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

João Miguel de Loureiro Ferreira
joao.loureiro.ferreira@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

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.

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

I. Introduction

A. Motivation

The global number of annual road accidents brings the urgent need to make driving increasingly safer. Thus, autonomous driving systems that aim to develop cars capable of reducing the need for human intervention have emerged. Among them, one of the main components is real-time 3D object detection, where accurate object localisation is essential for route planning, as well as collision avoidance. In recent years, several datasets have appeared in the autonomous driving scientific community, mainly using the Light Detection And Ranging (LiDAR) [1] sensor technology that is able to map the surrounding environment through 3D points. However, these datasets have been mainly obtained in almost flat cities, so the deep learning methods in the literature only consists of vehicle detection and regression of their horizontal orientation. The fact that cities like Lisboa (Portugal) [2] or San Francisco (California, USA) [3] 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.

B. Objectives

The main objectives of this work are 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; to develop an algorithm that allows manipulating existing datasets in order to modify vehicles by varying their vertical orientation; to make available a repository of code developed so it can be used for dataset creation and future work.

II. Background

A. LiDAR

LiDAR, as described in [1], is an active sensor that illuminates the surroundings by emitting laser beams and 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. that allows to precisely calculate the distance between each point and the subject. Figure 1, borrowed from [1], exemplifies a LiDAR system.

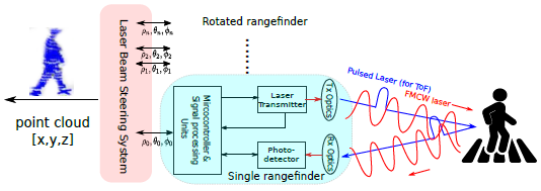

Figure 1. Example of a LiDAR system.

B. 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. 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 NLP (Natural Language Processing). DNNs (Deep Neural Networks) are the networks used in deep learning and the most popular types are MLP (Multi-Layer Perceptron), CNN (Convolutional Neural Network) and RNN (Recurrent Neural Network).

C. State-of-the-Art methods

1) Data augmentation: Deep learning models depends on large amounts of data so that training and subsequent results are consistent. Since this is not always possible, the need of data augmentation techniques arose to extend the number of data available in a dataset and, in some cases, to improve the quality of the data under study. Hahner et al. [4] published a work exploring the most commonly used data augmentation techniques in the task of detecting 3D objects through LiDAR where two categories stand out:

- Local augmentations - applied to the set of points belonging to an annotation;
- Global augmentations - applied simultaneously to all points of the point cloud.

The novel technique proposed in this work falls into the category of local augmentations.

2) LiDAR-based 3D Object Detection: Data representations in the LiDAR-based 3D object detection task depends on how the points in the point clouds are interpreted. Of these, two are highlighted: voxel grids and point sets. Methods based on voxel grids start by dividing the 3D points into a grid of voxels. PIXOR [5] and Complex-YOLO [6] are examples of methods that scattered voxels into pseudo-images that are later processed in object detection architectures. PointPillars [7] replaced 3D voxels by introducing the concept of 2D pillars in order to increase model efficiency. SECOND [8] and PVRCNN [9] 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 and the literature presents several approaches: object detection through a cropped point cloud obtained by 2D proposals in images (FPointNet [10]); proposition of objects directly from each point (PointRCNN [11]); proposal refinement through a sparse-to-dense (STD [12]) strategy.

3) Complex-YOLO: Complex-YOLO [6] is a state-of-the-art single-stage network for real-time 3D object detection that resorts to YOLO9000 [13], a 2D object detector for RGB images that creates a BEV representation of the point cloud and considers it as an image. Complex-YOLO [6] even 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 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, borrowed from [6], 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 [14]. Next, the E-RPN 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 and 3). This network modifies the common Grid-RPN approach by adding a component referring to the complex angle description of the object's orientation through the formula $arctan_2(t_{Im}, t_{Re})$, where t_{Im} and t_{Re} refer to complex parameters (see Figure 3).

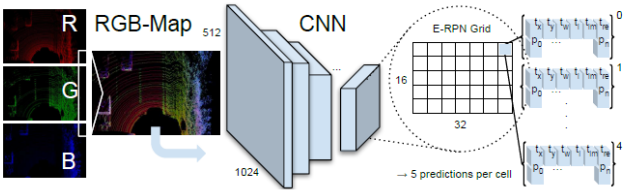

Figure 2. Complex-YOLO's pipeline. Image borrowed from [6]

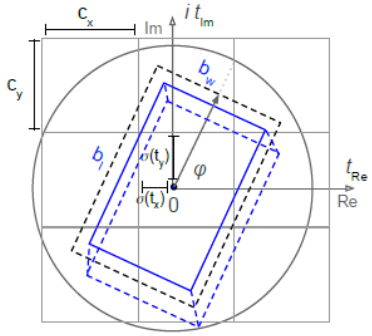

Figure 3. Complex-YOLO's bounding box regression. Image borrowed from [6].

