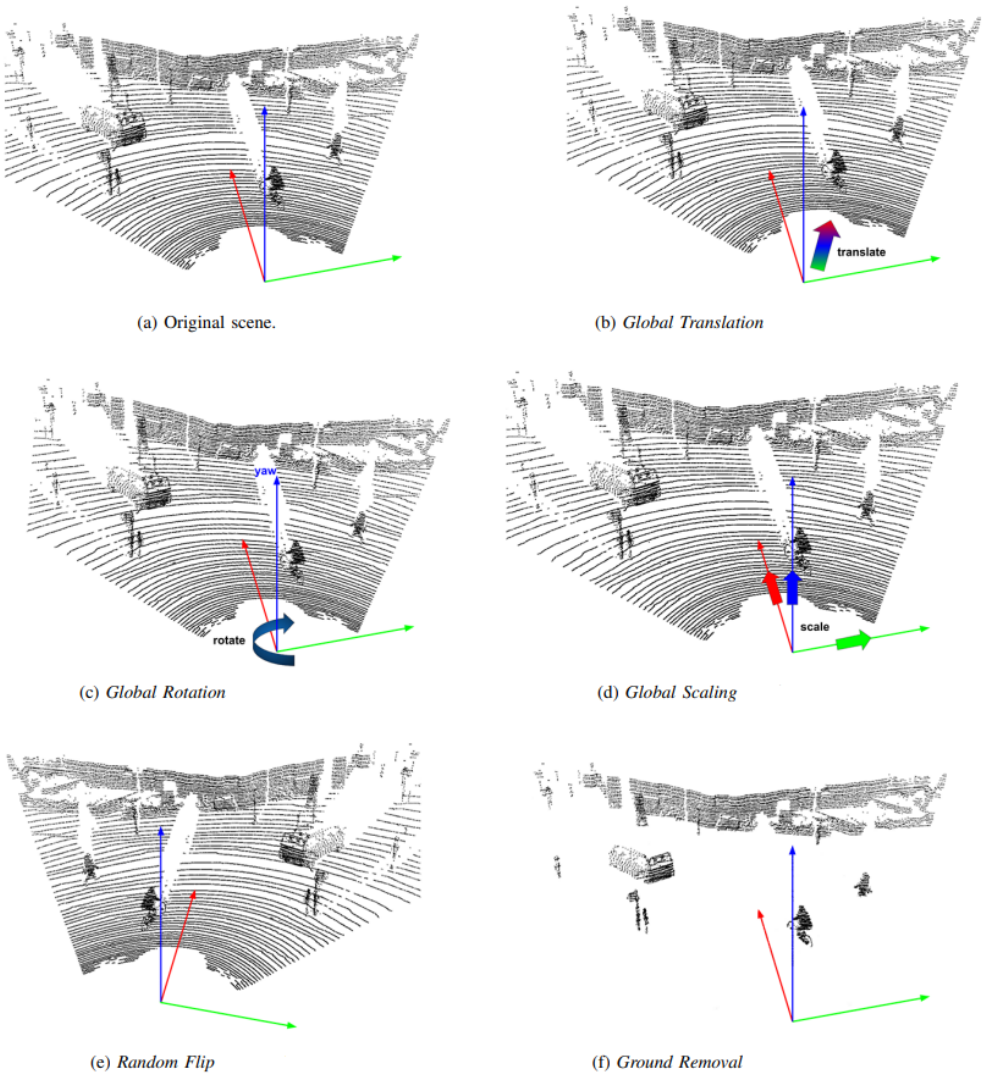
Figures 2.2 and 2.3, borrowed from [2], allow the visualisation of the techniques investigated on global and local augmentation. Translations, rotations and scaling can be considered as both global and data augmentation techniques. In the case of translations, a constant variation along each X, Y and Z coordinate is added to the coordinates of the points. Regarding rotations, in the literature these have been only applied along the vertical axis. Finally, the scaling technique is similar to the first one described here, but where the scalar obtained is common to the three coordinates X, Y and Z and where the new values are obtained by multiplication of that scalar by the object's coordinates relative to the centroid.

Within the category of global augmentations, there are two more techniques: random flip and ground removal. The random flip technique consists of mirroring the point cloud with a probability of 50% only along the X-axis (facing forward). Finally, the ground removal technique removes all points whose z coordinate value is lower than a threshold percentile of all existing values for this coordinate in the point cloud, in order to remove the ground floor points.

2.2.2 LiDAR-based 3D Object Detection

There are several data representations in the LiDAR-based 3D object detection task depending on how the points that make up the point clouds are interpreted. Of these, three stand out: voxel grids, point sets and hybrids.

Methods based on voxel grids start by dividing the 3D points into a grid of voxels (which are equivalent to the 3D representation of pixels) that are encoded using metrics such as voxel feature means and covariances. MV3D [11], PIXOR [12] and Complex-YOLO [3] are examples of methods that scattered

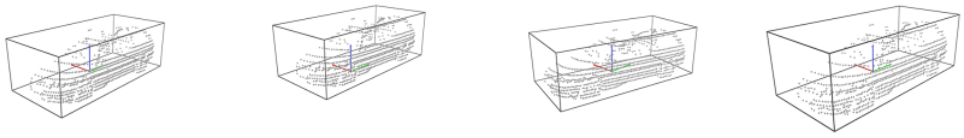


(a) Original scene. (b) *Global Translation*

(c) *Global Rotation* (d) *Global Scaling*

(e) *Random Flip* (f) *Ground Removal*

Figure 2.2: Visualisation of global augmentation techniques investigated in [2].



(a) Original annotation (b) *Local Translation* (c) *Local Rotation* (d) *Local Scaling*

Figure 2.3: Visualisation of local augmentation techniques investigated in [2].

voxels into pseudo-images that are later processed in object detection architectures. PointPillars [13] replaced 3D voxels by introducing the concept of 2D pillars in order to increase model efficiency. SECOND [14] and PVRCNN [15] are methods that kept the approach to 3D voxels, but applied 3D sparse convolutions on small voxels.

Point set methods treat point clouds as unordered sets, although the literature presents several ap-

proaches: object detection through a cropped point cloud obtained by 2D proposals in images (FPoint-Net [16]); proposition of objects directly from each point (PointRCNN [17]); proposal refinement through a sparse-to-dense (STD [18]) strategy.

Finally, hybrid methods try to combine different data representations in order to get the best out of them. Of these, the MultiView [19] and Pillar-MultiView [20] methods stand out. The first merges features learned by representing voxels into both Cartesian and spherical coordinate systems. The second method tries to improve the first one by projecting the obtained features in a Bird's-Eye View (BEV) perspective followed by convolution processing in order to obtain more robust features.

2.2.2.A Complex-YOLO

Complex-YOLO [3] is a single-stage network that, due to its contributions to the community, is considered state-of-the-art in the area of real-time 3D object detection. Although not using camera images as input, but only point clouds, this network resorts to YOLO9000 [7], a 2D object detector for RGB images that creates a BEV representation of the point cloud and considers it as an image. Complex-YOLO [3] expands this object detector by proposing an Euler-Region-Proposal Network (E-RPN) to estimate the pose of the object by adding the imaginary and real parts of the complex-valued representation of object's orientation to the regression network. This angular representation, composed of two values, is better than the single-valued angle representation because it does not suffer from the problem of wrapping around 180 degrees.

As it can be seen in Figure 2.4, borrowed from [3], the proposed new network takes as input a BEV RGB-map (directly in front of the LiDAR sensor origin) obtained through pre-processing techniques adapted from MV3D [11]. More specifically, the RGB-map is made up of pixels whose R, G and B channels encode, respectively, the maximum height, maximum intensity and normalised density of the points mapped in a segment of the point cloud. Next, the orientation of detected objects is obtained by complex angle regression and E-RPN, which allows Complex-YOLO to accurately detect multi-class 3D objects in real-time.

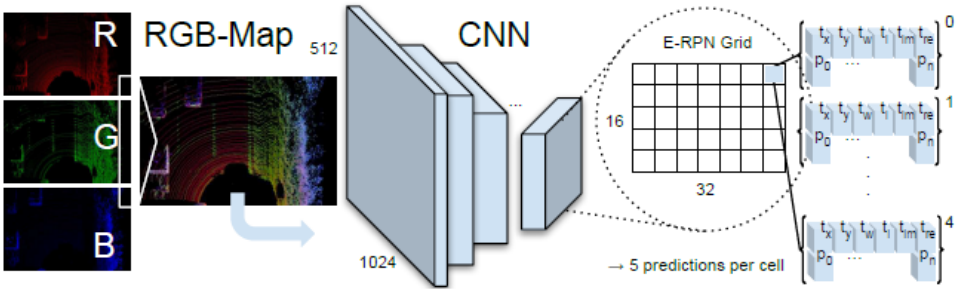


Figure 2.4: Complex-YOLO's pipeline. Image borrowed from [3]

Regarding the E-RPN itself, it parses the 3D position of the object, its dimensions, a general probability p_0, the class scores $p_1, ..., p_n$ and an orientation from the incoming feature map (see Figures 2.4 and 2.5). This network modifies the common Grid-RPN approach by adding a component referring to the complex angle description of the object's orientation. This addition implied other adaptations regarding the anchor box design and complex angle regression, which are explored below:

- Anchor Box design: due to angle regression, the number of Degrees of Freedom (DoF) increased. However, for reasons of efficiency, the authors [3] chose not to increase the number of predictions. Therefore, they defined only three different sizes and two angle directions as priors: vehicle size (heading up and down), cyclist size (heading up and down) and pedestrian size (heading left);
- Complex Angle regression: the orientation angle of each object can be obtained through the regression of the complex parameters t_{Im} and t_{Re}, simply through the formula $arctan_2(t_{Im}, t_{Re})$. Thus, singularities are avoided and a closed mathematical space is obtained, which has an advantageous impact on the generalisation of the model.

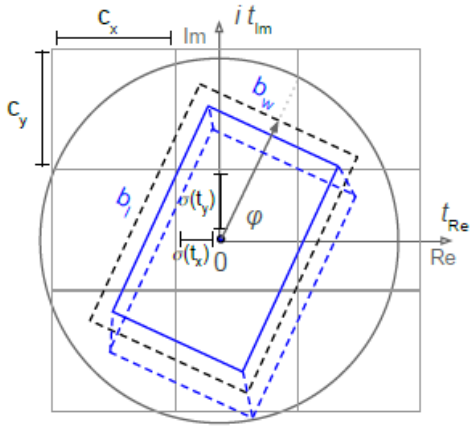


Figure 2.5: Complex-YOLO's bounding box regression. Image borrowed from [3].

3

Development

Contents

In this chapter, firstly the KITTI dataset [4] is presented. Secondly, is provided an overview of the data augmentation algorithm, developed to manipulate ground truths present in point clouds belonging to the KITTI dataset [4]. Next, it is given a detailed explanation of the main steps of this algorithm. Lastly, a pseudo-code structure of the algorithm is presented.

3.1 KITTI Dataset

Within the scope of autonomous driving, there are different datasets taking into consideration how the surrounding environment is analysed: either through object detection (e.g.: KITTI [4], nuScenes [21], Argoverse versions 1 [22] and 2 [23], Waymo [24]) or either through semantic segmentation (e.g.: Semantic3D.net [25], Semantic KITTI [26], KITTI 360 [27]).

Regarding the object detection task, in recent years high-quality and large-scale datasets have been publicly released, including readings from various types of sensors, as well as information about the environment through top-down rasterized maps (nuScenes dataset [21]) or even related to the ground height together with a vector representation of road lanes and their connectivity (Argoverse dataset [22] [23]).

However, the KITTI dataset [4], published in 2012, remains the benchmark dataset for object detection, being used for training and testing most models in the literature. In fact, it was the dataset used for the implementation of Complex-YOLO [3] that serves as the basis of this work, as well as for the development of the code repository resulting from this thesis.

From all the sensors in the setup, the most relevant is the LiDAR sensor - a Velodyne HDL-64E (where 64 represents the total number of channels) with a range of 120m, an accuracy of 2cm, an angular resolution of 0.09º, a 360º horizontal and 28º vertical Field of View (FoV). Having 64 channels (where each channel represents the emission of a laser beam) and at a frequency of 10Hz, this sensor is able to collect approximately 1.3 million points per second. Although the horizontal FoV is 360º, the KITTI dataset [4] only includes ground truth bounding boxes of objects that are present in the images, *i.e.*, that are contained in the FoV of the Camera sensor. Figure 3.1 illustrates the dimensions and mounting of the sensors in the vehicle, specially LiDAR and Camera.

In relation to the annotation of ground truth objects, they are present in label files where all values (15 columns in total) are separated by spaces and each row corresponds to one object. These values represent:

- Type (1 column): identifies the object's unique class. Possible classes are: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' and 'DontCare';
- Truncated (1 column): float from 0 (non-truncated) to 1 (truncated), referring to the object leaving image boundaries;

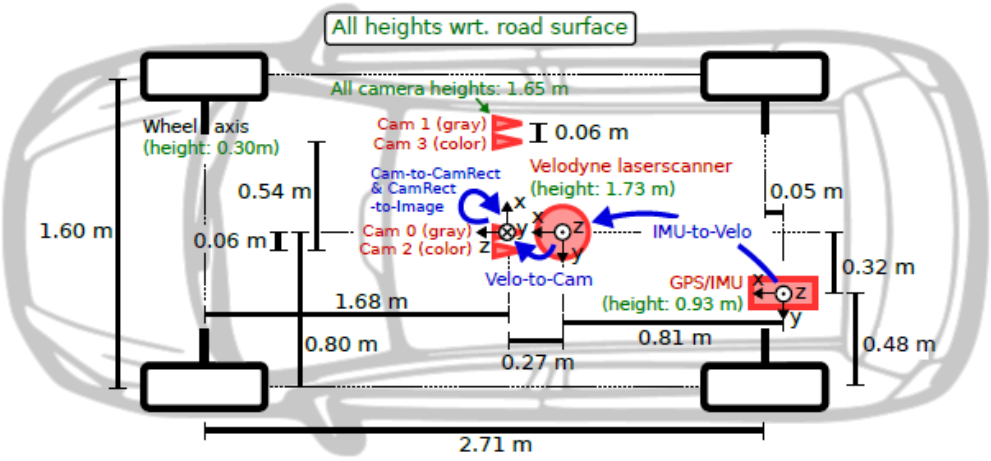


Figure 3.1: Representation of Vehicle's sensor setup, as seen in [4].

