Cone Detection for Autonomous Formula Using Deep Learning Methods

Author: Chenhao Yang

Advisors: Prof. Anath Fischer, Ronit Schneor

ISRAEL INSTITUTE OF TECHNOLOGY

Laboratory for CAD & LCE

Haifa, Israel

2021 Spring

Abstract

The development of detecting and position estimating of objects has became more and more prevailing in the field of robotics and autonomous driving. In this report, we developed a cone detection algorithm for the autonomous formula student using deep learning approaches. We created 3D point cloud data to simulate LiDAR scannings for supervised training. Our detection approach is to voxelize the point cloud and create pseudo images first, and then using 2D convolution layers and variational autoencoder model for feature extraction and generation of cone coordinate candidates. Finally, we used non-maximum suppression to get outputs and fine results.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Motivation

The formula student competition was initiated by the SAE(Society of Automotive Engineers), with tasks of DESIGN, BUILD, AND RACE a formula style vehicle under strict regulations. Participaters consist of approximately 600 universities from all around the world.

The competition is divided into combust engine group and electric group. In 2017, a new competition was raised. The students are required to upgrade their previous formula cars to driverless, and the car will be able to finish the race track fully autonomously.

Technion students [1] built up a simulation system and trained a deep neural network that uses a single camera as input for inferring car steering angles in real-time, the team achieved great success.

Within the scope of this report, we would like to use Point Cloud generated from LiDAR and methods of deep learning to accomplish the tasks of cone detection.

Figure 1.1: Formula Technion 2016 from [2]

Chapter 2

LITERATURE REVIEW

In this section, we are going to review basic autonomous vehicle systems and state-of-the-art object detection algorithms for applications on sutonomous driving.

2.1 Autonomous vehicle system overview

Autonomous vehicles(AV or auto) are vehicles where human drivers are not required to drive the vehicle, instead, combining sensors and software are to control, navigate, and drive the vehicle.

AVs are equipped with various sensors that are responsible for detecting environmental characteristics and their data are passed to the onboard computer. Currently, the most commonly used sensors are cameras, GPS, inertial measurement unit (IMU), and LiDARs.

Then, the onboard computer processes environmental information and uses algorithms to make decision whether to steer, accelerate, brake, etc. There are three systems can categorize these algorithms: perception, trajactory planning, and control. More details can be inspected from 2.1