III. Development

A. KITTI Dataset

Even though in recent years high-quality and large-scale datasets have been publicly released, the KITTI dataset [15], remains the benchmark dataset for object detection, being used by the authors for the implementation of Complex-YOLO [6].

Figure 4 illustrates the dimensions and mounting of the sensors in the vehicle. 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.

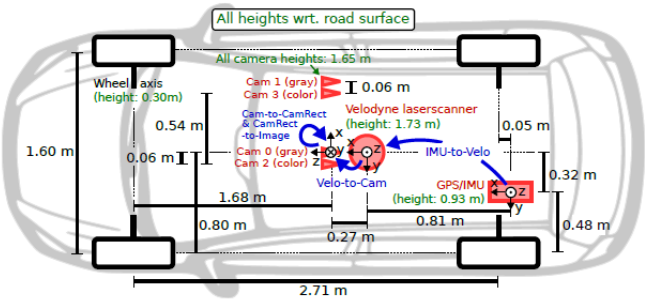

Figure 4. Representation of Vehicle's sensor setup, as seen in [15].

In relation to the annotation of ground truth objects, they are present in label files where the relevant characteristics for this work are:

- Type (1 column) - identifies the object's class ('Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' and 'DontCare');
- 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 4);
- Rotation_y (1 column) - rotation around Y-axis in Camera coordinate system (see Figure 4), ranging between $[-\pi, \pi]$.

B. Data Augmentation Algorithm

Since the main processes of this algorithm concern the transformation of objects into other coordinate systems, Figure 5 presents an illustrative scheme of the three used coordinate systems: Velodyne, Camera and Object.

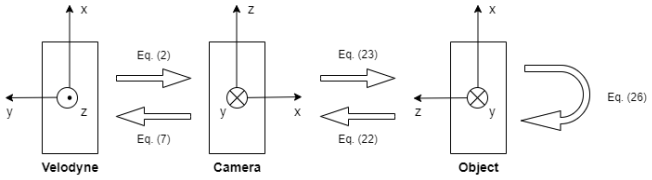

Figure 5. Representation of the different coordinate systems, as well as the equations representing transformations executed along this data augmentation algorithm.

1) Filter Point Cloud: Initially, the points that are far from the subject are excluded, thus obtaining a filtered point cloud that will be used in the subsequent tasks. Having in mind the LiDAR coordinate system (see Figure 5) the following boundaries were applied:

$$\begin{cases} x_m^{PCL} = 0[m] \\ x_M^{PCL} = 50[m] \\ y_m^{PCL} = -25[m] \\ y_M^{PCL} = 25[m] \\ z_m^{PCL} = -1.73[m] \\ z_M^{PCL} = 1.27[m] \end{cases} \quad (1)$$

Here, the value of z_m^{PCL} is given by the symmetric value of the height of LiDAR's sensor (see Figure 4 - height of Velodyne laserscanner, in green).

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 III-A), most of the algorithm's processes will be done in this coordinate system. Therefore, for this task it is required the translation matrix from Velodyne to Camera coordinate system that can be defined as follows:

$$^{c}\mathbf{T_v} = \begin{bmatrix} ^{c}\mathbf{R_v} & ^{c}\mathbf{t_v} \\ 0 & 1 \end{bmatrix} \quad (2)$$

Then, it is obtained a matrix only containing the points belonging to the point cloud:

$$^{v}\mathbf{P} = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \quad (3)$$

Here, n is the total number of points belonging to the point cloud. This matrix needs to be homogenised and transposed for the final step:

$$^{v}\tilde{\mathbf{P}} = \begin{bmatrix} ^{v}\mathbf{P} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \quad (4)$$

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

Finally, it is obtained a matrix of point cloud's points in Camera coordinate system:

$$^{c}\tilde{\mathbf{P}} = ^{c}\mathbf{T_v} \cdot ^{\mathbf{v}}\tilde{\mathbf{P}} = \begin{bmatrix} ^{c}\mathbf{P} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \quad (5)$$

Then, the file containing the labels of the objects is processed 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.