- Occluded (1 column): integer indicating occlusion state - 0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown;
- Alpha (1 column): observation angle of object, ranging between $[-\pi, \pi]$;
- BBox (4 columns): 2D bounding box of object in the image (0-based index) containing, respectively, left, top, right and bottom pixel coordinates;
- Dimensions (3 columns): 3D object dimensions (height, width and length) in meters;
- Location (3 columns): 3D object location (x, y, z)in meters in Camera coordinate system (see Figure 3.1);
- Rotation_y (1 column): rotation around Y-axis in Camera coordinate system (see Figure 3.1), ranging between $[-\pi, \pi]$.

For proper comprehension of the proposed data augmentation algorithm, it is crucial that it is stated the difference between the annotation's value Alpha and Rotation_y, present in [28]:

> *"The difference between Rotation_y and Alpha is that Rotation_y is directly given in camera coordinates, while Alpha also considers the vector from the camera centre to the object centre, to compute the relative orientation of the object with respect to the camera. For example, a car which is facing along the X-axis of the camera coordinate system corresponds to Rotation_y=0, no matter where it is located in the X/Z plane (bird's eye view), while Alpha=0 only when this object is located along the Z-axis of the camera. When moving the car away from the Z-axis, the observation angle will change."*

3.2 Data Augmentation Algorithm: Overview

In Figure 3.2 it is possible to see a flowchart that represents an overview of the data augmentation algorithm developed in this work. All the process represented is related to the processing of a point cloud of the KITTI dataset [4], so the process will be repeated as many times as the number of point clouds to be manipulated.

At the beginning of the process we have as inputs the data related to the point cloud: the set of points contained in it, the calibration matrices and the labels of the ground truth objects.

Initially, the point cloud is filtered in order to exclude points that are too far from the subject. Once the points of the point cloud are represented in the coordinate system of the LiDAR sensor used, the respective coordinates of these points are obtained in the coordinate system of the camera.

Then, the file containing the labels of the objects is scrolled through in order to manipulate one object at a time. A first evaluation is made on the object class, where objects that are not of the classes 'Car', 'Van' or 'Cyclist' are ignored. In the opposite case, where the object is of interest to the algorithm, the next steps consist in obtaining the coordinates of the centroid of the object in the LiDAR coordinate system and, taking into account the values of length, width and height of the annotated bounding box, in segmenting the boundaries of the point cloud where the points will be characteristic of the object under analysis.

The next evaluation to perform is if the object's bounding box is contained in the filtered point cloud obtained at the beginning of this process, in order to maintain the coherence of the algorithm and not manipulate objects that are not contained in this secondary point cloud. If it is contained, we obtain the boundaries of the bounding box in the camera coordinate system. Then, a Yaw rotation is applied on this system to the bounding box and, finally, a set of points is obtained that will be contained in these limits and, therefore, will belong to the object under analysis.

In case the set of points obtained contains 10 or less points, the object is ignored. Otherwise, a value in degrees is obtained for the pitch angle: if it is null, the object is also ignored because there will be no effective manipulation. Assuming the obtained pitch angle is non-zero, a rigid-body transformation is applied to the object and its bounding box, being the translation applied after the rotation, in order to avoid the object to contain points below the ground (which would not be feasible in the real case).

Finally, these new points are stored in the point cloud filtered on the camera coordinate system. Then, they are retrieved in the LiDAR coordinate system in order to be saved in the original point cloud. Before performing this sub-process for the next object, the new coordinates of the centroid of the object and the applied pitch angle are saved in the label.

After traversing all the objects present in the labels file, this process will output two new files with a new point cloud and the new labels of the objects, respectively.

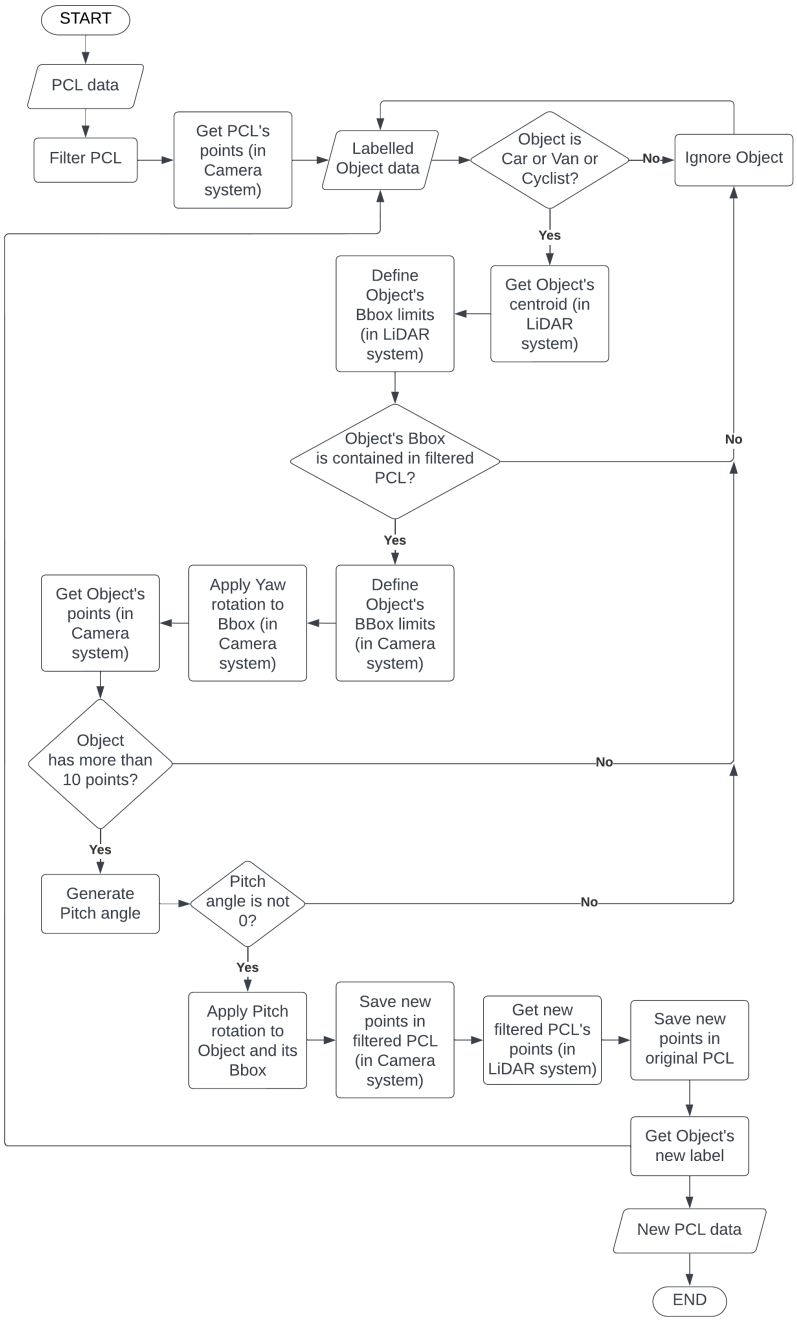


Figure 3.2: Flowchart of the developed data augmentation algorithm.

3.3 Data Augmentation Algorithm: Detailed Explanation

Since each point cloud is unique, initially it is necessary to load all necessary data for its processing. These data can be found in three files: one describing the points that belong to the point cloud; another containing calibration matrices, where is only relevant the matrix obtained for the calibration between

the LiDAR and camera sensors, because the object's centroid coordinates are labelled in Camera's coordinate system; a third one containing the labels of the annotated objects.

Knowing, from Section 3.2, the algorithm's complexity, as well as its relevance and contribution to this work, all its processes that involve calculations are detailed during this section. The remaining processes that are ancillary to the objective of this algorithm will only be identified in its pseudo-code structure contained in Section 3.4.

Lastly, taking into account that the main processes of this algorithm concern the transformation of objects into other coordinate systems, Figure 3.3 presents an illustrative scheme of the three possible coordinate systems (LiDAR, Camera and Object), as well as all the possible transformations between them.

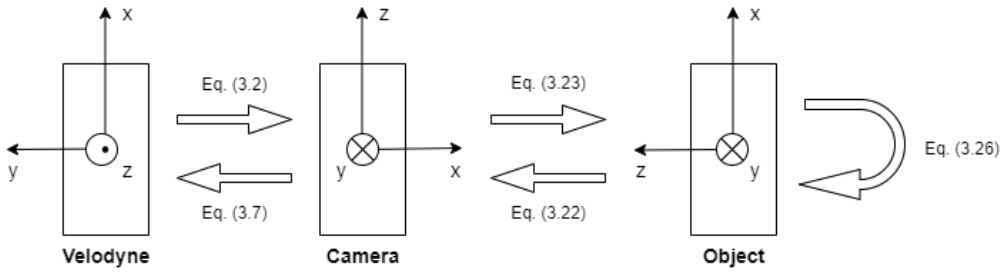


Figure 3.3: Representation of the different coordinate systems, as well as the equations representing transformations executed along this data augmentation algorithm.

3.3.1 Filter Point Cloud

In this first task are ignored points that are considered being too far from the vehicle (*i.e.*, subject). This task is also included in the original implementation of Complex-YOLO [3], while training the model.

For this algorithm, and having in mind the LiDAR's sensor coordinate system (see Figure 3.1), Equation 3.1 presents the boundaries that were defined:

$$\begin{cases} x_m^{PCL} = 0[m] \\ x_M^{PCL} = 50[m] \\ y_m^{PCL} = -25[m] \\ y_M^{PCL} = 25[m] \\ z_m^{PCL} = -1.73[m] \\ z_M^{PCL} = 1.27[m] \end{cases} \tag{3.1}$$

Here, the value of z_m^{PCL} is given by the symmetric value of the height of LiDAR's sensor (see Figure 3.1 - height of Velodyne laser-scanner, in green).

The application of the boundaries in the original point cloud will generate a filtered point cloud that will be used in the following tasks.

3.3.2 Get Point Cloud's points in Camera coordinate system

Knowing that the object's centroid coordinates and orientation are given in Camera coordinate system (see Section 3.1), most of the algorithm's processes will be done in this coordinate system. Therefore, for this task it is required the transformation matrix from Velodyne to Camera coordinate system (see Figure 3.3 that is given by a calibration matrix that can be defined as follows:

$$ {}^{c}\mathbf{T}_{v} = \begin{bmatrix} {}^{c}\mathbf{R}_{v} & {}^{c}\mathbf{t}_{v} \\ 0 & 1 \end{bmatrix} \tag{3.2} $$

Having defined this transformation matrix, the point cloud's points coordinates in Camera system can be obtained in the following way:

1. First, it is obtained a matrix only containing the points belonging to the point cloud - the last column containing the reflectance values is ignored, since it is not relevant for this algorithm.

$$ {}^{v}\mathbf{P} = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \tag{3.3} $$

   Here, n is the total number of points belonging to the point cloud.

2. The matrix defined in equation (3.3) needs to be homogenised and transposed for the final step.

$$ {}^{v}\tilde{\mathbf{P}} = \begin{bmatrix} {}^{v}\mathbf{P} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{3.4} $$

   Here and afterwards, **1** represents a vector of ones with size $1 \times n$, where n is given by the length of matrix ${}^{v}\mathbf{P}$.

3. Lastly, it is obtained a matrix of point cloud's points in Camera coordinate system from the multiplication between the matrices given in equations (3.2) and (3.4).

$$ {}^{c}\tilde{\mathbf{P}} = {}^{c}\,\mathbf{T}_{v} \cdot {}^{v}\,\tilde{\mathbf{P}} = \begin{bmatrix} {}^{c}\mathbf{P} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{3.5} $$

3.3.3 Get Object's centroid in LiDAR coordinate system

Object's centroid coordinates are given as location data in its label (see Section 3.1). This data is presented as a vector of dimension [1,3] and defines a single point described by its coordinates on the X, Y and Z axes.

1. Firstly, it is necessary to homogenise and transpose the centroid vector for matrix multiplication.

$$ {}^{c}\tilde{\mathbf{c}} = \begin{bmatrix} {}^{c}\mathbf{c} & 1 \end{bmatrix}^{\mathbf{T}} = \begin{bmatrix} {}^{c}\mathbf{c}_{x} & {}^{c}\mathbf{c}_{y} & {}^{c}\mathbf{c}_{z} & 1 \end{bmatrix}^{\mathbf{T}} \tag{3.6} $$

2. A Camera to LiDAR system transformation matrix is also required. This can be obtained through the matrix defined in equation (3.2).

$$ {}^{v}\mathbf{T}_{c} = \left[{}^{c}\mathbf{T}_{v} \right]^{-1} \tag{3.7} $$

3. Finally, it is obtained the centroid of the Object in LiDAR coordinate system from the multiplication of the transformation matrix with the homogeneous vector given in equations (3.7) and (3.6), respectively.

$$ {}^{v}\tilde{\mathbf{c}} = {}^{v}\,\mathbf{T}_{c} \cdot {}^{c}\,\tilde{\mathbf{c}} = \begin{bmatrix} {}^{v}\mathbf{c} & 1 \end{bmatrix}^{\mathbf{T}} \tag{3.8} $$

3.3.4 Define Object's bounding box limits in LiDAR coordinate system

Assuming that an Object is contained within the limits of the filtered point cloud, its orientation would only influence if it was intended to assess whether the Object is contained in the point cloud in its entirety or partially. As this evaluation is not relevant to the indication of the Object as of interest to the algorithm, and with a view to simplifying the code and optimising the algorithm's computation, in this task the Object's orientation will not be applied to its bounding box. In case the Object is identified as suitable for manipulation, the orientation will be applied in Section 3.3.7 so that the set of points belonging to this Object can be correctly obtained.

