# Complex-YOLO: Real-time 3D Object Detection on Point Clouds

## Introduction

Point cloud processing is becoming more and more important to autonomous driving due to the strong improvement of automotive LIDAR sensors in the recent years. Compare to images, Lidar points are sparse with varying density distributed all over the measurement area. Those points are unordered, they interact locally.

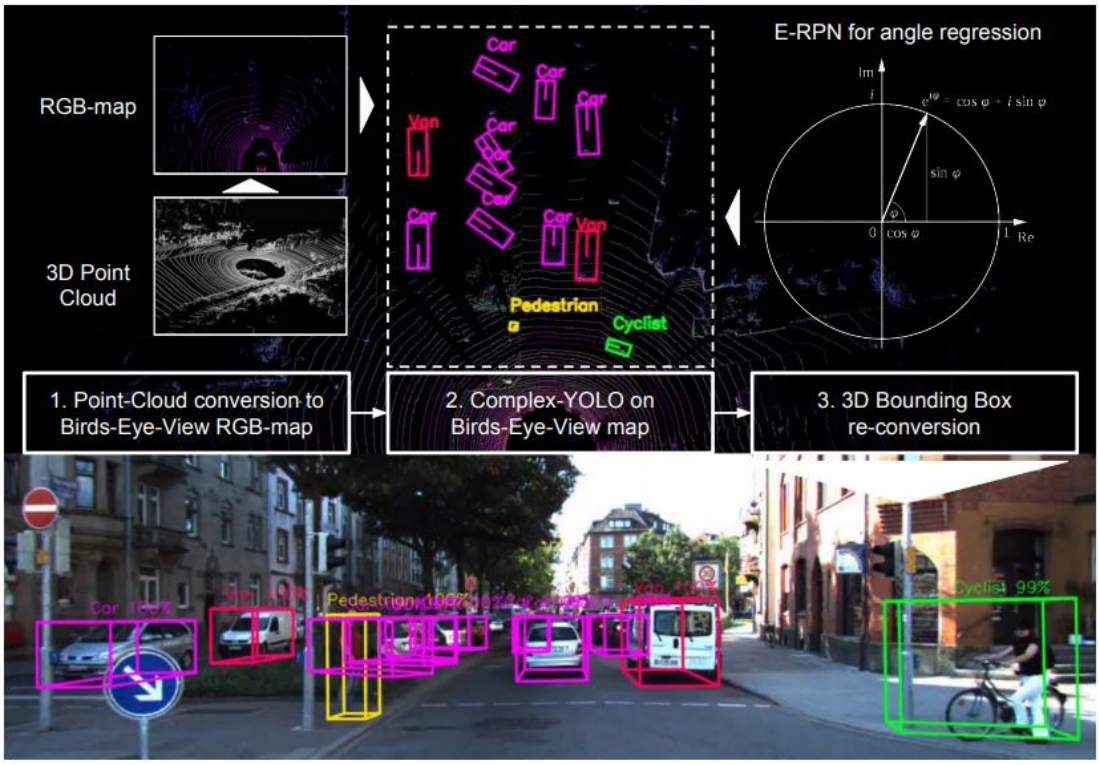



**Fig. 1.** Complex-YOLO is a very efficient model that directly operates on Lidar only based birds-eye-view RGB-maps to estimate and localize accurate 3D multiclass bounding boxes. The upper part of the figure shows a bird view based on a Velodyne HDL64 point cloud (Geiger et al. [1]) such as the predicted objects. The lower one outlines the re-projection of the 3D boxes into image space. Note: Complex-YOLO needs no camera image as input, it is Lidar based only.

## Contribution

The complex-yolo use the multi-view idea (MV3D)[1] for point cloud pre-processing and feature extraction. Complex-yolo only uses BEV from point cloud, therefore it is based on Lidar only. The main contributions are as follows:

a. Yolo-v2 is extended by introducing new E-RPN for angle regression for 3D box estimation.

b. Real time performance.
c. The heading angle of the 3D box supported by R-RPN estimated which enables to predict the trajectory of the object.