3) Get Object's centroid in LiDAR coordinate system: Object's centroid coordinates are given as location data in its label (see Section III-A). In order to obtain them in LiDAR coordinate system, firstly it is necessary to homogenise and transpose the centroid vector for matrix multiplication:

$$^{c}\tilde{\mathbf{c}} = \begin{bmatrix} ^{c}\mathbf{c_x} & ^{c}\mathbf{c_y} & ^{c}\mathbf{c_z} & 1 \end{bmatrix}^{\mathbf{T}} \quad (6)$$

A Camera to LiDAR system translation matrix is also required:

$$^{v}\mathbf{T_{cam}} = \left[^{c}\mathbf{T_{v}}\right]^{-1} \tag{7}$$

It is then possible to obtain the centroid of the Object in LiDAR coordinate system:

$$^{v}\tilde{\mathbf{c}} = ^{v}\ \mathbf{T_{c}} \cdot^{\mathbf{c}} \tilde{\mathbf{c}} = \begin{bmatrix} ^{v}\mathbf{c} & 1 \end{bmatrix}^{\mathbf{T}} \tag{8}$$

4) Define Object's bounding box limits in LiDAR coordinate system: Since it is not relevant to assess if an Object is contained in the point cloud in its entirety or partially, in this task the Object's orientation will not be considered. However, for this process it is essential to take into account the Object's height (h), width (w) and length (l), which are defined in its annotation data as the dimensions. Together with the Object's centroid and the LiDAR coordinate system arrangement (see Figure 4), it is then possible to spatially define its bounding box:

$$^{v}Bbox = \begin{bmatrix} ^{v}x_M & ^{v}y_M & ^{v}z_M \\ ^{v}x_M & ^{v}y_m & ^{v}z_M \\ ^{v}x_m & ^{v}y_M & ^{v}z_M \\ ^{v}x_m & ^{v}y_m & ^{v}z_M \\ ^{v}x_M & ^{v}y_M & ^{v}z_m \\ ^{v}x_M & ^{v}y_m & ^{v}z_m \\ ^{v}x_m & ^{v}y_M & ^{v}z_m \\ ^{v}x_m & ^{v}y_m & ^{v}z_m \end{bmatrix}^{\mathbf{T}} = \begin{bmatrix} ^{v}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} + \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} + \mathbf{h} \\ ^{v}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} - \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} + \mathbf{h} \\ ^{v}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} + \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} + \mathbf{h} \\ ^{v}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} - \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} + \mathbf{h} \\ ^{v}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} + \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} \\ ^{v}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} - \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} \\ ^{v}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} + \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} \\ ^{v}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_y} - \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_z} \end{bmatrix}^{\mathbf{T}} \tag{9}$$

Where the minimum and maximum values for each coordinate are defined as follows:

$$\begin{bmatrix} ^{v}x_m & ^{v}x_M \\ ^{v}y_m & ^{v}y_M \\ ^{v}z_m & ^{v}z_M \end{bmatrix} = \begin{bmatrix} ^{v}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{v}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} \\ ^{v}\mathbf{c_y} - \frac{\mathbf{w}}{\mathbf{2}} & ^{v}\mathbf{c_y} + \frac{\mathbf{w}}{\mathbf{2}} \\ ^{v}\mathbf{c_z} & ^{v}\mathbf{c_z} + \mathbf{h} \end{bmatrix} \tag{10}$$

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 11 are met:

$$\begin{cases} ^{v}x_m \geq x_m^{PCL} \\ ^{v}x_M \leq x_M^{PCL} \\ ^{v}y_m \geq y_m^{PCL} \\ ^{v}y_M \leq y_M^{PCL} \\ ^{v}z_m \geq z_m^{PCL} \\ ^{v}z_M \leq z_M^{PCL} \end{cases} \tag{11}$$

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 6 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.

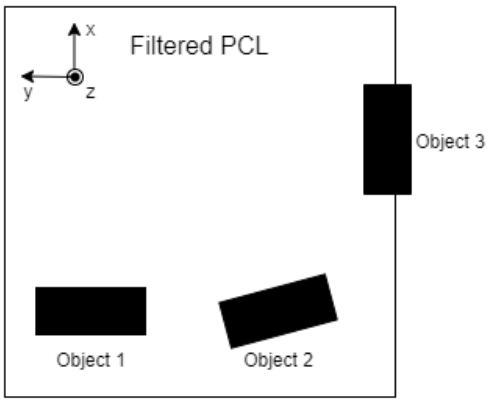

Figure 6. Example of objects in order to assess if they are within the filtered PCL's boundaries.

6) Define Object's bounding box limits in Camera coordinate system: Analogously to Section III-B4, the boundaries of the bounding box in the Camera coordinate system can be defined as follows:

$$^{c}Bbox = \begin{bmatrix} ^{c}x_M & ^{c}y_M & ^{c}z_M \\ ^{c}x_M & ^{c}y_m & ^{c}z_M \\ ^{c}x_m & ^{c}y_M & ^{c}z_M \\ ^{c}x_m & ^{c}y_m & ^{c}z_M \\ ^{c}x_M & ^{c}y_M & ^{c}z_m \\ ^{c}x_M & ^{c}y_m & ^{c}z_m \\ ^{c}x_m & ^{c}y_M & ^{c}z_m \\ ^{c}x_m & ^{c}y_m & ^{c}z_m \end{bmatrix}^{\mathbf{T}} = \begin{bmatrix} ^{c}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} - \mathbf{h} & ^{c}\mathbf{c_z} + \frac{\mathbf{w}}{\mathbf{2}} \\ ^{c}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} - \mathbf{h} & ^{c}\mathbf{c_z} + \frac{\mathbf{w}}{\mathbf{2}} \\ ^{c}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} - \mathbf{h} & ^{c}\mathbf{c_z} - \frac{\mathbf{w}}{\mathbf{2}} \\ ^{c}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} - \mathbf{h} & ^{c}\mathbf{c_z} - \frac{\mathbf{w}}{\mathbf{2}} \\ ^{c}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} & ^{c}\mathbf{c_z} + \frac{\mathbf{w}}{\mathbf{2}} \\ ^{c}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} & ^{c}\mathbf{c_z} + \frac{\mathbf{w}}{\mathbf{2}} \\ ^{c}\mathbf{c_x} - \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} & ^{c}\mathbf{c_z} - \frac{\mathbf{w}}{\mathbf{2}} \\ ^{c}\mathbf{c_x} + \frac{\mathbf{l}}{\mathbf{2}} & ^{c}\mathbf{c_y} & ^{c}\mathbf{c_z} - \frac{\mathbf{w}}{\mathbf{2}} \end{bmatrix}^{\mathbf{T}} \tag{12}$$

Where the extreme values for each coordinate are:

$$\begin{bmatrix} ^{c}x_m & ^{c}x_M \\ ^{c}y_m & ^{c}y_M \\ ^{c}z_m & ^{c}z_M \end{bmatrix} = \begin{bmatrix} ^{c}\mathrm{c}_x - \frac{l}{2} & ^{c}\mathrm{c}_x + \frac{l}{2} \\ ^{c}\mathrm{c}_y - h & ^{c}\mathrm{c}_y \\ ^{c}\mathrm{c}_z - \frac{w}{2} & ^{c}\mathrm{c}_z + \frac{w}{2} \end{bmatrix} \tag{13}$$

7) Apply Object's Yaw angle to defined bounding box in Camera coordinate system: In order to execute the next process, it is necessary to apply the orientation of the Object under analysis to its respective bounding box (see Figure 5). This orientation is given by the value of Rotation_y present in the Object's label and refers to the Yaw angle. Firstly, it is obtained the Object's bounding box coordinates in Object coordinate system:

$$^{o}BBox = ^{c}\ BBox - ^{c}\ \mathbf{c} \tag{14}$$

Then a homogeneous matrix with the bounding box points and the translation matrix given by the Yaw angle are obtained:

$$^{o}B\tilde{B}ox = \begin{bmatrix} ^{c}BBox & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{15}$$

$$\mathbf{T_y^c}(\psi) = \begin{bmatrix} \mathbf{R_y^c}(\psi) & 0 \\ 0 & 1 \end{bmatrix} \tag{16}$$

Here, $\mathbf{R_y^c}$ is defined by the rotation matrix along the Y-axis and the value of the Yaw angle by ψ:

$$\mathbf{R_y^c}(\psi) = \begin{bmatrix} cos(\psi) & 0 & sin(\psi) \\ 0 & 1 & 0 \\ -sin(\psi) & 0 & cos(\psi) \end{bmatrix} \tag{17}$$

Taking into account equations (15), (16) and (17), the coordinates of the points of the rotated bounding box are obtained by the following homogenised matrix:

$$^{o}B\tilde{B}ox_{\psi} = R_y^c(\psi) \cdot^{o} B\tilde{B}ox = \begin{bmatrix} ^{o}BBox_{\psi} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{18}$$

Where the matrix of coordinates of the points of the rotated bounding box can be defined as $^{o}BBox_{\psi}$. Finally, it is obtained the new Object's bounding box coordinates back in Camera coordinate system:

$$^{c}BBox_{\psi} =^{o} BBox_{\psi} +^{c} \mathbf{c} \tag{19}$$

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. 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 5) the Y-axis points downwards, a new value for the maximum limit in Y is thus obtained.

$$^{c}y_M =^{c} BBox_{\psi} \mid_{max\{y\}} -^{c}y_t \tag{20}$$

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 $^{c}y_t = 0.1m$.

Thereafter, every point of the filtered point cloud is evaluated. For a point to be considered as belonging to the Object, it must meet all the following conditions:

$$\begin{cases} p_x \geq^{c} BBox_{\psi} \mid_{min\{x\}} \\ p_x \leq^{c} BBox_{\psi} \mid_{max\{x\}} \\ p_y \geq^{c} BBox_{\psi} \mid_{min\{y\}} \\ p_y \leq^{c} y_{max} \\ p_z \geq^{c} BBox_{\psi} \mid_{min\{z\}} \\ p_z \leq^{c} BBox_{\psi} \mid_{max\{z\}} \end{cases} \tag{21}$$

If so, the point will be added to a matrix that will concern the set of all points of the Object. Before going further in the algorithm, a number of more than ten point cloud's points belonging to the Object under analysis is arbitrarily established as a valid selection criterion.