In conclusion, in order to define the Object's bounding box at this moment, it is essential to take into account its height (h), width (w) and length (l) values - which are defined in its annotation data as the 3D dimensions. Together with the Object's centroid and the LiDAR coordinate system arrangement (Figure 3.3), it is then possible to spatially define its bounding box.

1. Firstly, the minimum and maximum values for each coordinate are defined.

$$ \begin{bmatrix} {}^{v}x_{m} & {}^{v}x_{M} \\ {}^{v}y_{m} & {}^{v}y_{M} \\ {}^{v}z_{m} & {}^{v}z_{M} \end{bmatrix} = \begin{bmatrix} {}^{v}\mathbf{c}_{x} - \frac{l}{2} & {}^{v}\mathbf{c}_{x} + \frac{l}{2} \\ {}^{v}\mathbf{c}_{y} - \frac{w}{2} & {}^{v}\mathbf{c}_{y} + \frac{w}{2} \\ {}^{v}\mathbf{c}_{z} & {}^{v}\mathbf{c}_{z} + h \end{bmatrix} \tag{3.9} $$

2. Taking into account how the Velodyne coordinates are arranged (Figure 3.3), the coordinates of the bounding box limits are given by the following matrix:

$$ {}^{v}Bbox = \begin{bmatrix} {}^{v}x_{M} & {}^{v}y_{M} & {}^{v}z_{M} \\ {}^{v}x_{M} & {}^{v}y_{m} & {}^{v}z_{M} \\ {}^{v}x_{m} & {}^{v}y_{M} & {}^{v}z_{M} \\ {}^{v}x_{m} & {}^{v}y_{m} & {}^{v}z_{M} \\ {}^{v}x_{M} & {}^{v}y_{M} & {}^{v}z_{m} \\ {}^{v}x_{M} & {}^{v}y_{m} & {}^{v}z_{m} \\ {}^{v}x_{m} & {}^{v}y_{M} & {}^{v}z_{m} \\ {}^{v}x_{m} & {}^{v}y_{m} & {}^{v}z_{m} \end{bmatrix}^{\mathbf{T}} = \begin{bmatrix} {}^{v}\mathbf{c}_{x} + \frac{l}{2} & {}^{v}\mathbf{c}_{y} + \frac{w}{2} & {}^{v}\mathbf{c}_{z} + h \\ {}^{v}\mathbf{c}_{x} + \frac{l}{2} & {}^{v}\mathbf{c}_{y} - \frac{w}{2} & {}^{v}\mathbf{c}_{z} + h \\ {}^{v}\mathbf{c}_{x} - \frac{l}{2} & {}^{v}\mathbf{c}_{y} + \frac{w}{2} & {}^{v}\mathbf{c}_{z} + h \\ {}^{v}\mathbf{c}_{x} - \frac{l}{2} & {}^{v}\mathbf{c}_{y} - \frac{w}{2} & {}^{v}\mathbf{c}_{z} + h \\ {}^{v}\mathbf{c}_{x} + \frac{l}{2} & {}^{v}\mathbf{c}_{y} + \frac{w}{2} & {}^{v}\mathbf{c}_{z} \\ {}^{v}\mathbf{c}_{x} + \frac{l}{2} & {}^{v}\mathbf{c}_{y} - \frac{w}{2} & {}^{v}\mathbf{c}_{z} \\ {}^{v}\mathbf{c}_{x} - \frac{l}{2} & {}^{v}\mathbf{c}_{y} + \frac{w}{2} & {}^{v}\mathbf{c}_{z} \\ {}^{v}\mathbf{c}_{x} - \frac{l}{2} & {}^{v}\mathbf{c}_{y} - \frac{w}{2} & {}^{v}\mathbf{c}_{z} \end{bmatrix}^{\mathbf{T}} \tag{3.10} $$

3.3.5 Check if Object's bounding box is contained in filtered Point Cloud

The Object is considered to be contained in the filtered point cloud if all the conditions of Equation 3.11 are met:

$$\begin{cases} {}^{v}x_m \geq x_m^{PCL} \\ {}^{v}x_M \leq x_M^{PCL} \\ {}^{v}y_m \geq y_m^{PCL} \\ {}^{v}y_M \leq y_M^{PCL} \\ {}^{v}z_m \geq z_m^{PCL} \\ {}^{v}z_M \leq z_M^{PCL} \end{cases} \tag{3.11}$$

This validation considers how the Velodyne coordinate system is defined and is relevant to preserve the coherence of the algorithm by not manipulating objects that might contain points outside of the filtered PCL's boundaries. Figure 3.4 presents some scenarios: Objects 1 and 2 are fully contained within the boundaries, thus taken into consideration for this algorithm; Object 3 is partially outside, therefore it will be ignored.

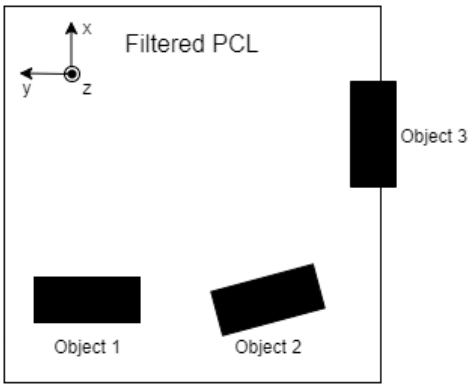


Figure 3.4: Example of objects in order to assess if they are within the filtered PCL's boundaries.

In case one or more conditions are not met, it is concluded that the Object is partially or totally outside the limits, thus being ignored and a new iteration of the inner cycle will be carried out where the next Object will be analysed.

In the opposite case, in which it can be determined that the Object is fully contained in the filtered point cloud, the following steps (detailed in Sections 3.3.6 and 3.3.7) are carried out beforehand so that the relevance of the Object within the scope of this implementation can be evaluated.

3.3.6 Define Object's bounding box limits in Camera coordinate system

The bounding box limits in Camera coordinate system are obtained analogously to the one described previously in Section 3.3.4.

1. As in Section 3.3.4, it is first obtained the extreme values for each coordinate considering the Camera coordinate system (Figure 3.3).

$$\begin{bmatrix} {}^{c}x_m & {}^{c}x_M \\ {}^{c}y_m & {}^{c}y_M \\ {}^{c}z_m & {}^{c}z_M \end{bmatrix} = \begin{bmatrix} {}^{c}\mathbf{c}_x - \frac{l}{2} & {}^{c}\mathbf{c}_x + \frac{l}{2} \\ {}^{c}\mathbf{c}_y - h & {}^{c}\mathbf{c}_y \\ {}^{c}\mathbf{c}_z - \frac{w}{2} & {}^{c}\mathbf{c}_z + \frac{w}{2} \end{bmatrix} \tag{3.12}$$

   Here, ${}^{c}\mathbf{c}$ concerns the vector of coordinates of the Object's centroid in Camera coordinate system, given as location data in its label (see Section 3.1).

2. Taking into account how the Camera coordinates are arranged (Figure 3.1) and the dimensions of the bounding box (width, length and height), the coordinates of the bounding box limits are given by the following matrix:

$${}^{c}Bbox = \begin{bmatrix} {}^{c}x_M & {}^{c}y_M & {}^{c}z_M \\ {}^{c}x_M & {}^{c}y_m & {}^{c}z_M \\ {}^{c}x_m & {}^{c}y_M & {}^{c}z_M \\ {}^{c}x_m & {}^{c}y_m & {}^{c}z_M \\ {}^{c}x_M & {}^{c}y_M & {}^{c}z_m \\ {}^{c}x_M & {}^{c}y_m & {}^{c}z_m \\ {}^{c}x_m & {}^{c}y_M & {}^{c}z_m \\ {}^{c}x_m & {}^{c}y_m & {}^{c}z_m \end{bmatrix}^{\mathbf{T}} = \begin{bmatrix} {}^{c}\mathbf{c}_x - \frac{l}{2} & {}^{c}\mathbf{c}_y - h & {}^{c}\mathbf{c}_z + \frac{w}{2} \\ {}^{c}\mathbf{c}_x + \frac{l}{2} & {}^{c}\mathbf{c}_y - h & {}^{c}\mathbf{c}_z + \frac{w}{2} \\ {}^{c}\mathbf{c}_x - \frac{l}{2} & {}^{c}\mathbf{c}_y - h & {}^{c}\mathbf{c}_z - \frac{w}{2} \\ {}^{c}\mathbf{c}_x + \frac{l}{2} & {}^{c}\mathbf{c}_y - h & {}^{c}\mathbf{c}_z - \frac{w}{2} \\ {}^{c}\mathbf{c}_x - \frac{l}{2} & {}^{c}\mathbf{c}_y & {}^{c}\mathbf{c}_z + \frac{w}{2} \\ {}^{c}\mathbf{c}_x + \frac{l}{2} & {}^{c}\mathbf{c}_y & {}^{c}\mathbf{c}_z + \frac{w}{2} \\ {}^{c}\mathbf{c}_x - \frac{l}{2} & {}^{c}\mathbf{c}_y & {}^{c}\mathbf{c}_z - \frac{w}{2} \\ {}^{c}\mathbf{c}_x + \frac{l}{2} & {}^{c}\mathbf{c}_y & {}^{c}\mathbf{c}_z - \frac{w}{2} \end{bmatrix}^{\mathbf{T}} \tag{3.13}$$

3.3.7 Apply Object's Yaw angle to defined bounding box in Camera coordinate system

In order to subsequently obtain the set of points belonging to the Object under analysis, it is first necessary to apply its orientation to its respective bounding box. See Figure 3.3 for better understanding.

This orientation is given by the value *rotation_y* - defined in the interval $[-\pi, \pi]$ - present in its label and refers to the Yaw angle which, in turn, is given by the rotation of the Object along the Y-axis in Camera coordinate system.

1. Firstly, it is obtained the Object's bounding box coordinates in Object coordinate system.

$${}^{o}BBox = {}^{c}\,BBox - {}^{c}\,\mathbf{c} \tag{3.14}$$

2. In order to apply the Object's orientation to its bounding box, it is first necessary to obtain a homogeneous matrix with the bounding box points, as well as the transformation matrix given by the Yaw angle. These two matrices are defined as follows:

$${}^{o}B\tilde{B}ox = \begin{bmatrix} {}^{o}BBox & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{3.15}$$

$$\mathbf{T}_y^c(\psi) = \begin{bmatrix} \mathbf{R}_y^c(\psi) & 0 \\ 0 & 1 \end{bmatrix} \tag{3.16}$$

Here, $\mathbf{R}_y^c(\psi)$ is defined by the rotation matrix along the Y-axis and the value of the Yaw angle by ψ.

$$\mathbf{R}_y^c(\psi) = \begin{bmatrix} cos(\psi) & 0 & sin(\psi) \\ 0 & 1 & 0 \\ -sin(\psi) & 0 & cos(\psi) \end{bmatrix} \tag{3.17}$$

3. Taking into account equations (3.15), (3.16) and (3.17), the coordinates of the points of the rotated bounding box are obtained by the following homogenised matrix:

$$^{o}B\tilde{B}ox_{\psi} = \mathbf{R}_y^c(\psi) \cdot {}^{o}\,B\tilde{B}ox = \begin{bmatrix} ^{o}BBox_{\psi} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{3.18}$$

Where the matrix of coordinates of the points of the rotated bounding box can be defined as $^{o}BBox_{\psi}$.

4. Finally, it is obtained the new Object's bounding box coordinates back in Camera coordinate system.

$$^{c}BBox_{\psi} =^{o}\, BBox_{\psi} +^{c} \mathbf{c} \tag{3.19}$$

3.3.8 Get Object's points in Camera coordinate system

During this step, all filtered point cloud's points that belong to the Object under analysis are obtained.

1. In order to avoid selecting points that may belong to the real ground, a threshold is arbitrarily defined. Bearing in mind that in Camera coordinate system (see Figure 3.3) the Y-axis points downwards, a new value for the maximum limit in Y is thus obtained.

$$^{c}y_M =^{c}\, BBox_{\psi} \mid_{max\{y\}} -^{c} y_t \tag{3.20}$$

Here, $^{c}BBox_{\psi} \mid_{max\{y\}}$ represents the maximum Y-axis value of Object's orientated bounding box in Camera coordinate system. For the present development, it was defined that $^{c}y_t = 0.1m$.

2. Thereafter, the filtered point cloud is covered in full and each point is evaluated. For a point to be considered as belonging to the Object, it must meet all the conditions presented in Equation 3.21:

$$\begin{cases} p_x \geq^{c} BBox_{\psi} \mid_{min\{x\}} \\ p_x \leq^{c} BBox_{\psi} \mid_{max\{x\}} \\ p_y \geq^{c} BBox_{\psi} \mid_{min\{y\}} \\ p_y \leq^{c} y_{max} \\ p_z \geq^{c} BBox_{\psi} \mid_{min\{z\}} \\ p_z \leq^{c} BBox_{\psi} \mid_{max\{z\}} \end{cases} \tag{3.21}$$

If so, the point will be added to a matrix that will concern the set of all points of the Object. At the same time, a list is also created where the indices of these points are stored, for later obtaining the final manipulated point cloud.