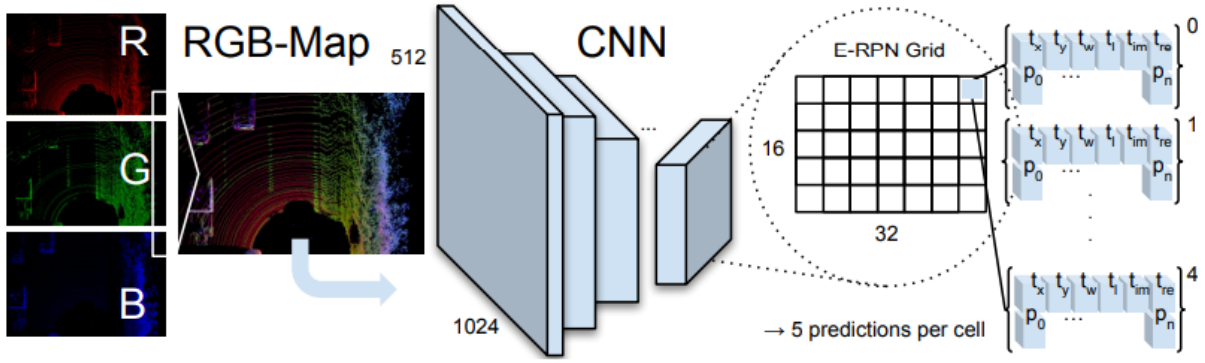



**Fig. 2. Complex-YOLO Pipeline.** We present a slim pipeline for fast and accurate 3D box estimations on point clouds. The RGB-map is fed into the CNN (see Tab. 1). The E-RPN grid runs simultaneously on the last feature map and predicts five boxes per grid cell. Each box prediction is composed by the regression parameters $t$ (see Fig. 3) and object scores $p$ with a general probability $p_0$ and $n$ class scores $p_1 ... p_n$.

**Point Cloud Preprocessing**

The velodyne data used to get BEV from the following range.

$$\mathcal{P}_\Omega = \{\mathcal{P} = [x, y, z]^T | x \in [0, 40m], y \in [-40m, 40m], z \in [-2m, 1.25m]\} \quad (1)$$

It covers the front point cloud area of **80m x 40m** which is directly in front of the origin of the sensor. The BEV is encoded by three channels (height, intensity and density) similar to RGB channels in 2D image. The size of the grid map is defined with n = 2024 and m = 512, that turns out to be g = 8cm grid resolution in BEV grid map.

A sample source code for obtaining above BEV can be found here.

https://github.com/AI-liu/Complex-YOLO/blob/master/utils.py#L31

**Euler-Region-Proposal**

The E-RPN in complex yolo parses the 3D position ($b_x$, $b_y$), object dimensions (width $b_w$, length $b_l$) as well as probability $p_0$, class scores $p_1,\ldots p_n$ and finally orientation $b_{angle}$ from the incoming feature map. In overall there is just small extension to yolo which is described by following eq.

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_l &= p_l e^{t_l} \\
b_\phi &= arg(|z|e^{ib_\phi}) = \arctan_2(t_{Im}, t_{Re})
\end{aligned}
$$

**Anchor Box Design**

Complex-yolo defined only three differentsizes and two angle directions as priors, based on the distribution of boxes within the KITTI dataset: i) vehicle size (heading up); ii) vehicle size (heading down); iii) cyclist size (heading up); iv) cyclist size (heading down); v) pedestrian size (heading left).

**Complex Angle Regression**

The orientation angle for each object is obtained from responsible regression parameters $t_{im}$ and $t_{re}$, the angle is given simply by using following equation: $\arctan_2(t_{im}, t_{re})$

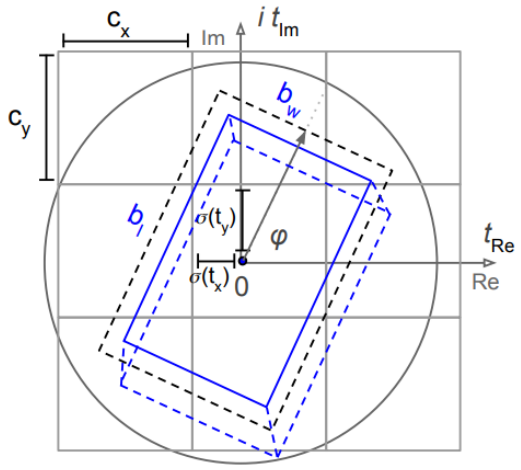



**Fig. 3. 3D Bounding box regression.** We predict oriented 3D bounding boxes based on the regression parameters shown in YOLOv2 [13], as well as a complex angle for box orientation. The transition from 2D to 3D is done by a predefined height based on each class.