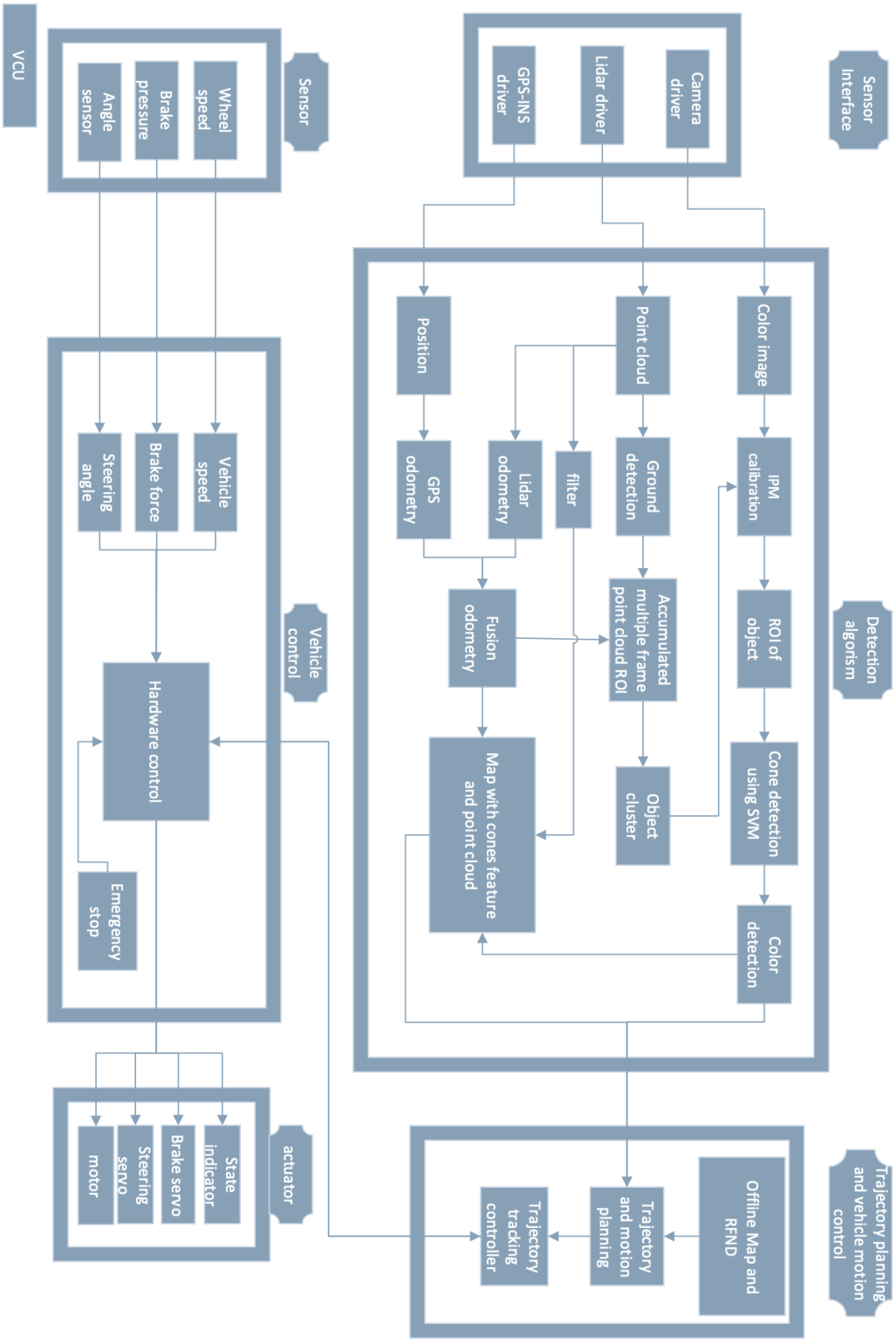


Figure 2.1: System of autonomous driving vehicle[3]

2.2 The Object Detection Challenge in Autonomous Driving

Object detection is one of the goals in perception algorithm, it requires the system to find out necessary information entering sensors. For example, understanding from a given frame whether an surrounding object is another vehicle, a person, or something else.

So far, there are numerious researchers around the world working in this field, most of their approaches are based on Deep Learning. We first introduce their commonly used dataset as for training or validation purposes, and then briefly introduce their work classified by their training data type.

Dataset

Most object detection algorithms in AV used supervised models learned from labeled data. Here we introduce two public datasets, figure **??** also provides a quantitative comparison between more datasets.

- **KITTI**

  KITTI dataset ([4]) was recorded from a vehicle (figure 2.2) while driving in and around Karlsruhe, Germany. It is the most well-known dataset in AV field since it was the earlist published. The main goal of this dataset is to push forward the development of computer vision and robotic algorithms targeted to autonomous driving. It includes camera images, laser scans, high-precision GPS measurements and IMU accelerations from a combined GPS/IMU system.

  There exist 7481 training scenes and 7581 test scenes in the KITTI dataset. Each training example is a labeled 3d scene captured via two camera images generated by the two forward facing cameras and the point cloud generated by the Velodyne HDL-64E lidar sensor mounted on the roof of the car.

- **ApolloScape**

  ApolloScape dataset [5] was published by Baidu Inc. It includes trajectory dataset, 3D perception Lidar object detection and tracking dataset. A mobile LiDAR scanning sensor Reigl is used to collected point cloud data, which allows more precise and denser points than Velodyne LiDAR. The detection dataset includes about 100K image frames, 80k LiDAR point cloud frames. ApolloScape adopts an efficient semiautomatic labelling strategy for 3D point cloud labelling. A PointNet++[6] network is used to pre-label the over-segmented point cloud clusters, then the annotations are manually refined.