Before going further in the algorithm, a number of at least 11 (or more than 10) point cloud's points belonging to the Object under analysis is arbitrarily established as a valid selection criterion. If this verifies, then the process will continue, where a Pitch angle will be generated for a possible manipulation of the Object. Otherwise, the current Object is ignored and a new iteration of the inner cycle of this algorithm will start.

3.3.9 Generate Pitch angle

After verifying that we are facing an Object of interest, it makes sense to obtain a value for the Pitch angle to apply.

For this, it is used the function *randint* from Python Standard Library's module *Random* [29] where, given a range of integers, it will return an integer belonging to that range. This function presents a discrete uniform distribution.

For the generation of the Pitch angle, an arbitrary range is defined, taking into consideration the study proposed in section 5.1. Subsequently, and for the scope of the mathematical operations performed in the algorithm, the respective Pitch angle value in radians is obtained using the function *radians* from the Python library *NumPy*.

In Chapter 5, a plot with the Pitch values obtained during the manipulation of all objects belonging to the set of point clouds is presented.

Finally, it is important to note that the rest of the algorithm is only executed if the value obtained for the Pitch angle is different from zero. Otherwise, the Object is ignored and a new iteration of the inner cycle will start.

3.3.10 Apply Pitch rotation to Object

In this step the Pitch angle is applied to all points of the Object, obtaining a new set of points characteristic of a rotated object. Taking into account the provided explanation at the end of Section 3.1, an Object doesn't have any horizontal orientation when it's facing the X-axis of the Camera coordinate system. This means that, when the Object is facing exactly the same direction as the subject (vehicle with the sensors), in its coordinate system the front-facing axis will be X and not Z, that will in fact be facing left. In conclusion, this so-called Pitch rotation, when applied in the Object's coordinate system, refers to a rotation along the Z-axis, and not to a rotation along the X-axis in the case of the Camera coordinate system (see Figure 3.3).

1. First, it is obtained the transformation matrix from the Object to the Camera coordinate system, as well as its inverse, which describes a transformation in the opposite direction.

$$ {}^{c}\mathbf{T}_{o} = \begin{bmatrix} \mathbf{R}_{y}^{c}(\psi) & {}^{c}\mathbf{c} \\ 0 & 1 \end{bmatrix} \tag{3.22} $$

$$ {}^{o}\mathbf{T}_{c} = [{}^{c}\mathbf{T}_{o}]^{-1} \tag{3.23} $$

Here, $\mathbf{R}_{y}^{c}(\psi)$ is the Yaw rotation matrix given by equation (3.17) and ${}^{c}\mathbf{c}$ the centroid of the object in Camera coordinate system.

2. Having homogenised the matrices containing the points of the Object and its bounding box, they are obtained in the Object coordinate system, where Pitch rotation will be applied later.

$$ {}^{o}\tilde{Obj} = {}^{o}\,\mathbf{T}_{c} \cdot {}^{c}\,\tilde{Obj} \tag{3.24} $$

$$ {}^{o}\tilde{BBox} = {}^{o}\,\mathbf{T}_{c} \cdot {}^{c}\,\tilde{BBox} \tag{3.25} $$

Here, ${}^{o}\mathbf{T}_{c}$ is given by equation (3.23).

3. The rigid-body transformation matrix defined by the Pitch rotation is also obtained.

$$ {}'\mathbf{T}_{o}(\theta) = \begin{bmatrix} \mathbf{R}_{z}^{c}(\theta)^{\mathbf{T}} & 0 \\ 0 & 1 \end{bmatrix} \tag{3.26} $$

Here, $\mathbf{R}_{z}^{c}(\theta)$ is defined by the rotation matrix along the Z-axis, being θ the Pitch angle. Since the Object coordinate system has the Y-axis pointing downwards (Figure 3.1), the desired rotation matrix will be the matrix transpose of the one defined as follows:

$$ \mathbf{R}_{z}^{c}(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.27} $$

4. It is then possible to apply the Pitch rotation to the matrices obtained in equations (3.24) and (3.25), obtaining new sets of points for the Object and its bounding box, respectively.

$$ {}^{o\prime}\tilde{Obj} = {}'\,\mathbf{T}_{o}(\theta) \cdot {}^{o}\,\tilde{Obj} \tag{3.28} $$

$$ {}^{o\prime}\tilde{BBox} = {}'\,\mathbf{T}_{o}(\theta) \cdot {}^{o}\,\tilde{BBox} \tag{3.29} $$

Here, ${}'\mathbf{T}_{o}(\theta)$ is the transformation matrix defined by equation (3.26).

5. Once the new sets of points are obtained in the Object coordinate system, they are as well obtained

in the Camera coordinate system.

$$^{c\prime}\tilde{Obj} = ^{c}\mathbf{T}_{o} \cdot ^{o\prime}\tilde{Obj} \tag{3.30}$$

$$^{c\prime}\tilde{BBox} = ^{c}\mathbf{T}_{o} \cdot ^{o\prime}\tilde{BBox} \tag{3.31}$$

Here, $^{c}\mathbf{T}_{o}$ is the transformation matrix of the coordinate system from the Object to the Camera, given by equation (3.22).

6. Finally, in order to correct the possible existence of points below the ground - which is physically impossible in the real world - a vertical translation is applied to all points of both sets (see Figure 3.5):

   (a) Firstly, the difference between the maximum value on the Y-axis (perpendicular to the ground) in the set of Object points before applying the pitch rotation and, similarly, the same maximum value after applying said rotation, is calculated;

   $$^{c}y_{offset} = ^{c}BBox\,|_{max\{y\}} - ^{c\prime}BBox\,|_{max\{y\}} \tag{3.32}$$

   (b) Once obtained, this value, defined as an offset on the Y-axis, is added to all the points of the sets obtained by the equations (3.30) and (3.31);

   $$^{c\prime\prime}Obj = ^{c\prime}Obj + \begin{bmatrix}0 & ^{c}y_{offset} & 0\end{bmatrix}^{\mathbf{T}} \tag{3.33}$$

   $$^{c\prime\prime}BBox = ^{c\prime}BBox + \begin{bmatrix}0 & ^{c}y_{offset} & 0\end{bmatrix}^{\mathbf{T}} \tag{3.34}$$

   (c) It is also needed to obtain the Object's centroid new coordinates, in order to update the Object's label. Therefore, it is added to the centroid's Y coordinate the value of the offset on the Y-axis given in (3.32).

   $$^{c\prime}\mathbf{c} = ^{c}\mathbf{c} + \begin{bmatrix}0 & ^{c}y_{offset} & 0\end{bmatrix}^{\mathbf{T}} \tag{3.35}$$

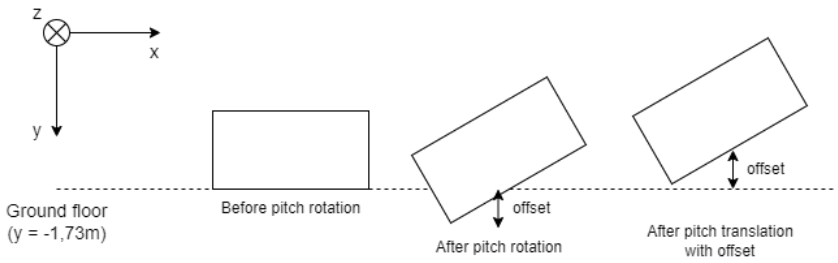


Figure 3.5: Example of a scenario where the pitch rotation applied to an object causes an offset to be applied through a vertical translation.

3.3.11 Get new filtered Point Cloud's points in LiDAR coordinate system

Using the list of indices obtained during Section 3.3.8, the coordinates of the points that were determined to belong to the Object under analysis are replaced in the original point cloud, obtaining the new modified point cloud in Camera coordinate system.

For further tasks such as training, evaluation and testing of the detection and classification model, as well as the visualisation of this final point cloud, it is necessary to obtain it back in the LiDAR coordinate system. This can be done using equation (3.7), thus the homogeneous matrix of the final Point Cloud in LiDAR coordinate system being obtained as follows:

$$ {}^{v}\tilde{\mathbf{P}} = {}^{v}\mathbf{T}_{c} \cdot {}^{c}\tilde{\mathbf{P}} = \begin{bmatrix} {}^{v}\mathbf{P} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{3.36} $$

Here, ${}^{v}\mathbf{P}$ is the point cloud's matrix to which is added a fourth column containing the reflectance value of each point cloud's point, hence obtaining the matrix to be used in the tasks mentioned above.

It is then possible to obtain the final label by updating the Object's centroid coordinates with the new ones and adding the Pitch rotation given by the θ angle value in radians.

After the complete execution of the inner cycle, the final point cloud is obtained, as well as all the new labels referring to its set of annotated objects. Therefore, the two files can be rewritten and the execution of the outer cycle can continue until all point clouds of the dataset have been changed and their objects manipulated.

3.4 Data Augmentation Algorithm: Pseudo-Code

The pseudo-code structure of the data augmentation algorithm, fully detailed in Section 3.3, can be analysed in Algorithm 3.1. This pseudo-code structure also includes other tasks that are ancillary to the main objective of the algorithm and were not previously mentioned and detailed.

Algorithm 3.1: Vehicle orientation and label augmentation

```
for each Point Cloud do
    Load data
    Filter Point Cloud (3.3.1)
    Get Point Cloud's points in Camera coord. system (3.3.2)
    for each labelled Object do
        Get Object data
        if Object is Car OR Van OR Cyclist then
            Get Object's centroid in LiDAR coord. system (3.3.3)
            Define Object's bounding box limits in LiDAR coord. system (3.3.4)
            if Object's bounding box is contained in filtered Point Cloud (3.3.5) then
                Define Object's bounding box limits in Camera coord. system (3.3.6)
                Apply Object's Yaw angle to defined bounding box in Camera coord. system
                 (3.3.7)
                Get Object's points in Camera coord. system (3.3.8)
                if Object is defined by 10 or less points then
                    Ignore Object
                end
                Generate Pitch angle (3.3.9)
                if Pitch angle is equal to 0 then
                    Ignore Object
                else
                    Apply Pitch rotation to Object (3.3.10)
                    Save new points in filtered Point Cloud in Camera coord. system
                    Get new filtered Point Cloud's points in LiDAR coord. system (3.3.11)
                    Save new points in original Point Cloud
                    Get Object's new label
                end
            else
                Ignore Object
            end
        else
            Ignore Object
        end
    end
    Save new Point Cloud data to file
    Save new label data to file
end
```

4

Implementation

Contents

This chapter defines the experimental setup that allows the implementation of the objectives of this work, as well as the subsequent obtaining of results. Firstly, it is indicated the code repository, machine and software that were used to accomplish this work. Next, it is explored the implementation of the object detection model used, YOLOv3 [9]. Following, the tasks of training and testing the models are specified. Finally, the evaluation metrics used in the literature for object detection and angle orientation estimation tasks are described.

4.1 Environment

For the development of this dissertation, a *GitHub* repository [8] has been used that implements the real-time 3D object detection network, Complex-YOLO [3], where the used 2D object detector is a newer version of the one used by the authors, YOLOv3 [9], instead of YOLOv2 [7]. The code in this repository was extended in order to train and test the models obtained as intended, including: the early stopping feature and the Average Orientation Similarity (AOS) performance metric (defined in Section 4.2) were implemented.

4.1.1 Computational Infrastructure

During the development of this work, it was used a machine belonging to the Institute for Systems and Robotics (ISR) whose hardware and software versions are described in Table 4.1.

Graphic Board	Nvidia GeForce GTX 1070ti
Frame Buffer	8GB GDDR5
Processor	Intel i7-8700 @ 3,20 GHz
Operating System	Ubuntu 18.04
Python (version)	3.6.9
Torch (version)	1.1.0
CUDA (version)	10.1

Table 4.1: Specifications of ISR's machine.

The access to this machine was always done remotely, using the *MobaXTerm* software. To transfer data between personal's local machine and ISR's remote machine, the *WinSCP* software was used.

4.1.2 Models: Training and Testing

The original KITTI dataset [4], presented in Section 3.1, consists of two sets: a training set with 7481 samples and a testing set with 7518 samples. Since the testing set does not include ground truths, it will not be used in this work. Therefore, the original training set will be divided into three subsets: training

and validation, used during the training task; evaluation, used for testing purposes. These subsets can be defined as follows:

- Training subset - containing 5236 samples (from sample 000000 to 005235), roughly 70% of the total number of samples;
- Validation subset - including 749 samples (from sample 005236 to 005984), approximately 10% of the total;
- Evaluation subset - consisting of 1496 samples (from 005985 to 007481), about 20% the total number of samples.

In order to validate the datasets obtained through the implemented data augmentation algorithm, object detection models were obtained by adapting an existing code repository for the implementation of the Complex-YOLO detection network [8]. The task of obtaining these models is called training, which must be followed by another task called testing, where performance metrics are obtained that allow the analysis and evaluation of the models and work carried out in general.

The following Table 4.2 describes the training and testing tasks with some relevant information:

Training	**Testing**
Training samples: variable Batch size = 2 samples Gradient accumulation = 2 steps Data augmentation Multi-scale training Evaluation samples = 1496 Early stopping with patience = 10 epochs	Testing samples: variable Batch size = 2 samples Object confidence threshold = 0,5 Intersection over Union (IoU) threshold = 0,5 IoU threshold for Non-Max Suppression (NMS) = 0,5

Table 4.2: Description of training and testing tasks for obtaining and evaluating object detection models.