**Loss Function**

The complex-yolo loss function is simple extension to yolo loss function.

$$\mathcal{L} = \mathcal{L}_{\text{Yolo}} + \mathcal{L}_{\text{Euler}}$$

Yolo loss:

$$\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left( p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}$$

Euler loss:

$$\begin{aligned}
\mathcal{L}_{\text{Euler}} &= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left| (e^{ib_\phi} - e^{i\hat{b}_\phi})^2 \right| \\
&= \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[ (t_{im} - \hat{t}_{im})^2 + (t_{re} - \hat{t}_{re})^2 \right]
\end{aligned}$$

Pytorch-implementation: https://github.com/AI-liu/Complex-YOLO/blob/master/region_loss.py

**Training Details**

Stochastic gradient descent with a weight decay of 0.0005 and momentum 0.9

The problem with KITTI dataset is data distribution among classes. The class distribution with more than 75% Car, less than 4% Cyclist and less than 15% Pedestrian is disadvantageous. They mentioned this issue in the paper but didn't provide any solution for this data distribution problem.

For the first epochs, we started with a small learning rate to ensure convergence. After some epochs, we scaled the learning rate up and continued to gradually decrease it for up to 1,000 epochs.

**Results**

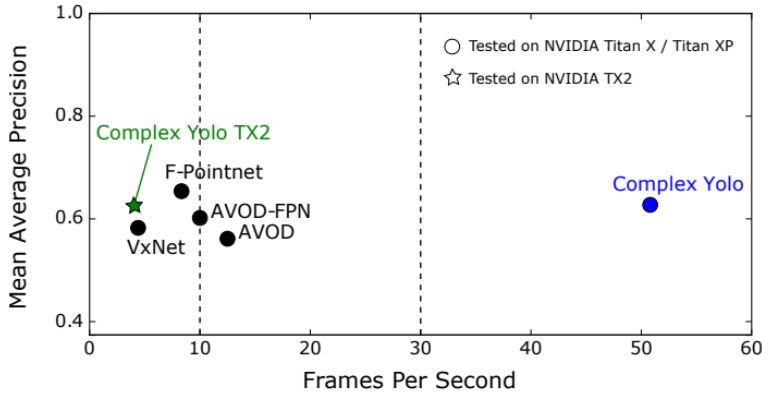


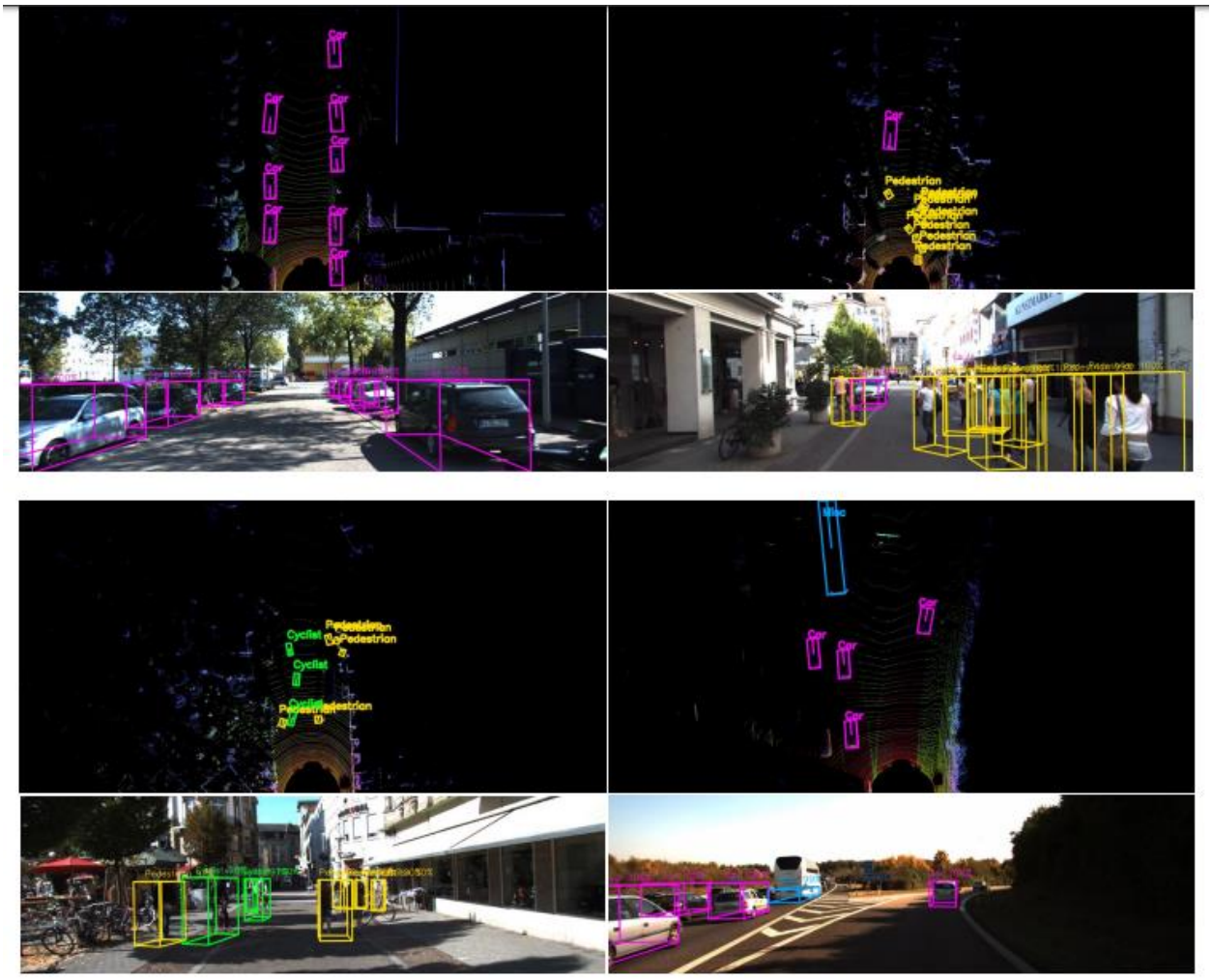