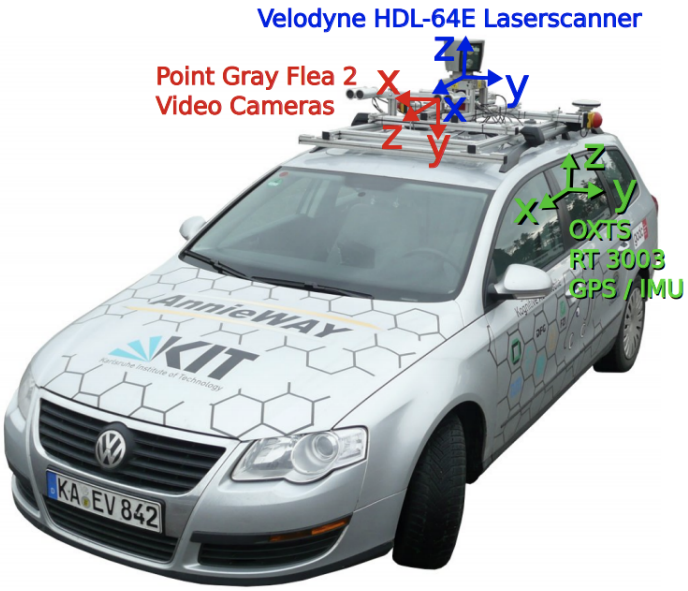


Figure 2.2: The recording platform of KITTI Dataset. [4]

3-D Detection from Mono Camera

The task of 3D object detection from a 2D image is fundamentally ill-posed as the critical information of the depth dimension is missing. However, this task is still tractable and 3D information of vehicles can be extracted under certain conditions and our prior knowledge of shapes of cars, traffic systems, etc.
There are mainly 4 methods available([7]), as listed:

- **Representation Transformation(pseudo-Lidar, BEV)**
  This method is to convert perspective images to Birds-eye-view (BEV) which makes use of an inverse perspective mapping to effectively estimate the distance from the image. ([8],[9],[10],[11])

- **Keypoints and Shapes**
  This method takes advantages of well-known common parts of vehicles that can be used as keypoints for detection, classification and others. Additionally, the distance between objects to ego-vehicle can be estimated from their size. Most of the studies([12],[13],[14],[15],[16]) extended the 2D object detection framework (one-stage such as Yolo or RetinaNet, or two-stage such as Faster RCNN) to predict keypoints.

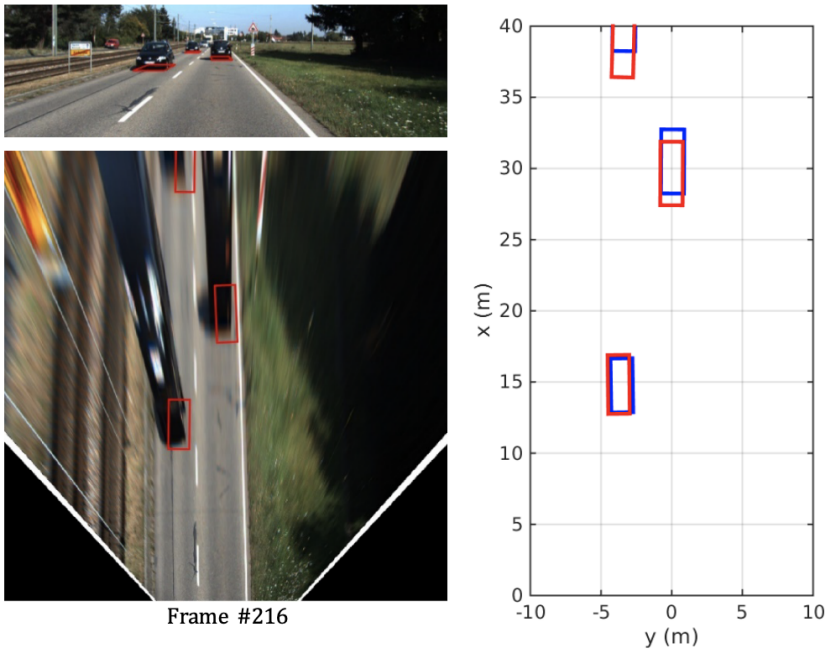


Figure 2.3: Perspective image from mono cam to BEV. [8]