9) Generate Pitch angle: For this task, it is used the function randint from Python Standard Library's module Random [16] 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 Study 1 (see Section V). It is important to note that the rest of the algorithm is only executed if the value obtained for the Pitch angle is not zero.

10) Apply Pitch rotation to Object: In this step the Pitch angle is applied to the Object, obtaining a new set of points characteristic of a rotated object. From Figure 5, this so-called Pitch rotation, when applied in the Object's coordinate system, refers to a rotation along the Z-axis. Firstly, it is obtained the translation matrix from the Object to the Camera coordinate system, as well as its inverse:

$$^{c}\mathbf{T_o} = \begin{bmatrix} \mathbf{R_y^c}(\psi) & ^{c}\mathbf{c} \\ 0 & 1 \end{bmatrix} \tag{22}$$

$$^{o}\mathbf{T_c} = [^{\mathbf{c}}\mathbf{T_o}]^{-\mathbf{1}} \tag{23}$$

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:

$$^{o}\tilde{Obj} =^{o} \mathbf{T_c} \cdot^{\mathbf{c}} \tilde{\mathbf{Obj}} \tag{24}$$

$$^{o}B\tilde{B}ox =^{o} \mathbf{T_c} \cdot^{\mathbf{c}} \mathbf{B\tilde{B}ox} \tag{25}$$

Then, the translation matrix defined by the Pitch rotation is also obtained:

$$'\mathbf{T_o}(\theta) = \begin{bmatrix} \mathbf{R_z^c}(\theta)^{\mathbf{T}} & 0 \\ 0 & 1 \end{bmatrix} \tag{26}$$

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 (see Figure 5), the desired rotation matrix will be the matrix transpose of the one defined as follows:

$$\mathbf{R_z^c}(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{27}$$

Finally, it is possible to apply the Pitch rotation to the matrices obtained in equations (24) and (25), obtaining new sets of points for the Object and its bounding box, respectively:

$$^{o\prime}\tilde{Obj} =' \mathbf{T_{obj}}(\theta) \cdot^{\mathbf{o}} \tilde{\mathbf{Obj}} \tag{28}$$

$$^{o\prime}B\tilde{B}ox =' \mathbf{T_o}(\theta) \cdot^{\mathbf{o}} \mathbf{B\tilde{B}ox} \tag{29}$$

Here, ${}'\mathbf{T_o}(\theta)$ is the translation matrix defined by equation (26). Once the new sets of points are obtained in the Object coordinate system, they are as well obtained in the Camera coordinate system:

$$ {}^{c\prime}\tilde{Obj} = {}^{c}\,\mathbf{T_o} \cdot {}^{o\prime}\,\tilde{\mathbf{Obj}} \tag{30} $$

$$ {}^{c\prime}\tilde{BBox} = {}^{c}\,\mathbf{T_o} \cdot {}^{o\prime}\,\tilde{\mathbf{BBox}} \tag{31} $$

Here, ${}^{c}\mathbf{T_o}$ is the translation matrix of the coordinate system from the Object to the Camera, given by equation (22).

In order to correct the existence of points below the ground, it is calculated a value defined as the offset on the Y-axis to be applied to all points of both sets (see Figure 7). This offset, defined by the difference between the maximum value on the Y-axis in the set of Object points before and after applying the pitch rotation and, is calculated:

$$ {}^{c}y_t = {}^{c}\,BBox\,|_{max\{y\}} - {}^{c\prime}BBox\,|_{max\{y\}} \tag{32} $$

Then, this offset is added to all the points of the sets obtained by the equations (30) and (31):

$$ {}^{c\prime\prime}Obj = {}^{c\prime}\,Obj + \begin{bmatrix} 0 & {}^{c}y_t & 0 \end{bmatrix}^{\mathbf{T}} \tag{33} $$

$$ {}^{c\prime\prime}BBox = {}^{c\prime}\,BBox + \begin{bmatrix} 0 & {}^{c}y_t & 0 \end{bmatrix}^{\mathbf{T}} \tag{34} $$

Lastly, it is obtained the Object's centroid new coordinates, in order to update the Object's label. This can be done by adding to the centroid's Y coordinate the value of the threshold on the Y-axis given in (32):

$$ {}^{c\prime}\mathbf{c} = {}^{c}\,\mathbf{c} + \begin{bmatrix} 0 & {}^{c}y_t & 0 \end{bmatrix}^{\mathbf{T}} \tag{35} $$

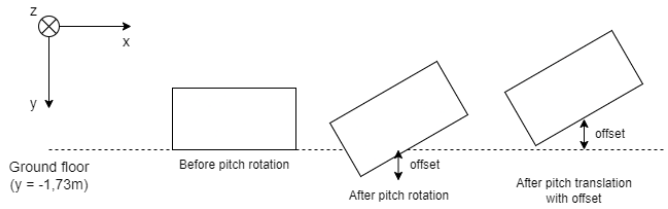

Figure 7. Example of a scenario where the pitch rotation applied to an object causes an offset to be applied through a pitch translation (without rotation).