It is pertinent to further describe and explain some of the characteristics of the tasks described in Table 4.2:

- Gradient accumulation: it is a mechanism used for training a neural network that splits the batch of samples into several mini-batches of samples and runs them sequentially for a fixed number of steps (in this case, two) without updating the model variables. By not updating the variables, the gradients will be accumulated and then used to compute the variable updates before the next sequence;
- Data augmentation: in the implemented code, it consists in either a global rotation of the object around the yaw angle (with a random angle varying between $[-20, 20]$ degrees) or a scaling of the image via a factor which its value will be random and belonging to the range $[0,95; 1,05]$;

- Multi-scale training: as defined in the implemented code, it will set a new image size for every ten batches;

- Early stopping: it is a training feature that allows this task to be automatically stopped whenever a chosen metric has stopped improving, ultimately avoiding overfitting. This feature was implemented as an adaptation to the used code repository and the chosen metric is the validation loss calculated in the end of every epoch. The patience parameter sets the amount of epochs required before the training terminates assuming that the validation loss will not improve over the training loss;

- Object confidence: while classifying detected objects, the models also calculate a confidence score (between 0 and 1) defining how sure they are about the object's class. One object will be defined as belonging to a certain detection class only if its confidence score is greater than the defined threshold;

- NMS: it is a technique mainly used in computer vision methods that allows the models to select a single entity out of many overlapping entities, where the criteria is usually ignoring the ones which their probability bound are below a defined threshold.

4.2 Evaluation Metrics

In order to compare the models obtained during the implementation of the proposals with each other and with the benchmark, quantitative metrics are used. These metrics also make it possible to carry out an in-depth analysis of each model, allowing the assessment of the impact of the proposals of this work.

The trained models will be evaluated for object detection and angle estimation tasks. For each of these, the following official *KITTI Vision Benchmark Suite*'s [30] metrics will be used:

- Object detection - IoU and Average Precision (AP);

- Angle estimation - AOS.

The metrics for object detection are present in the KITTI evaluation levels [31] set (easy, moderate and difficult) which takes into account three detection characteristics: bounding box's minimum height, occlusion's maximum level and maximum truncation.

However, the adapted *GitHub* repository [8] does not have implemented these three levels of evaluation, and as it is not the objective of this dissertation the implementation of evaluation metrics, it will be presented AP values for each class evaluated - 'Car/Van', 'Pedestrian' and 'Bicycle'.

4.2.1 Object Detection

4.2.1.A Intersection over Union

The Jaccard Index, more commonly known as Intersection over Union (IoU), is an essential metric for the object detection task that allows calculating the overlap between predicted and ground truth bounding boxes. Mathematically, this is defined by the quotient between the overlap (*i.e.* intersection) of these two types of bounding boxes with the union of them:

$$IoU = \frac{BBox_{predicted} \cap BBox_{groundTruth}}{BBox_{predicted} \cup BBox_{groundTruth}} \tag{4.1}$$

Here, $BBox_{predicted}$ and $BBox_{groundTruth}$ represent the predicted bounding box and the ground truth bounding box, respectively. Both terms of equation (4.1) define an area or a volume, in the case of object detection in 2D (*e.g.*, BEV) or 3D, respectively.

The KITTI Vision Benchmark Suite uses the PASCAL criteria for the evaluation of object detection performance, where a minimum of 70% overlap for cars and a minimum of 50% overlap for cyclists and pedestrians [31] is required for the detection to be considered as a True Positive (TP).

4.2.1.B Average Precision

The Average Precision (AP) metric, first introduced in the *VOC2007* challenge as the interpolated average precision to evaluate the detection and classification of objects and described in [32], summarises the shape of the Precision/Recall curve, so in order to define this metric it is first necessary to understand the following concepts:

- True Positive (TP), when the model detects an object and classifies it correctly;
- False Positive (FP), when the model detects an object and classifies it with the wrong class;
- False Negative (FN), when the model is unable to detect an existing object;
- Precision, that quantifies the proportion of objects identified as belonging to a certain class that actually belong to the ground truth objects of that class and can be defined by the quotient between the true positives and the total number of detection given by the sum of true positives and false positives;

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

- Recall, that quantifies the proportion of ground truth objects of a class that were correctly identified and can be defined by the quotient between the true positives and the total number of ground truth

given by the sum of true positives and false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

Each detection performed by the model is associated with a score for the level of confidence in that detection. In order to calculate the AP, it is first necessary to calculate the precision and recall for a set of several threshold scores. Then, the Precision/Recall curve is obtained by interpolating the precision at each recall level, r, by getting the maximum value of precision obtained for any recall that respects the condition $\widetilde{r} \geq r$. From this curve, it is chosen a precision value for each recall belonging to a set of eleven equally spaced levels $r \in \{0, 0.1, ..., 0.9, 1.0\}$. Finally, the AP will be obtained by calculating the average of these eleven precision values, through the following equations:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, ..., 0.9, 1.0\}} p_{interp}(r) \tag{4.4}$$

$$p_{interp}(r) = \max_{\widetilde{r}:\widetilde{r} \geq r} p(\widetilde{r}) \tag{4.5}$$

By defining the AP in this way, models are required to have high precision at all levels of recall in order to have a relevant AP-score. On the other hand, models that show high accuracy in a small subset of recall levels (usually the lowest) will be penalised.

4.2.2 Angle Estimation

4.2.2.A Average Orientation Similarity

The Average Orientation Similarity (AOS) metric was first introduced in [30] and evaluates the 3D orientation of the detection which is given by the rotation angle around the Y-axis, yaw, in the Camera coordinate system [4] (Figure 3.1). This metric is based on the AP metric defined in equation (4.4) where, instead of calculating the precision for each threshold score, the orientation similarity, $s \in [0, 1]$, is calculated. Similarly to AP, for this metric, the average of orientations similarities for recalls belonging to the set of levels $r \in \{0, 0.1, ..., 0.9, 1.0\}$ is calculated at the end using the following equation:

$$AOS = \frac{1}{11} \sum_{r \in \{0, 0.1, ..., 0.9, 1.0\}} \max_{\widetilde{r}:\widetilde{r} \geq r} s(\widetilde{r}) \tag{4.6}$$

Here, the orientation similarity $s \in [0, 1]$ at recall r is a ($[0...1]$) normalised variant of the cosine similarity [30] defined by the following equation:

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + \cos \Delta_\theta^{(i)}}{2} \delta_i \tag{4.7}$$

Here, $D(r)$ indicates the set of all object detections at recall rate r, $\Delta_\theta^{(i)}$ indicates the difference in angle between predicted and ground truth orientation of detection i and δ_i is a variable used to penalise multiple detections which explain a single object: $\delta_i = 1$ if detection i has been assigned to a ground truth bounding box (where the overlap value is at least 50%) and $\delta_i = 0$ otherwise [30].

4.2.3 Frame Rate

In the context of autonomous driving and its respective object detection task, taking into account that it implies a constantly moving environment that can occur at high speeds, it is relevant to infer the obtained models about its frame rate, *i.e.* Frames Per Second (Frames Per Second (FPS)). This metric is given by the inverse of the inference time, and can be defined by the following equation:

$$FPS = \frac{1}{inferenceTime} \tag{4.8}$$

Here, inference time is defined as the time required for the model to complete a forward propagation, given a specific input - in the context of this work, BEV RGB-maps. Therefore, through this FPS metric it is possible to determine, for a given speed, the average distance travelled between each instant (or processing of an input by the model under evaluation).

5

Results

Contents

This chapter proposes and defines three studies in order to better evaluate the developed data augmentation algorithm. For that, presents and discusses accordingly the results obtained for each one.

5.1 Studies and Datasets

In order to validate the developed and implemented ground truth manipulation algorithm, three possible studies referring to different types of analysis are specified below:

- Study 1 - tries to recreate cities located on hills, such as Lisbon (Portugal) [5] or San Francisco (USA) [6], suggesting streets with different levels of slope contained in the range of $[-30, 30]$ degrees, defined through the pitch angle. Likewise, tries to recreate cities located in plain terrains, such as Amsterdam (Netherlands) [33] or Leeds (UK) [34], proposing streets with similar and low levels of slope contained in the range of $[-5, 5]$ degrees. Compares these two scenarios with the baseline (original, non-augmented, KITTI dataset [4]) in order to evaluate the impacts of this augmentation in the performance of object detection, as well as the regression of the yaw orientation;

- Study 2 - It compares the object detection performance and the yaw orientation regression, as well as the data processing speed, between the two typologies (Normal and Tiny) available for the YOLOv3 neural network, included in the Complex-YOLO object detection model;

- Study 3 - Just for the Normal typology of the YOLOv3 neural network, it analyses in detail the performance metrics obtained for the three most used detection classes in the literature: 'Car/Van', 'Pedestrian' and 'Cyclist'.

For the task of detecting objects within the scope of autonomous driving systems, the literature focuses mainly on three classes: 'Car' (which usually also includes objects of class 'Van'), 'Cyclist' and 'Pedestrian'. However, in the current work that proposes the variation of the inclination of objects in the driving environment, this type of rotation does not apply to the 'Pedestrian' class - for reasons of movement and balance, human beings will instinctively always present an approximate zero inclination. Bearing this in mind, this work will apply the developed algorithm to objects belonging to detection classes 'Car', 'Van' and 'Cyclist', ignoring the objects of detection class 'Pedestrian'.

Regarding the two datasets that will be created, they can be described as follows:

- High Slope dataset - the augmented objects will be manipulated with a pitch angle varying between $[-30, 30]$ degrees;

- Low Slope dataset - the augmented objects will be manipulated with a pitch angle varying between $[-5, 5]$ degrees.

It is relevant to state that these will be validated and compared by the studies through the AP metrics (defined in Section 4.2) referring to the yaw angle regression that is implemented in the literature and in the object detection network used [3] [8], since the implementation of the regression for the pitch angle is not foreseen in the objectives of this work.

In Figure 5.1 it can be seen the histograms of the pitch angles generated and used to manipulate the objects during the creation of the two proposed datasets. Analysing these, it is possible to notice a tendency towards a uniform discrete distribution of values, as desired. On the other hand, in Table 5.1 is presented a summary of the datasets creation process.

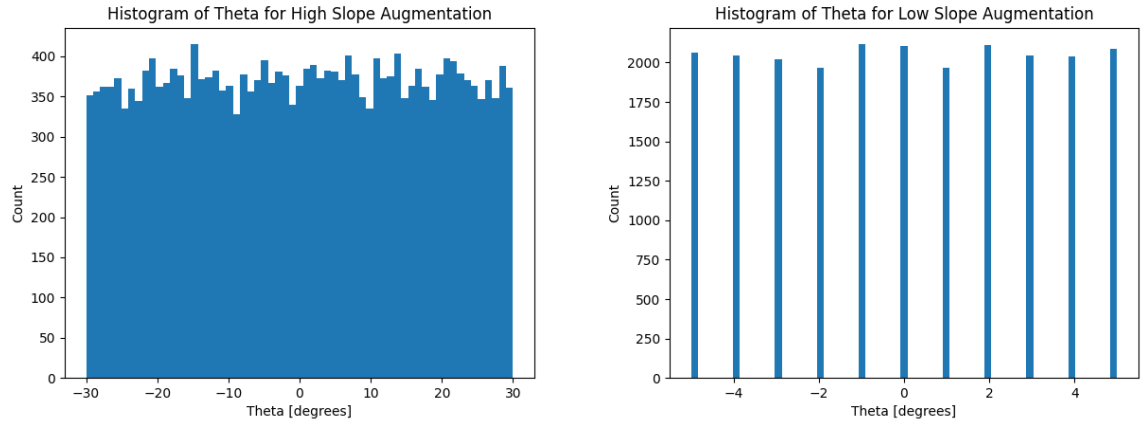


Figure 5.1: Histograms of pitch angles generated during the creation of each dataset: High Slope (on the left) and Low Slope (on the right).

Dataset	Total execution time [hh:mm:ss]	Total number of manipulated objects
High Slope	03:27:17	22567
Low Slope	03:27:25	22567

Table 5.1: Summary of datasets creation process.

In Appendix A are presented some examples of figures including manipulated ground truths.

5.2 Study 1: Vehicle Detection Accuracy Analysis

In order to better evaluate the behaviour of the object detection models obtained (one for each dataset), the performance metrics for each of them were obtained in the respective testing sets of the three available datasets (Baseline - official KITTI dataset [4], High Slope and Low Slope). Table 5.2 presents all the average values of each performance metric obtained for each possible scenario, as previously described.

Testing set	Model	mAP [%]	mAOS [%]
Baseline	Baseline	**76,51**	**72,70**
	Low Slope	74,23	72,41
	High Slope	71,90	67,88
Low Slope	Baseline	**77,12**	**74,05**
	Low Slope	74,13	72,19
	High Slope	72,78	65,14
High Slope	Baseline	**73,68**	68,23
	Low Slope	72,44	**68,42**
	High Slope	70,61	66,35

Table 5.2: mAP and mAOS values of the three models when evaluated in the three different testing sets. In this Study, only the Normal typology is evaluated.

When analysing Table 5.2, it is possible to conclude that the Baseline model presented the best metrics in the three testing sets. Taking a closer look, it is possible to notice that the only exception in this pattern appears in the High Slope testing set, where the best mean AOS value is obtained for the Low Slope model, even though it is a very similar value to the one obtained for the Baseline model.

It is possible to notice that the values of the performance metrics decrease as the variation of the pitch angles in the manipulated objects increases: the High Slope model is the one that presents the lowest performance metrics in all three testing sets. However, in the case of the mAP metric (see third column of Table 5.2, the performance loss oscillates between 3% (difference between the extreme values in the High Slope dataset) and 5% (difference between the extreme values in the Baseline dataset), which means it is not a considerable loss. In the case of the mAOS metric (see fourth column of Table 5.2), the performance loss can be considered, oscillating between 2% (difference between the extreme values in the High Slope dataset) and 9% (difference between the extreme values in the Low Slope dataset).