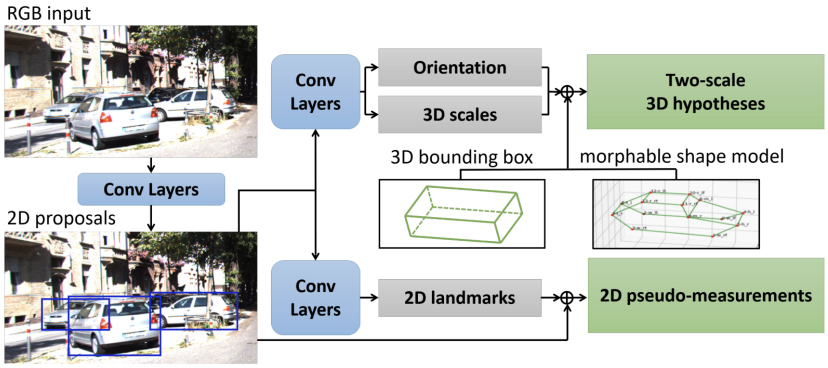


Figure 2.4: Keypoints extraction pipeline purposed in **Mono3D++**. [16]

- **Distance estimation through 2D/3D constraints**

  This method extends 2D object detection framework by adding a branch regressing the local yaw and the dimension offset from the subtype average. Using these geometric hints, it solves an over-constrained optimization problem to obtain the 3D location, lifting 2D bounding boxes to 3D. ([17],[18],[19],[20],[21])

- **Direct Generation of 3D proposal**

  This method focuses on direct 3D proposal generation and generates dense

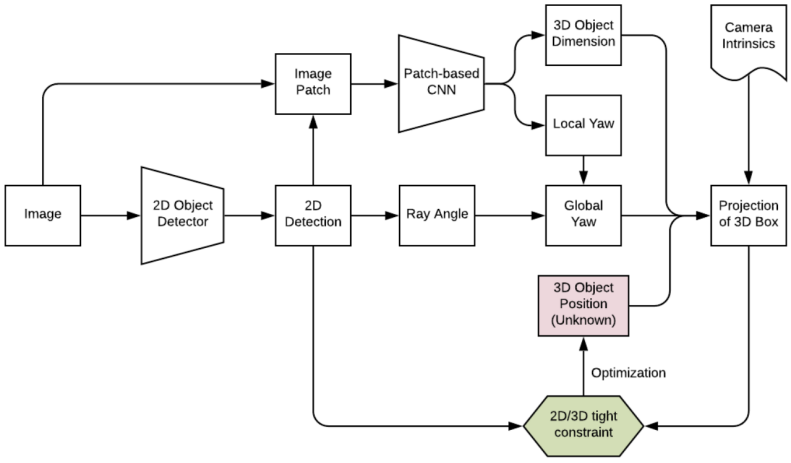


Figure 2.5: The architecture of **Deep3DBox** [17], representative of many other similar works.

proposals. Then it scores each proposal by several hand-crafted features such as class semantic, object shape, context, and so on. Then, it performs non-maximum suppression(NMS) and scoring to get the final detection results ([22]). Similar methods are also proposed ([23], [24], [25],[26],[27]).