11) Get new filtered Point Cloud's points in LiDAR coordinate system: The new points of the Object are stored in the filtered point cloud on the Camera coordinate system. Then, they are retrieved in the LiDAR coordinate system in order to be saved in the original point cloud. This can be done using equation (7), thus the homogeneous matrix of the final Point Cloud in LiDAR coordinate system being obtained as follows:

$$ {}^{v}\tilde{\mathbf{P}} = {}^{v}\,\mathbf{T_c} \cdot {}^{\mathbf{c}}\,\tilde{\mathbf{P}} = \begin{bmatrix} {}^{v}\mathbf{P} & \mathbf{1} \end{bmatrix}^{\mathbf{T}} \tag{36} $$

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 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.

IV. Implementation

A. 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 I. A GitHub repository [17] that implements Complex-YOLO [6] and YOLOv3 [18] as the 2D object detector was used.

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 I
Specifications of ISR's machine.

B. Models: Training and Testing

The original KITTI dataset [15] 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 - containing 5236 samples, roughly 70% of the total number of samples;
- Validation - used for training purposes and including 749 samples, approximately 10% of the total;
- Evaluation - used for testing purposes and consisting of 1496 samples, about 20% the total.

C. Evaluation Metrics

1) Intersection over Union: This is an essential metric for the object detection task that allows calculating the overlap between predicted and ground truth bounding boxes. Mathematically, it can be defined as:

$$ IoU = \frac{BBox_{predicted} \cap BBox_{groundTruth}}{BBox_{predicted} \cup BBox_{groundTruth}} \tag{37} $$

Here, $BBox_{predicted}$ and $BBox_{groundTruth}$ represent the predicted bounding box and the ground truth bounding box, respectively. Both terms of equation (37) define an area or a volume, in the case of object detection in 2D (e.g., Bird's-Eye View) 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 [19] is required for the detection to be considered as a True Positive (TP).

2) Average Precision: The Average Precision (AP) metric, summarises the shape of the Precision/Recall curve. In order to calculate it, it is first necessary to calculate the precision and recall for a set of 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:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, ..., 0.9, 1.0\}} p_{interp}(r) \tag{38}$$

$$p_{interp}(r) = \max_{\widetilde{r}:\widetilde{r} \geq r} p(\widetilde{r}) \tag{39}$$

3) Average Orientation Similarity: The Average Orientation Similarity (AOS) metric is based on the AP metric defined in equation (38) 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:

$$AOS = \frac{1}{11} \sum_{r \in \{0, 0.1, ..., 0.9, 1.0\}} \max_{\widetilde{r}:\widetilde{r} \geq r} s(\widetilde{r}) \tag{40}$$

Here, the orientation similarity $s \in [0, 1]$ at recall r is a ([0...1]) normalised variant of the cosine similarity [20] defined by the following equation:

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + \cos \Delta_{\theta}^{(i)}}{2} \delta_i \tag{41}$$

Where $D(r)$ 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 [20].

4) Frame Rate: In the context of autonomous driving it is relevant to infer the obtained models about its frame rate, i.e. Frames Per Second (FPS), that can be defined by the following equation:

$$FPS = \frac{1}{inferenceTime} \tag{42}$$

Here, inference time is defined as the time required for the model to complete a forward propagation, given a specific input. Therefore, through this FPS metric it is possible to determine, for a given speed, the average distance travelled between each input processing.

V. Results

A. Studies and Datasets

In order to evaluate the developed algorithm, three studies are proposed:

- Study 1 - tries to recreate cities located on hills, such as Lisbon (Portugal) [2] or San Francisco (USA) [3], suggesting streets with different levels of slope, defined through the pitch angle. Likewise, tries to recreate cities located in plain terrains, such as Amsterdam (Netherlands) [21] or Leeds (UK) [22], proposing streets with similar and low levels of slope. Then, compares these two scenarios with the baseline 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 - extends the previous study with an analysis on computational performance between the two typologies (Normal and Tiny) available for the YOLOv3 neural network;
- Study 3 - only for the Normal typology, it details the performance metrics obtained for the three most used detection classes in the literature.

For the task of detecting objects within the scope of autonomous driving systems, the literature focuses mainly on three classes: 'Car/Van', 'Cyclist' and 'Pedestrian'. However, this work will not manipulate objects belonging to detection classes 'Pedestrian'.