In conclusion, a generalised loss of performance by the models was expected, since a substantial modification was being introduced to the objects (which could even be considered as noise data). However, the loss values do not prove to be critical or conditioning: overall, the three models maintain a good object detection capability, especially when compared with the literature in a real-time 3D vehicle detection perspective.

5.3 Study 2: Computational Performance Analysis

Similar to what is done in Section 5.2, this study adds the detection performance results obtained for the other typology available in YOLOv3 [9], Tiny.

By analysing Table 5.3, what was already expected is directly confirmed: the Normal typology presents better values for the detection performance and, on the other hand, the Tiny typology presents better values for the computational performance.

Similar to what was concluded in Section 5.2, the Tiny typology presents an increasing loss of per-

Testing set	Model	Typology	mAP [%]	mAOS [%]	Frame Rate [fps]			Execution time [hh:mm:ss]	Time gain (Tiny vs. Normal)
					Min.	Max.	Avg.		
Baseline	Baseline	Normal	**76,51**	**72,70**	0,65	16,09	3,52	00:02:50	2,66
		Tiny	65,23	62,27	**4,01**	**87,45**	**13,01**	**00:01:04**	
	Low Slope	Normal	**74,23**	**71,05**	1,74	15,40	3,89	00:02:17	2,14
		Tiny	59,37	56,93	**3,85**	**109,10**	**13,12**	**00:01:04**	
	High Slop	Normal	**71,90**	**67,88**	0,65	15,26	3,26	00:03:04	2,92
		Tiny	58,28	56,33	**4,44**	**110,80**	**13,45**	**00:01:03**	
Low Slope	Baseline	Normal	**77,12**	**73,23**	0,65	16,01	3,40	00:02:51	2,71
		Tiny	65,15	62,14	**3,18**	**91,09**	**13,01**	**00:01:03**	
	Low Slope	Normal	**74,13**	**71,02**	1,44	15,99	3,94	00:02:18	2,16
		Tiny	59,02	56,53	**3,19**	**110,30**	**13,12**	**00:01:04**	
	High Slope	Normal	**72,78**	**68,56**	0,64	15,98	3,24	00:03:06	3,05
		Tiny	57,96	56,04	**5,01**	**110,50**	**13,46**	**00:01:01**	
High Slope	Baseline	Normal	**73,68**	**68,23**	0,62	14,93	3,48	00:02:54	2,85
		Tiny	61,92	57,91	**3,39**	**87,95**	**13,24**	**00:01:01**	
	Low Slope	Normal	**72,44**	**68,42**	1,78	16,01	3,96	00:02:15	2,18
		Tiny	55,90	52,29	**3,75**	**111,60**	**13,45**	**00:01:02**	
	High Slope	Normal	**70,61**	**66,35**	0,66	14,27	3,25	00:03:04	3,02
		Tiny	57,48	55,58	**3,97**	**111,50**	**13,52**	**00:01:01**	

Table 5.3: A thorough comparison between all the obtained models for both available typologies: Normal and Tiny.

formance in the detection metrics, in relation to the variety of angles applied to the manipulated objects. In this case, it is already quite significant, where the performance loss varies similarly for both mean AP and mean AOS (see fourth and fifth columns of Table 5.3), ranging between 10% (difference between extreme values for Baseline's duo testing set / model and mAOS metric) and 17% (difference between extreme values for High Slope's duo testing set / model and mAP metric).

However, in terms of computational performance, the Tiny typology is far superior to Normal. In fact, the average values of Frame Rate for the Normal typology are similar to the minimum values for the Tiny typology: between 3 fps and 4 fps (see Table 5.3, columns Min. and Avg. for Frame Rate). The Tiny typology has very high maximum Frame Rate values: between 85 fps and 110 fps. Lastly, considering the last column of Table 5.3, it is noted that the Tiny typology has a data processing speed approximately 2.1 (minimum value of time gain) to 3 (maximum value) times higher than the Normal typology.

Although the performance values for the detection of objects of the Tiny typology deviate from the benchmark in the literature, this cannot be left out of consideration. In fact, this typology guarantees the task of detecting 3D vehicles in real-time with relative success regardless of the available computational infrastructure, and especially when this is of lower computational capacity.

5.4 Study 3: Class-Specific Detection Analysis

In this last Study, Table 5.4 shows the values obtained for the object detection metrics for each of the three most used classes in the literature. For simplicity, during this Study only models trained for the Normal typology were analysed.

When analysing Table 5.4, it is possible to notice that, in general, the values obtained for the two met-

Testing set	Model	AP [%]			AOS [%]		
		Car/Van	Pedestrian	Cyclist	Car/Van	Pedestrian	Cyclist
Baseline	Baseline	**96,03**	**59,97**	**73,53**	**95,56**	**50,42**	72,13
	Low Slope	95,48	54,36	72,84	95,21	45,53	**72,41**
	High Slope	93,82	57,93	63,94	93,38	47,44	62,82
Low Slope	Baseline	**95,93**	**59,91**	**75,51**	**95,44**	**50,20**	**74,05**
	Low Slope	95,37	54,41	72,61	95,15	45,72	72,19
	High Slope	93,80	57,72	66,83	93,30	47,24	65,14
High Slope	Baseline	93,36	**60,24**	67,43	89,86	**50,32**	64,50
	Low Slope	93,23	55,73	**68,37**	91,11	46,71	**67,45**
	High Slope	**93,67**	58,01	60,16	**92,94**	47,37	58,74

Table 5.4: Values for AP and AOS for single detection classes 'Car/Van', 'Pedestrian' and 'Cyclist'. In this Study, only the Normal typology is evaluated.

rics decrease with the increase of the variety of objects, commonly among the three detection classes.

Still referring to Table 5.4, it is noted that the 'Car/Van' detection class has the highest performance values. Oscillating between 93% and 96% for the AP metric and between 89% and 95% for the AOS metric, these values are quite promising.

Regarding the 'Pedestrian' and 'Cyclist' detection classes, the obtained values for detection performance are lower, especially for 'Pedestrian'. Bearing in mind that the increasing order of object size is 'Pedestrian', 'Cyclist' and 'Car/Van', performance metrics suggest that there is a direct relationship between size and detection performance, where this will be all the better the larger the size of the object in question.

On the other hand, it is known from Table 5.5 that the universe of manipulated objects is rather unbalanced: the number of objects of the 'Car/Van' class is approximately 17.5 times higher than the number referring to the 'Cyclist' class (as explained in Section 5.1, 'Pedestrian' class objects were not subjected to the data augmentation algorithm). This is also a factor to consider as a possible explanation for a lower detection performance in the 'Cyclist' class.

Detection Class	Total number of augmented objects	Objects' height [m]		
		Minimum	Maximum	Average
Car/Van	21350	1,14	2,91	1,56
Cyclist	1217	1,41	2,09	1,74

Table 5.5: Summary for detection classes 'Car/Van' and 'Cyclist' obtained during the creation of the datasets.

6

Conclusion

Contents

This last chapter presents the conclusion of the present thesis' dissertation and it is divided in two sections. Firstly, it is given an overview of the contributions and achievements carried during this work. Lastly, it is suggested some future steps of this work and ways of improving it.

6.1 Contributions

For the execution of this work, two main objectives were defined: the implementation of a data augmentation algorithm capable of manipulating the ground truth of the KITTI dataset [4] in order to introduce a variation in its vertical orientation; and sharing with the scientific community the code repository developed to achieve the objective described above (to obtain the datasets and its respective models, as well as to evaluate their performance).

Since the main datasets in the existing literature for the real-time detection of multi-class objects through object detection do not include information regarding the pitch angle and that in the literature there is no data augmentation method for this angle, the achievement of this objective presents a relevant contribution for the scientific community in the field of autonomous driving systems.

Having presented a detailed description of the algorithm developed in Section 3.3 and it being available in a code repository (https://github.com/joaomlf/kittidataset_pitchangleaugmentation), it is possible to confidently state that the objectives proposed by this work were achieved. Furthermore, through the studies carried out and presented in Chapter 5, it can be affirmed that the addition of this variation in the objects does not have a negative impact on the correct detection and classification to the point of being considered a setback or impediment to continue investigating this topic.

6.2 Future Work

Having achieved the objectives proposed by this work, some examples of possible future work are provided below:

- Adapting (and possibly improving) this algorithm to other datasets of the literature used for object detection, as well as for semantic segmentation, with a view in reducing possible errors that the one developed can create while defining points as belonging to a certain object. This is an acknowledged possibility, as there may be human error during the annotation of objects or because they can possibly be occluded by other elements or objects present in the driving environment;

- Implementing or expanding object detection networks to include pitch angle regression. In the case of *Complex-YOLO* [3], since it was used in this work, there can be the possibility of replicating the loss function already implemented for the regression of the yaw angle, but now for the pitch angle.

Another possibility to be explored could be the regression of the two angles as a single set using quaternions (this option is already applied in the *nuScenes* dataset [35]) or a spherical coordinate system.

Bibliography

[1] Y. Li and J. Ibanez-Guzman, “Lidar for Autonomous Driving: The principles, challenges, and trends for automotive lidar and perception systems,” 4 2020. [Online]. Available: http://arxiv.org/abs/2004.08467http://dx.doi.org/10.1109/MSP.2020.2973615

[2] M. Hahner, D. Dai, A. Liniger, and L. Van Gool, “Quantifying Data Augmentation for LiDAR based 3D Object Detection,” 4 2020. [Online]. Available: http://arxiv.org/abs/2004.01643

[3] M. Simon, S. Milz, K. Amende, and H.-M. Gross, “Complex-YOLO: Real-time 3D Object Detection on Point Clouds,” 3 2018. [Online]. Available: http://arxiv.org/abs/1803.06199

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets Robotics: The KITTI Dataset,” Karlsruhe Institute of Technology, Tech. Rep., 2012. [Online]. Available: http://www.cvlibs.net/publications/Geiger2013IJRR.pdf

[5] rosamfelix, “rosamfelix/gis/declives/DeclivesLisboa.” [Online]. Available: https://web.tecnico.ulisboa.pt/~rosamfelix/gis/declives/DeclivesLisboa.html

[6] u/JPPlus, “Map of San Francisco Bike Paths Showing Slope.” [Online]. Available: https://www.reddit.com/r/BAbike/comments/mr4wy6/map_of_san_francisco_bike_paths_showing_slope/

[7] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” University of Washington, Tech. Rep., 7 2017. [Online]. Available: http://pjreddie.com/yolo9000/

[8] ghimiredhikura, “ghimiredhikura/Complex-YOLOv3: PyTorch implementation of Complex-YOLO paper with YoloV3.” [Online]. Available: https://github.com/ghimiredhikura/Complex-YOLOv3

[9] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” 4 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[10] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, “Object Detection with Deep Learning: A Review,” pp. 3212–3232, 11 2019.

[11] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, “Multi-View 3D Object Detection Network for Autonomous Driving,” Tsinghua University, Tech. Rep., 2017.

[12] B. Yang, W. Luo, and R. Urtasun, “PIXOR: Real-time 3D Object Detection from Point Clouds,” Uber Advanced Technologies Group, Tech. Rep., 2018.

[13] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “PointPillars: Fast Encoders for Object Detection from Point Clouds,” nuTonomy, Tech. Rep., 2019. [Online]. Available: https://github.com/nutonomy/second.pytorch

[14] Y. Yan, Y. Mao, and B. Li, “Second: Sparsely embedded convolutional detection,” *Sensors (Switzerland)*, vol. 18, no. 10, 10 2018.

[15] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, “PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection,” The Chinese University of Hong Kong, Tech. Rep., 2020.

[16] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum PointNets for 3D Object Detection from RGB-D Data,” Stanford University, Tech. Rep., 2018.

[17] S. Shi, X. Wang, and H. Li, “PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud,” The Chinese University of Hong Kong, Tech. Rep., 2019. [Online]. Available: https://github.com/sshaoshuai/PointRCNN.

[18] Z. Yang, Y. Sun, S. Liu, X. Shen, J. Jia, and Y. Lab, “STD: Sparse-to-Dense 3D Object Detector for Point Cloud,” YouTu Lab, Tencent, Tech. Rep., 2019.

[19] Y. Zhou, P. Sun, Y. Zhang, D. Anguelov, J. Gao, T. Ouyang, J. Guo, J. Ngiam, and V. Vasudevan, “End-to-End Multi-View Fusion for 3D Object Detection in LiDAR Point Clouds,” Waymo LLC, Tech. Rep., 2020.