**Fig. 5. Performance comparison.** This plot shows the mAP in relation to the run-time (fps). All models were tested on a Nvidia Titan X or Titan Xp. Complex-Yolo achieves accurate results by being five times faster than the most effective competitor on the KITTI benchmark [1]. We compared to the five leading models and measured our network on a dedicated embedded platform (TX2) with reasonable efficiency (4fps) as well. Complex-Yolo is the first model for real-time 3D object detection.

**Table 2. Performance comparison for birds-eye-view detection.** APs (in %) for our experimental setup compared to current leading methods. Note that our method is validated on our splitted validation dataset, whereas all others are validated on the official KITTI test set.

| Method | Modality | FPS | Car | | | Pedestrian | | | Cyclist | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Easy | Mod. | Hard | Easy | Mod. | Hard | Easy | Mod. | Hard |
| MV3D [2] | Lidar+Mono | 2.8 | 86.02 | 76.90 | 68.49 | - | - | - | - | - | - |
| F-PointNet [5] | Lidar+Mono | 5.9 | 88.70 | 84.00 | 75.33 | **58.09** | **50.22** | **47.20** | **75.38** | 61.96 | 54.68 |
| AVOD [7] | Lidar+Mono | 12.5 | 86.80 | **85.44** | 77.73 | 42.51 | 35.24 | 33.97 | 63.66 | 47.74 | 46.55 |
| AVOD-FPN [7] | Lidar+Mono | 10.0 | 88.53 | 83.79 | **77.90** | 50.66 | 44.75 | 40.83 | 62.39 | 52.02 | 47.87 |
| VoxelNet [3] | Lidar | 4.3 | **89.35** | 79.26 | 77.39 | 46.13 | 40.74 | 38.11 | 66.70 | 54.76 | 50.55 |
| Complex-YOLO | Lidar | **50.4** | 85.89 | 77.40 | 77.33 | 46.08 | 45.90 | 44.20 | 72.37 | **63.36** | **60.27** |