Adding to the Baseline dataset (original KITTI dataset [15]), two datasets will be created:

- 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.

In Figure 8 can be seen the histograms of the pitch angles generated 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.

Figures 9 to 12 present some examples manipulated ground truths. In them, the green bounding box refers to the ground truth (including the yaw rotation) and the red bounding box refers to the augmentation (adding the pitch rotation).

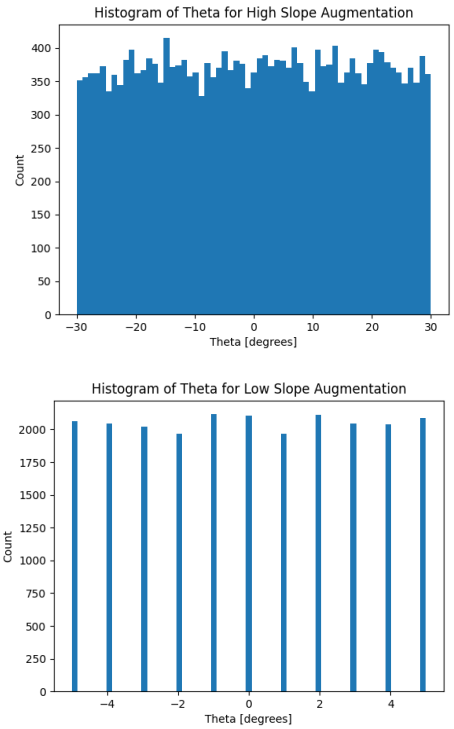

Figure 8. Histograms of pitch angles generated during the creation of each dataset: High Slope (on the left) and Low Slope (on the right).

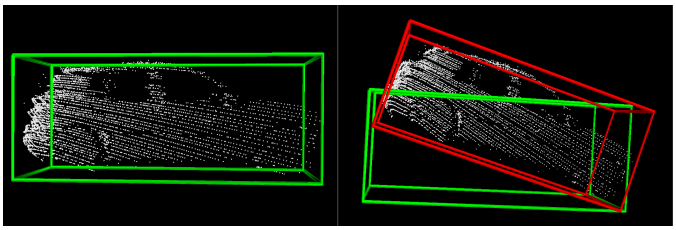

Figure 9. Example of augmented object. Dataset: High Slope. Class: Car/Van. Pitch angle = -16 degrees.

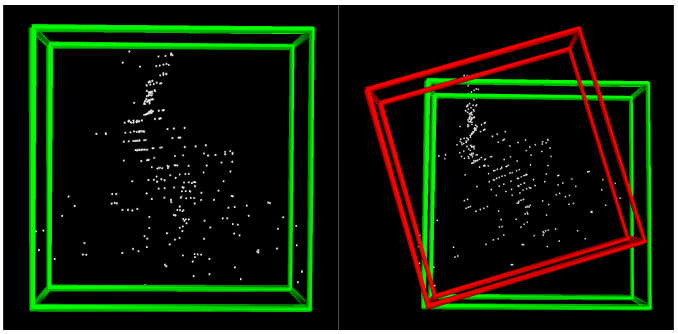

Figure 10. Example of augmented object. Dataset: High Slope. Class: Cyclist. Pitch angle = 17 degrees.

B. Study 1: Vehicle Detection Accuracy Analysis

Table II presents all the average values of each performance metric obtained for each possible scenario. When analysing it, it is possible to conclude that:

- the Baseline model presents the best metrics in the three testing sets;
- The values of the performance metrics decrease as the variation of the pitch angles in the manipulated objects increases;
- The performance loss in the mAP metric is not considerable, which can not be said regardind the mAOS metric.

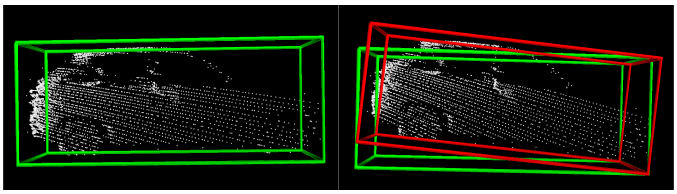

Figure 11. Example of augmented object. Dataset: Low Slope. Class: Car/Van. Pitch angle = -5 degrees.

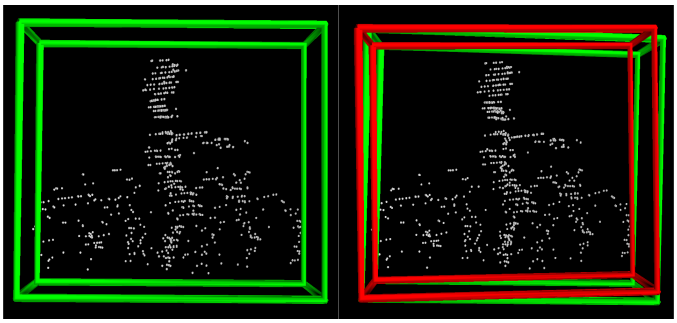

Figure 12. Example of augmented object. Dataset: Low Slope. Class: Cyclist. Pitch angle = 2 degrees.