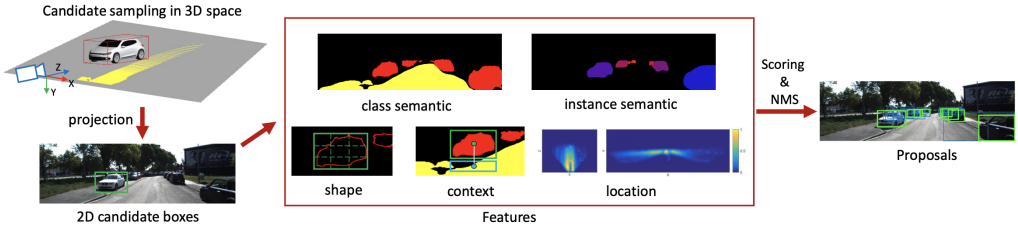


Figure 2.6: Overview of direct generaction approach. [22]

3-D Detection with LiDAR

Unlike two-dimensional detection based on camera images, sensors with spatial sensing capabilities are beneficial for identifying real-world objects. LiDAR (Light Detection and Ranging) uses laser beams for three-dimensional imaging. The 3D point cloud obtained by transmitting and receiving laser pulses within the scanning range of LiDAR describes the 3D world. Due to the directionality and energy concentration of the laser, the 3D point cloud data is precise and has a high spatial resolution. In addition, as an active imaging system, LiDAR is not limited by illumination conditions compared to passive optical instruments, making it naturally suitable for 3D object detection in complex environments.[28] For example, LiDAR

sensors still have sensing abilities in foggy whether while cameras are not.

There are mainly three key catalogs of technologies in 3D detection networks based on their challenges and technical evolutions.[29]

Most of the approaches are recently proposed and developed, here we present a chronological illustration of the approaches in 2.7

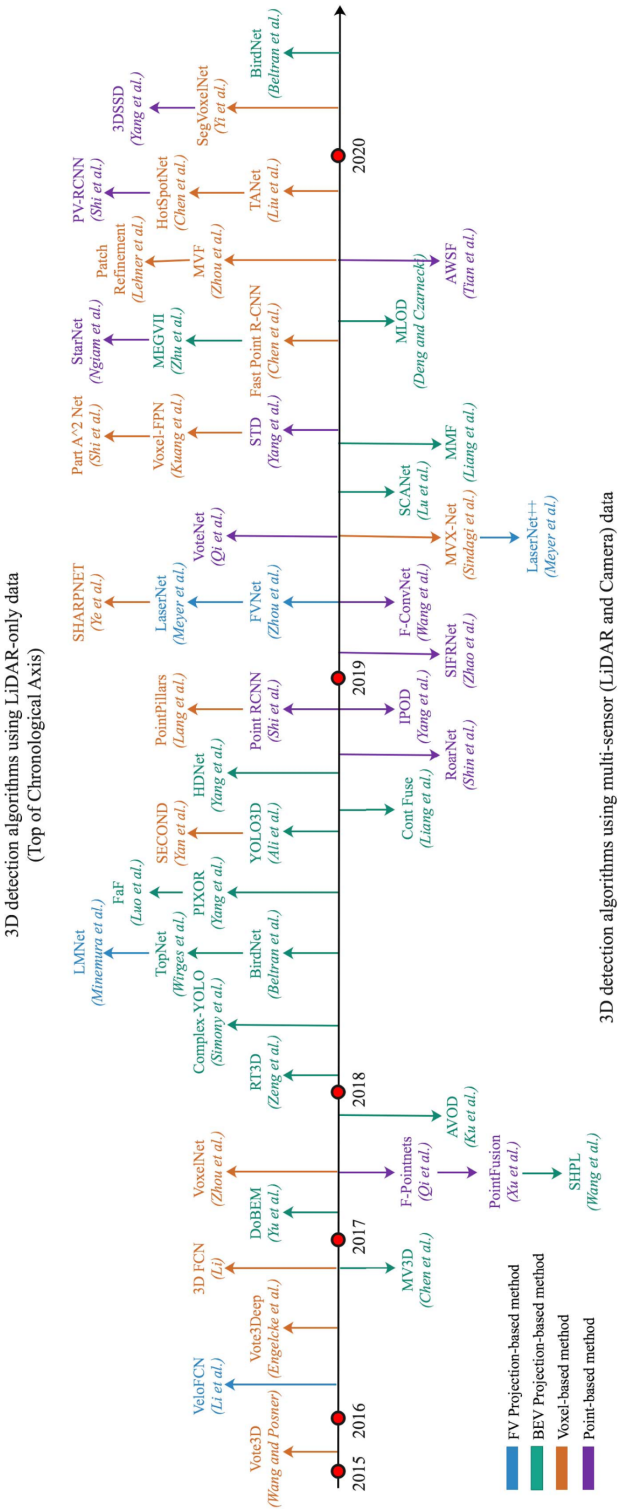


Figure 2.7: Chronological overview of 3D object detection algorithms using LiDAR data. [29]