[20] Y. Wang, A. Fathi, A. Kundu, D. Ross, C. Pantofaru, T. Funkhouser, and J. Solomon, “Pillar-based Object Detection for Autonomous Driving,” 7 2020. [Online]. Available: http://arxiv.org/abs/2007.10323

[21] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, “nuScenes: A multimodal dataset for autonomous driving,” 3 2019. [Online]. Available: http://arxiv.org/abs/1903.11027

[22] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, “Argoverse: 3D Tracking and Forecasting with Rich Maps,” 11 2019. [Online]. Available: http://arxiv.org/abs/1911.02620

[23] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, J. Hays, A. Ai, and G. Tech, "Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting," Argo AI, Tech. Rep., 2021. [Online]. Available: https://github.com/argoai/argoverse-api

[24] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, S. Zhao, S. Cheng, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," 12 2019. [Online]. Available: http://arxiv.org/abs/1912.04838

[25] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, "Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark," 4 2017. [Online]. Available: http://arxiv.org/abs/1704.03847

[26] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," 4 2019. [Online]. Available: http://arxiv.org/abs/1904.01416

[27] Y. Liao, J. Xie, and A. Geiger, "KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D," 9 2021. [Online]. Available: http://arxiv.org/abs/2109.13410

[28] bostondiditeam, "KITTI dataset - README." [Online]. Available: https://github.com/bostondiditeam/kitti/blob/master/resources/devkit_object/readme.txt

[29] Python Software Foundation, "The Python Standard Library, module random — Generate pseudo-random numbers." [Online]. Available: https://docs.python.org/3/library/random.html

[30] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," Karlsruhe Institute of Technology, Tech. Rep., 2012. [Online]. Available: http://www.cvlibs.net/publications/Geiger2012CVPR.pdf

[31] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, "The KITTI Vision Benchmark Suite." [Online]. Available: http://www.cvlibs.net/datasets/kitti/eval_object.php

[32] M. Everingham, L. Van Gool, C. K I Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," University of Leeds, Tech. Rep., 2009. [Online]. Available: https://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf

[33] rosamfelix, "rosamfelix/gis/declives/SlopesAmsterdam." [Online]. Available: https://web.tecnico.ulisboa.pt/~rosamfelix/gis/declives/SlopesAmsterdam.html

[34] ——, “rosamfelix/gis/declives/SlopesLeeds.” [Online]. Available: https://web.tecnico.ulisboa.pt/~rosamfelix/gis/declives/SlopesLeeds.html

[35] Caesar HBankiti VLang AVora SLiong VXu QKrishnan APan YBaldan GBeijbom O, “nuScenes - Data format.” [Online]. Available: https://www.nuscenes.org/nuscenes#data-format

A

Examples of augmented objects

The following images represent the objects referred in Tables A.1 and A.2 of Section 5.1. These objects were obtained through segmentation of the point clouds to which they belong, where the boundaries of their bounding boxes are defined through equations 3.12 and 3.13. Subsequently, the respective rotations (yaw and pitch) are applied through equations 3.19, 3.31 and 3.34.

Figure	Class	Pitch angle [deg]
A.1	Car/Van	10
A.2		30
A.3		-16
A.4		-23
A.5	Cyclist	17
A.6		30
A.7		-8
A.8		-21

Table A.1: List of figures representing manipulated objects during the execution of Algorithm 3.1 for obtaining the High Slope dataset.

Figure	Class	Pitch angle [deg]
A.9	Car/Van	4
A.10		-5
A.11	Cyclist	2
A.12		-2

Table A.2: List of figures representing manipulated objects during the execution of Algorithm 3.1 for obtaining the Low Slope dataset.

The green bounding box refers to the ground truth (where the yaw rotation is included) and the red bounding box refers to the augmentation (where the pitch rotation is applied). For better visualisation, the points belonging to the object in question are represented by white spheres of arbitrary volume.

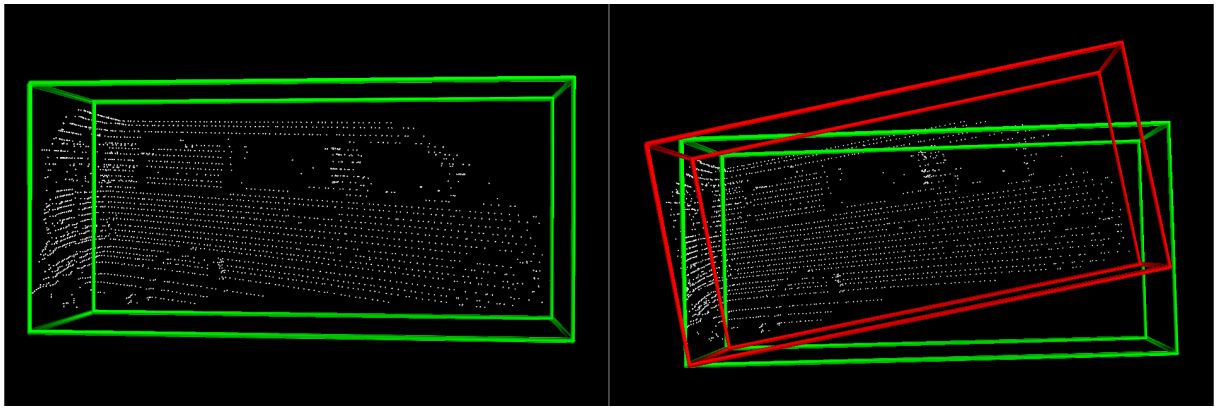


Figure A.1: Example of augmented object. Dataset: High Slope. Class: Car/Van. Pitch angle = 10 degrees.

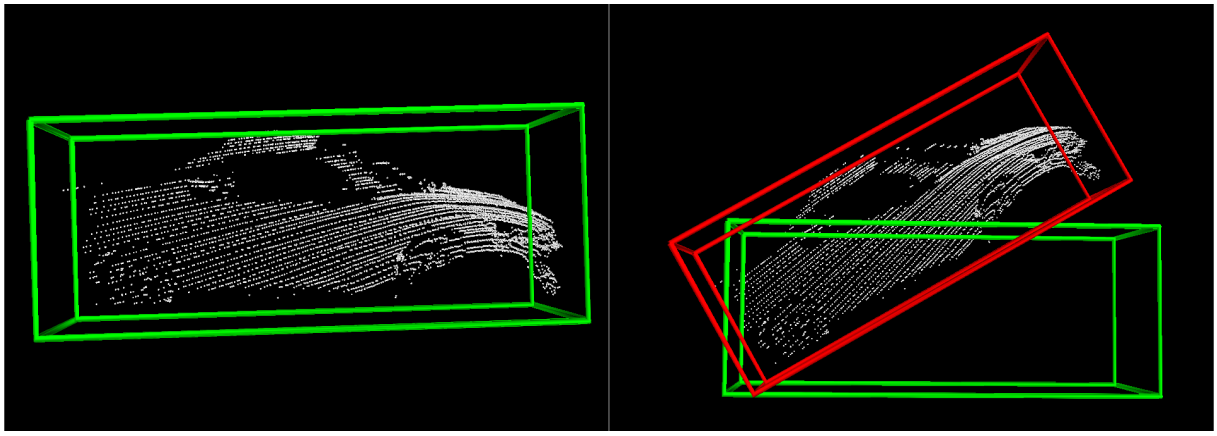


Figure A.2: Example of augmented object. Dataset: High Slope. Class: Car/Van. Pitch angle = 30 degrees.

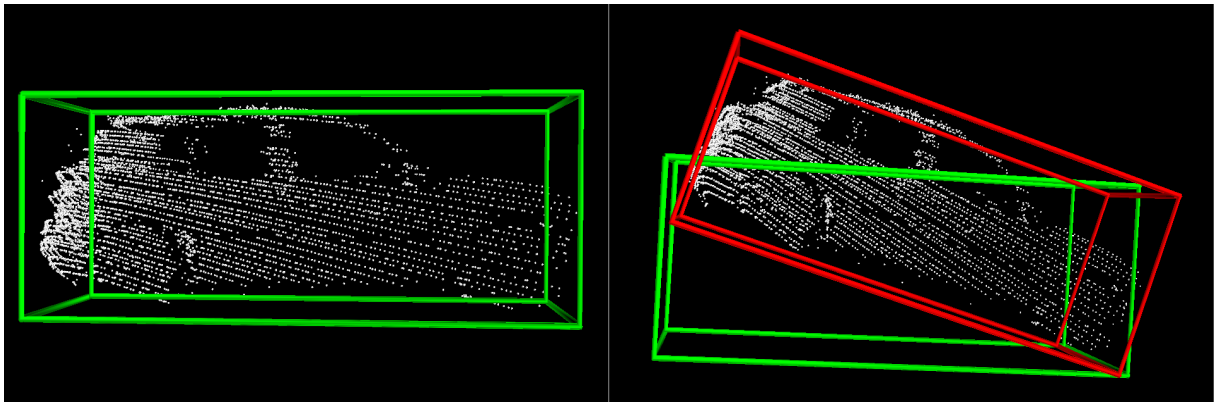


Figure A.3: Example of augmented object. Dataset: High Slope. Class: Car/Van. Pitch angle = -16 degrees.

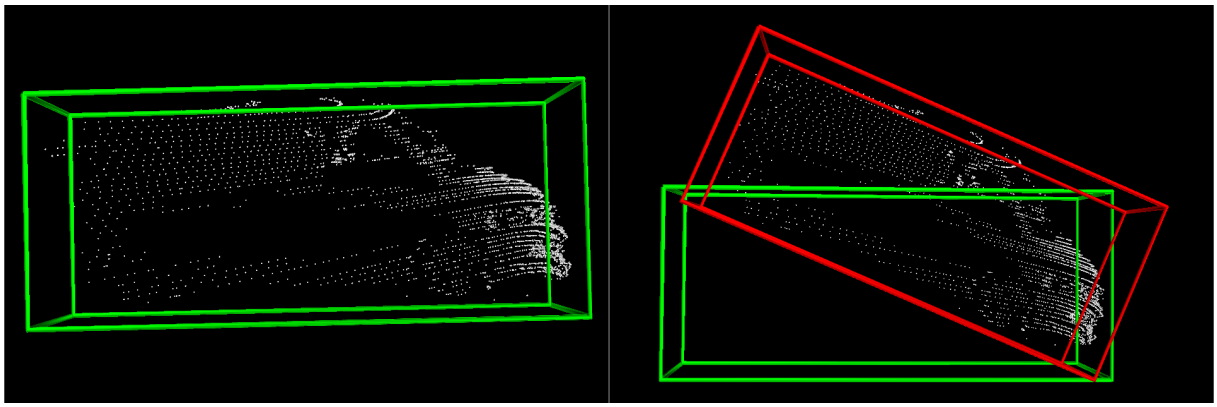


Figure A.4: Example of augmented object. Dataset: High Slope. Class: Car/Van. Pitch angle = -23 degrees.

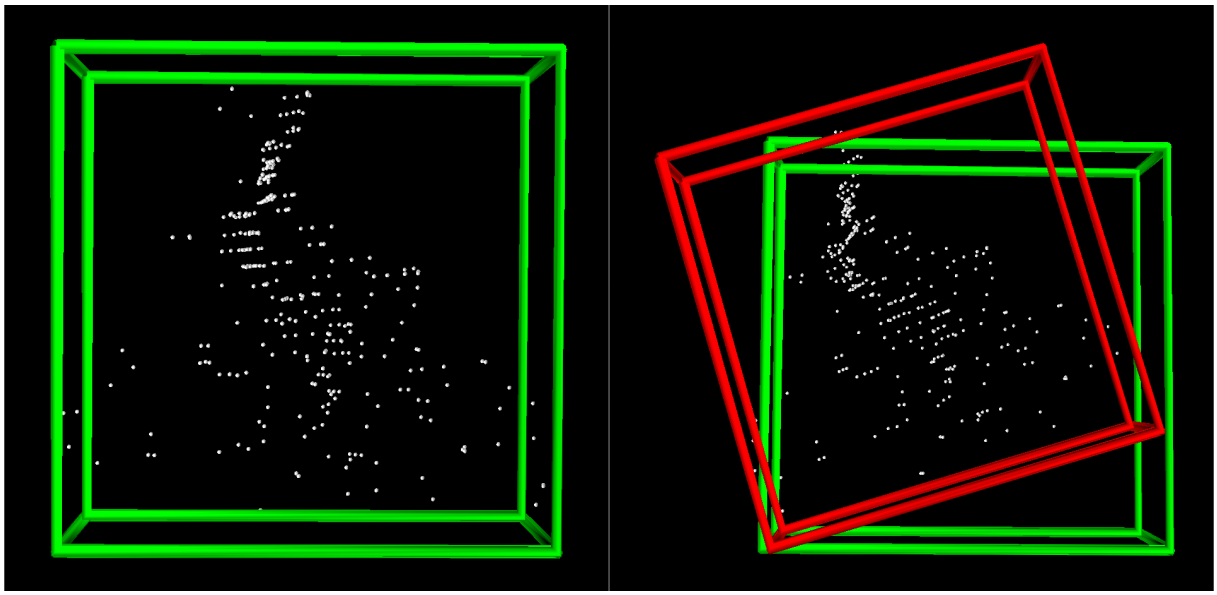


Figure A.5: Example of augmented object. Dataset: High Slope. Class: Cyclist. Pitch angle = 17 degrees.

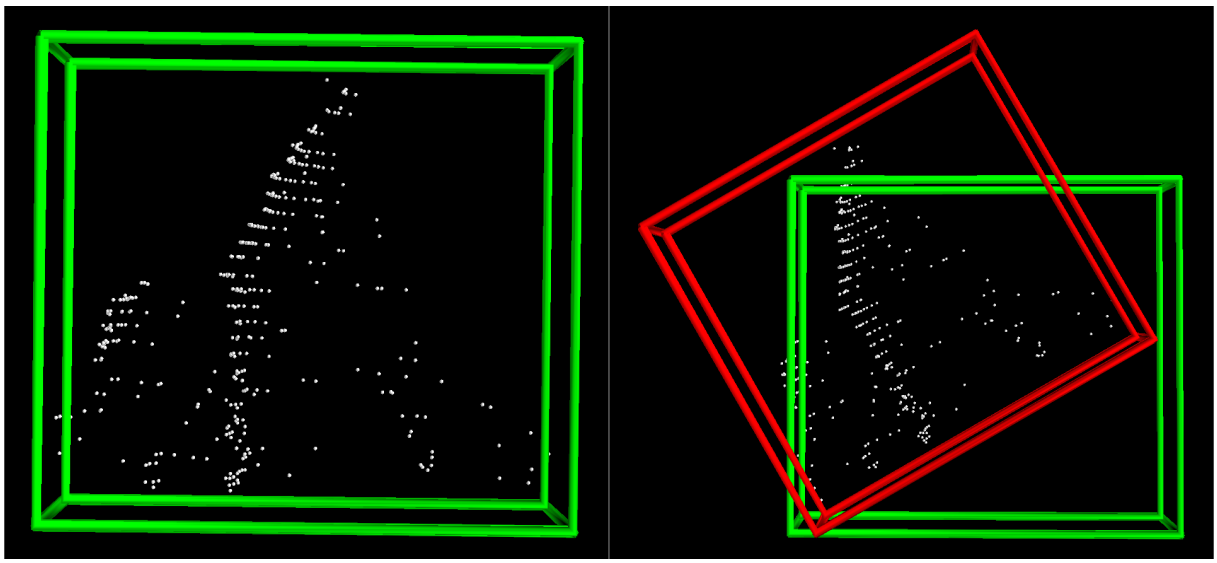


Figure A.6: Example of augmented object. Dataset: High Slope. Class: Cyclist. Pitch angle = 30 degrees.