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 II
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.

In conclusion, a generalised loss of performance by the models was expected, since a substantial modification was being introduced to the objects. 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.

C. Study 2: Computational Performance Analysis

Similar to what is done in Section V-B, Table III adds the detection performance results obtained for the other typology available in YOLOv3 [18], Tiny. By analysing it, one direct conclusion is that 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.

Likewise the Normal typology, also the Tiny presents an increasing loss of performance in the detection metrics in relation to the variety of angles applied to the manipulated objects, being quite significant. However, in terms of computational performance, the Tiny typology is far superior to Normal: considering the last column of Table III, 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.

Although the performance values for the detection of objects of the Tiny typology deviate from the benchmark in the literature, 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.

D. Study 3: Class-Specific Detection Analysis

In this last Study, Table IV shows detection performance for each of the three most used classes in the literature.

When analysing Table IV, it is noticeable that:

- In general, the loss of performance for both metrics increases with the variation of pitch orientation among the three detection classes;
- The 'Car/Van' detection class presents quite promising performance values;
- The 'Pedestrian' and 'Cyclist' detection classes shows lower values for detection performance, especially the 'Pedestrian';
- These values 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 V 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. This is also a factor to consider as a possible explanation for a lower detection performance in the 'Cyclist' class.

VI. Conclusions

A. Contributions

Since the main datasets in the existing literature do not include information regarding the pitch angle and that there is no data augmentation method for this angle, this work presents a relevant contribution for the scientific community in the field of autonomous driving systems. Also, by having presented a detailed description of the algorithm developed and it being available in a code repository[1], it is possible to confidently state that the objectives proposed by this work were achieved. Finally, through the studies carried out, it can be affirmed that the addition of a pitch orientation on the objects does not have an impact on the correct detection negative to the point of being considered a setback or impediment to continue investigating further.

B. Future Work

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

- Adapting 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;
- Implementing or expanding object detection networks to include pitch angle regression. 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 [23]) or a spherical coordinate system.

References

[1] Y. Li and J. Ibanez-Guzman, "Lidar for Autonomous Driving: The principles, challenges, and trends for automotive lidar and perception systems," 4 2020.

[2] rosamfelix, "rosamfelix/gis/declives/DeclivesLisboa."

[3] u/JPPlus, "Map of San Francisco Bike Paths Showing Slope."

[4] M. Hahner, D. Dai, A. Liniger, and L. Van Gool, "Quantifying Data Augmentation for LiDAR based 3D Object Detection," 4 2020.

[5] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D Object Detection from Point Clouds," tech. rep., Uber Advanced Technologies Group, 2018.

[6] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-YOLO: Real-time 3D Object Detection on Point Clouds," 3 2018.

[7] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," tech. rep., nuTonomy, 2019.

[8] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," Sensors (Switzerland), vol. 18, 10 2018.

[9] 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," tech. rep., The Chinese University of Hong Kong, 2020.

[10] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," tech. rep., Stanford University, 2018.

[11] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud," tech. rep., The Chinese University of Hong Kong, 2019.

[12] Z. Yang, Y. Sun, S. Liu, X. Shen, J. Jia, and Y. Lab, "STD: Sparse-to-Dense 3D Object Detector for Point Cloud," tech. rep., YouTu Lab, Tencent, 2019.

[13] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," tech. rep., University of Washington, 7 2017.

[1] https://github.com/joaomlf/kittidataset_pitchangleaugmentation

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 III

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

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 IV

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

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 V

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

[14] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-View 3D Object Detection Network for Autonomous Driving," tech. rep., Tsinghua University, 2017.

[15] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," tech. rep., Karlsruhe Institute of Technology, 2012.

[16] Python Software Foundation, "The Python Standard Library, module random — Generate pseudo-random numbers."

[17] ghimiredhikura, "ghimiredhikura/Complex-YOLOv3: PyTorch implementation of Complex-YOLO paper with YoloV3."

[18] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 4 2018.

[19] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago, "The KITTI Vision Benchmark Suite."

[20] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," tech. rep., Karlsruhe Institute of Technology, 2012.

[21] rosamfelix, "rosamfelix/gis/declives/SlopesAmsterdam."

[22] rosamfelix, "rosamfelix/gis/declives/SlopesLeeds."

[23] Caesar HBankiti VLang AVora SLiong VXu QKrishnan APan YBaldan GBeijbom O, "nuScenes - Data format."