- **Projection-Based Methods**

  The sparsity nature of point cloud makes it hard to learn through previously purposed 2D state-of-the-art neural networks. To overcome this problem, projection-based methods focused on viewing the point cloud data from a specific perspective. 3D point cloud is firstly projected into compact and ordered 2D map under a specific viewpoint. It is attractive to re-purpose well-established 2D Convolutional Neural Network(CNN) for 3D tasks. Many early 3D detection algorithms are projection-based methods.

  Depending on the projection surface of point cloud, projection-based methods can be divided into two subcate- gories: Front View (FV) based-methods([30], [31], [32], [33]), and Bird Eye View (BEV)-based methods.([34], [35], [36], [37])

- **Voxel-Based Methods**

  Voxel-based methods is to discretize the sparse point cloud into a neat matrix called voxel grids, and then to apply convolution filters. Each voxel contains unstructured points internally. Voxelization preserves the 3D structure of raw point cloud data.

  The challenges of this method lie in efficiently parsing the sparse voxels. Because sparse LiDAR point cloud leads to large number of empty voxels, meanwhile the 3D spatial computation grows exponentially with finer voxel resolution. ([38], [39], [40], [41])

- **Point-Based Methods**

  Instead of regularizing the raw point cloud into image grids or 3D voxels to apply convolution layers for feature extraction and detection which for sure loses natural geometry of points. Point based method directly extract the feature of raw point cloud to ease information loss.

  In 2017, a point cloud classification and segmentation algorithm PointNet [42] introduces a unified deep network architecture that directly consumes unordered and sparse point sets to capture the local and global point features. The idea of directly processing point cloud has quickly transferred to 3D related tasks. It becomes feasible to detect 3D objects irrespective of the point cloud format. Since then, point-based 3D detectors start to emerge. (Frustum-PointNet [43], F-ConvNet [44], RoarNet [45])