**Table 3. Performance comparison for 3D object detection.** APs (in %) for our experimental setup compared to current leading methods. Note that our method is validated on our splitted validation dataset, whereas all others are validated on the official KITTI test set.

| Method | Modality | FPS | Car | | | Pedestrian | | | Cyclist | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Easy | Mod. | Hard | Easy | Mod. | Hard | Easy | Mod. | Hard |
| MV3D [2] | Lidar+Mono | 2.8 | 71.09 | 62.35 | 55.12 | - | - | - | - | - | - |
| F-PointNet [5] | Lidar+Mono | 5.9 | 81.20 | 70.39 | 62.19 | **51.21** | **44.89** | **40.23** | **71.96** | 56.77 | 50.39 |
| AVOD [7] | Lidar+Mono | 12.5 | 73.59 | 65.78 | 58.38 | 38.28 | 31.51 | 26.98 | 60.11 | 44.90 | 38.80 |
| AVOD-FPN [7] | Lidar+Mono | 10.0 | **81.94** | **71.88** | **66.38** | 46.35 | 39.00 | 36.58 | 59.97 | 46.12 | 42.36 |
| VoxelNet [3] | Lidar | 4.3 | 77.47 | 65.11 | 57.73 | 39.48 | 33.69 | 31.51 | 61.22 | 48.36 | 44.37 |
| Complex-YOLO | Lidar | **50.4** | 67.72 | 64.00 | 63.01 | 41.79 | 39.70 | 35.92 | 68.17 | **58.32** | **54.30** |



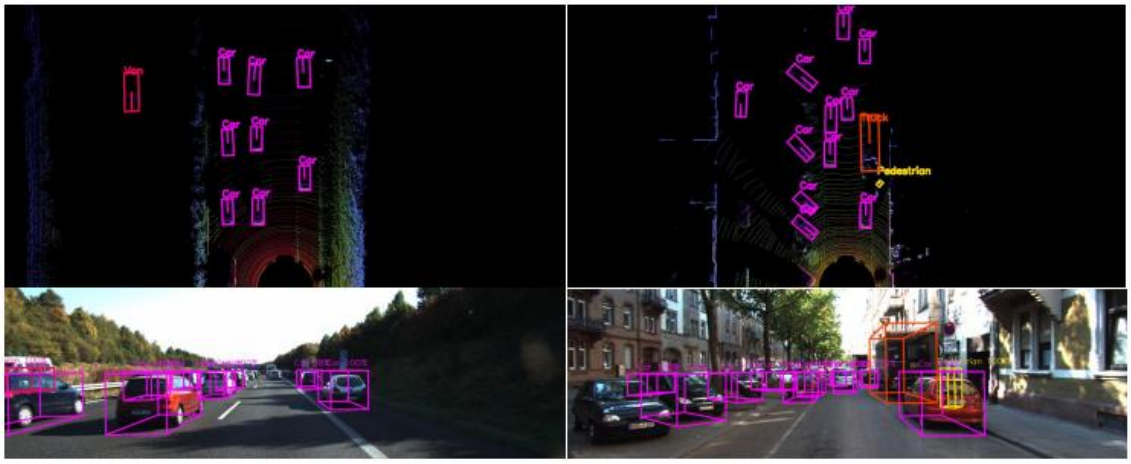



**Fig. 6. Visualization of Complex-YOLO results.** Note that predictions are exclu sively based on birds-eye-view images generated from point clouds. The re-projectio into camera space is for illustrative purposes only.

## References

1. Chen, Xiaozhi, et al. "Multi-view 3d object detection network for autonomous driving." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017. https://arxiv.org/abs/1611.07759