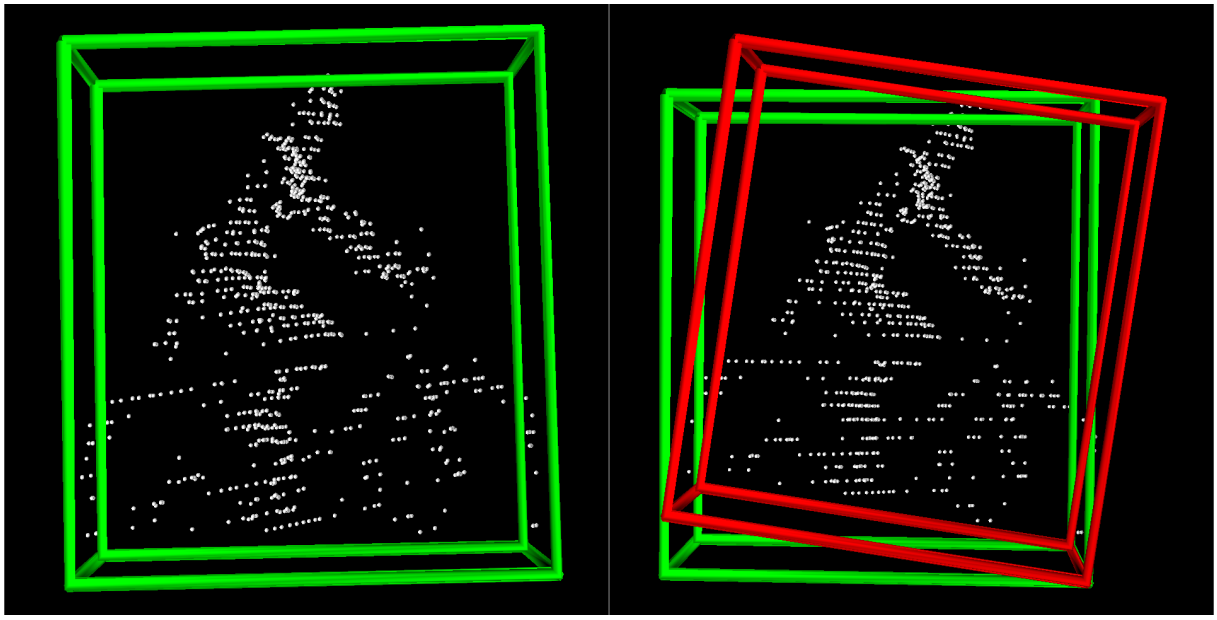


Figure A.7: Example of augmented object. Dataset: High Slope. Class: Cyclist. Pitch angle = -8 degrees.

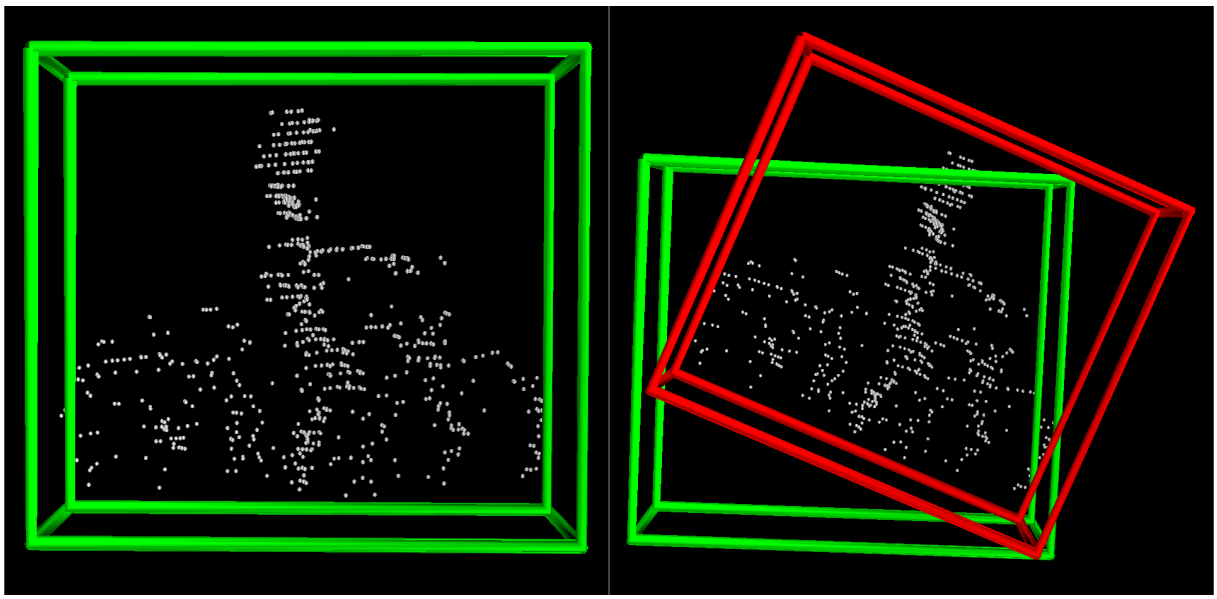


Figure A.8: Example of augmented object. Dataset: High Slope. Class: Cyclist. Pitch angle = -21 degrees.

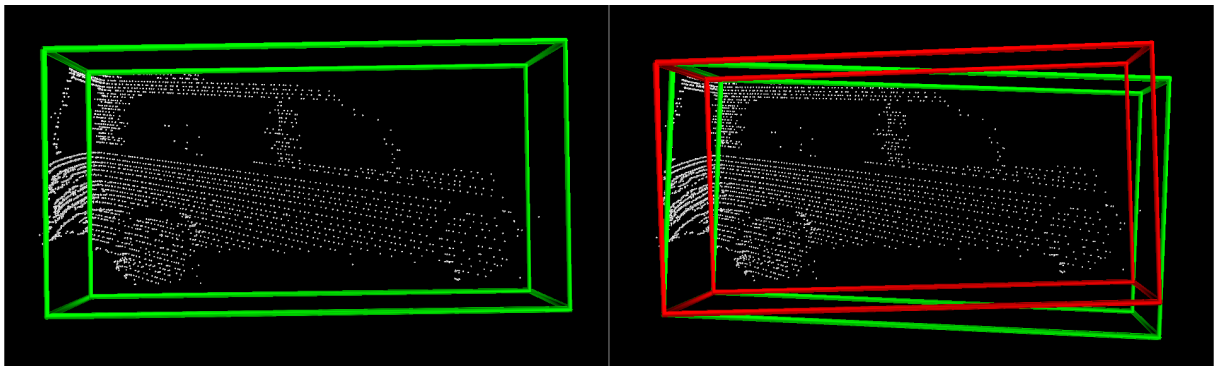


Figure A.9: Example of augmented object. Dataset: Low Slope. Class: Car/Van. Pitch angle = 4 degrees.

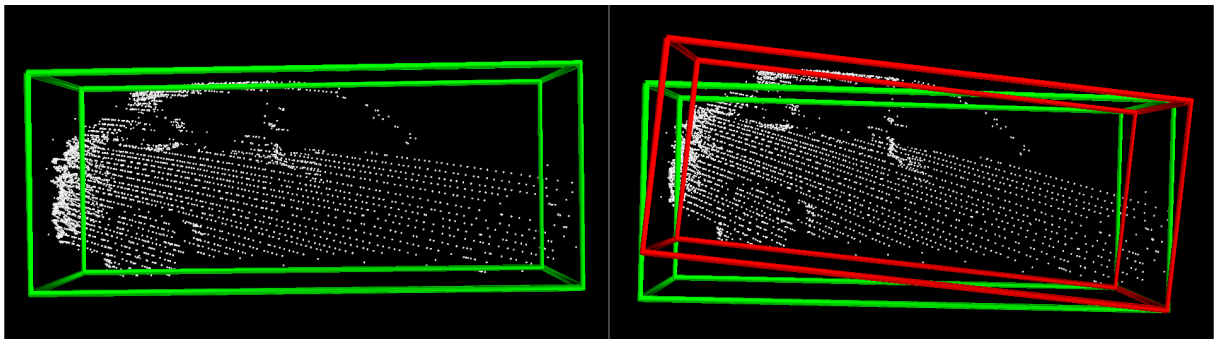


Figure A.10: Example of augmented object. Dataset: Low Slope. Class: Car/Van. Pitch angle = -5 degrees.

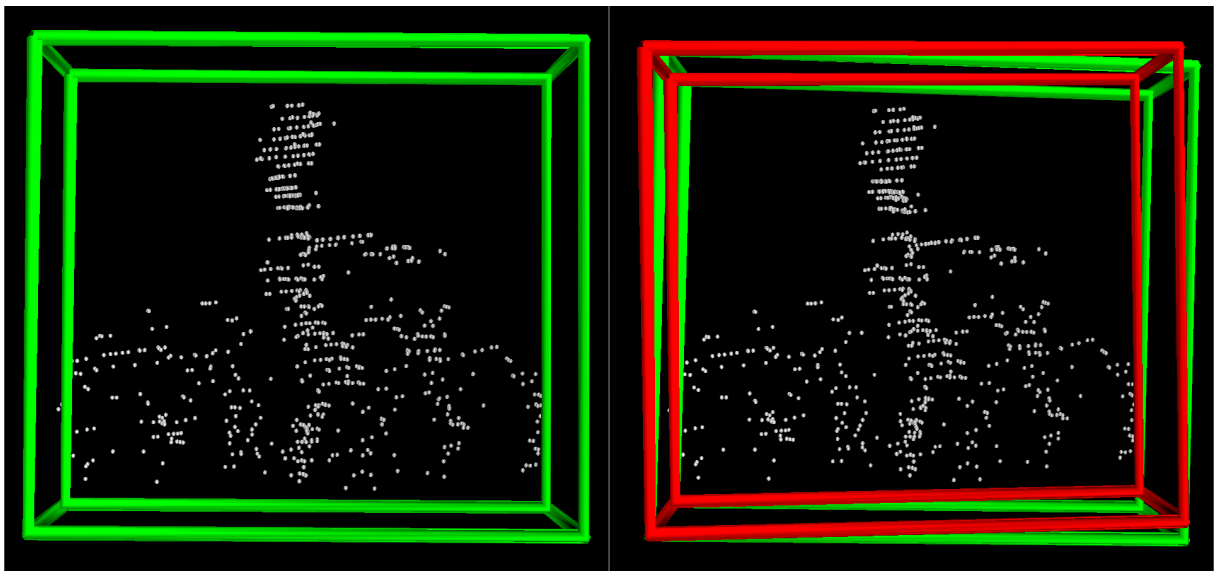


Figure A.11: Example of augmented object. Dataset: Low Slope. Class: Cyclist. Pitch angle = 2 degrees.

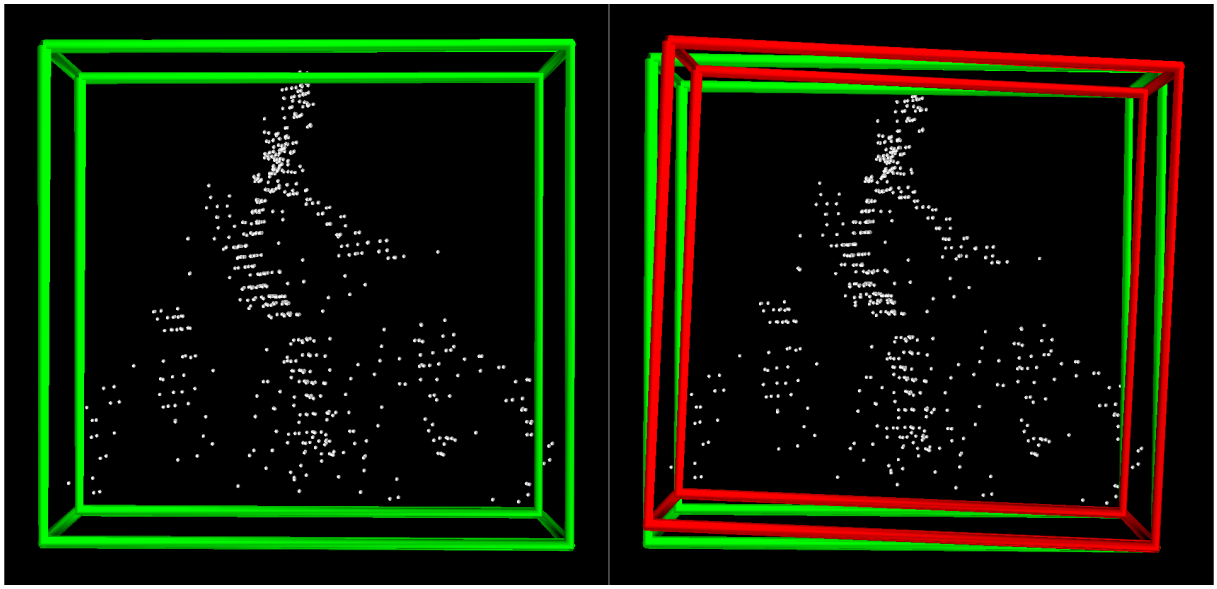


Figure A.12: Example of augmented object. Dataset: Low Slope. Class: Cyclist. Pitch angle = -2 degrees.