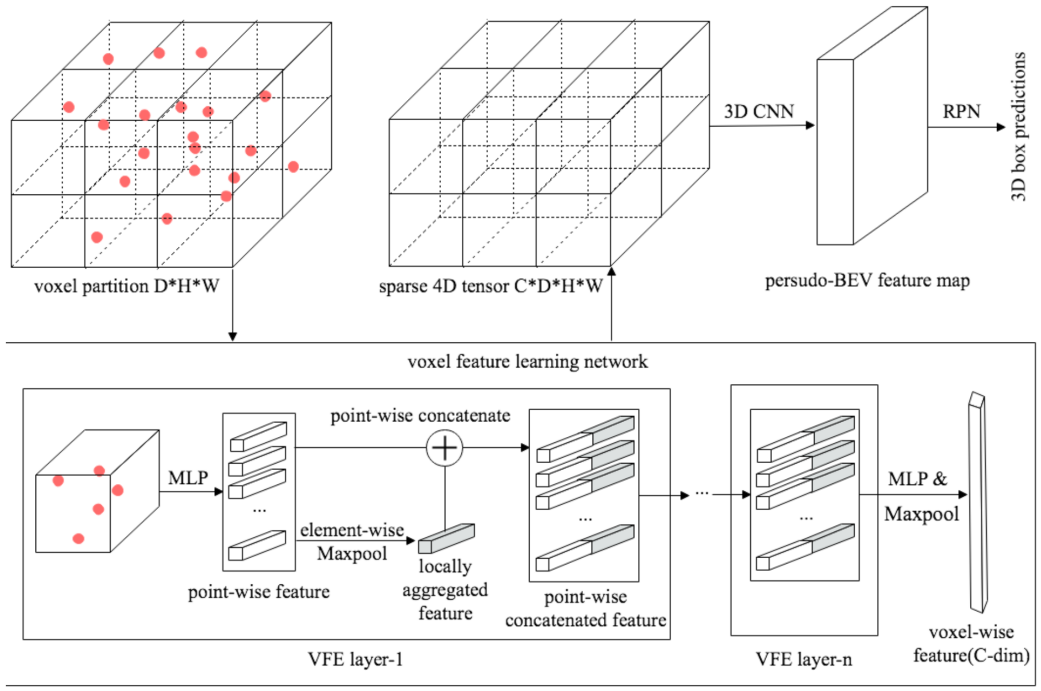


Figure 2.8: Network structure of a representative voxel-based 3D object detection network VoxelNet [40]

2.3 Classical Vision Based Cone Detection

Most of the classicial approaches use mono-camera as input device, and were hand crafted by unique feature of shape or color of the cone.

Color Detection

The RGB images recorded by the color camera are converted to Hue, Saturation, Value Colorspace (HSV) to separate color and luminance information. After that, the overall intensity of the orange color is calculated and compared with a threshold value. Detection based on object color is computationally very efficient, but vulnerable because many objects in the environment have similar colors, as shown in the figure below.

Triangle Detection

To detect the unique triangle shape of cones, a Canny Edge Detector is firstly applied, the RGB image from the color camera has to be converted into a grayscale image. Then, a Hough transformation is applied to locate lines in the image. The unique triangle shape of cones in the image can be found by computing and searching for the correct angle surrounded by two lines. Once a specific triangle shape of cones

is detected, the image can be classified as a cone. [46]

Figure 2.9: (1) correcctly and falsly (3) classified image using color detection, (2) and (4) the detected masks. [46]

Chapter 3

PROBLEM DESCRIPTION AND THE APPROACH

In short, our goal is to detect the traffic cones lying on the ground. We need to count the number of cones and localize them for later path planning and control tasks. Therefore, it requires us to output precise 3D location and number of cones in the scene.

Based on the uniqueness of our cone detection task, we can utilize ideal of PointPillars [41] and design a simplified version. Our method is devided into several sub-tasks:
First, we will voxelize point cloud into 3D grids and create pseudo BEV(bird eye view) images based on voxels. It means compressing the 3D point cloud into 2D images. Second, we will build 2D neural network based on common existing methods for object detection and train the networks. Since our problem is much simplier than the object detection challenges in autonomous driving community, we still retrive necessary information need for cone detection. Finally, we output the locations and number information from trained neural network.

Chapter 4

IMPLEMENTATION

In this chapter, we are going to present in detail our approaches of generating training data, prepossing, selection of model and training results.

4.1 Data Acquisition

A typical deep learning training process requires a huge amount of data, it is time consuming and expensive to acquire real point cloud data from LiDAR. In the scope of our project, we create artifical point cloud dataset for training. In order to resemble the real point cloud output LiDAR, we build an algorithm which imitates TOF process used in LiDAR.

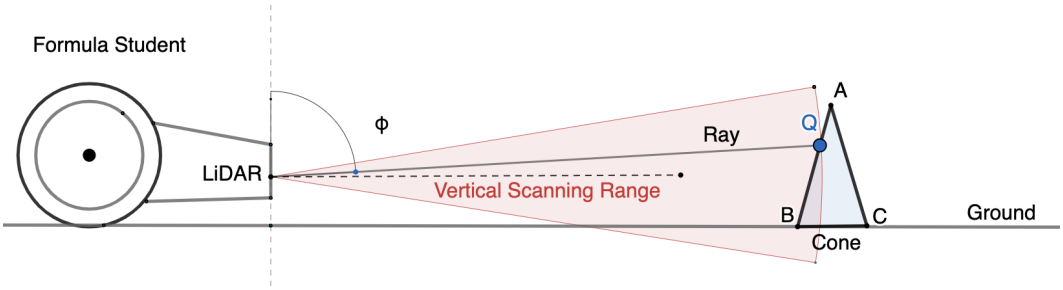


Figure 4.1: Side View of LiDAR Detection Process

First, we parameterize the Cone and LiDAR, we define the position of LiDAR and Cone as:

$$\mathbf{P}_{LiDAR} = [0, 0, h]^T, \quad \mathbf{P}_{Cone} = [x_0, y_0, 0]^T \tag{4.1}$$

where h is the height of LiDAR sensor.

Second, we write conical surface in the form:

$$\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} - \frac{(z - h_{Cone})^2}{c^2} = 0 \tag{4.2}$$

where h_{Cone} is the height of the Cone, (a, b, c) are constant parameters of conical surface.

Next, we define the direction of LiDAR ray as $\mathbf{r}$, and the intersection point of LiDAR ray and Cone as $\mathbf{Q}$.:

$$\mathbf{r} = [\sin\varphi\cos\theta, \quad \sin\varphi * \sin\theta, \quad \cos\varphi)]^T \tag{4.3}$$

$$\mathbf{Q} = \mathbf{P}_{LiDAR} + k\mathbf{r} = [k \sin\varphi \cos\theta, \quad k \sin\varphi * \sin\theta, \quad h + k\cos\varphi)]^T \tag{4.4}$$

Subs 4.4 into 4.2 we can solve k for given θ, φ.

$$k = \begin{pmatrix} \frac{0.1\,(\sigma_5 - \sigma_1 - 3.0\cos(\phi) + \sigma_3 + \sigma_4)}{\sigma_2} \\ \frac{0.1\,(\sigma_1 - 3.0\cos(\phi) + \sigma_5 + \sigma_3 + \sigma_4)}{\sigma_2} \end{pmatrix}$$

where

$$\begin{aligned}
\sigma_1 = &- 5.0 \; - 5.0\, sqrt[3.24 \sin(\phi)^2 - 10.8 \sin(2\phi)\;(x_0 \cos(\theta) + y_0 \sin(\theta)) \\
&- 21.6\, h \sin(\phi)^2 + 36.0\, h^2 \sin(\phi)^2 + 36.0\,(x_0{}^2 + y_0^2)\cos(\phi)^2 - 324.0\, y_0{}^2 \cos(\theta)^2 \\
&+ x_0{}^2 \sin(\phi)^2 \sin(\theta)^2 + 36.0\, h \sin(2\phi)\;(x_0 \cos(\theta) + y_0 \sin(\theta)) \\
&+ 648.0\, x_0\, y_0 \cos(\theta)\;\sin(\phi)^2 \sin(\theta)]
\end{aligned}$$

$$\sigma_2 = -1.0\cos(\phi)^2 + 9.0\sin(\phi)^2$$

$$\sigma_3 = 90.0\, x_0 \cos(\theta)\;\sin(\phi)$$

$$\sigma_4 = 90.0\, y_0 \sin(\phi)\;\sin(\theta)$$

$$\sigma_5 = 10.0\, h \cos(\phi)$$

(4.5)

We get two solutions of k, since the LiDAR ray would interact with the cone only once, we take the smaller one. The same technique can be applied for solving interaction point with ground, by parameterizing the surface of ground:

$$Ax + By + Cz = D \tag{4.6}$$

where flat ground has $A = B = D = 0, \quad C = 1$.

And the solution is:

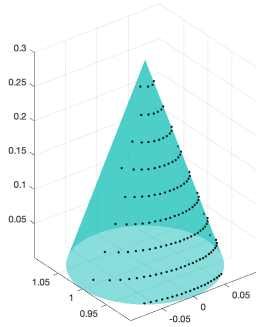
$$k = -\frac{h}{\cos(\phi)} \tag{4.7}$$

So far, we solved intersection points of LiDAR ray and surfaces for the given θ, φ. To simulate a Velodyne Puck LiDAR(16 channels), which is commonly used in autonomous vehicle industry, we list its useful specifications:

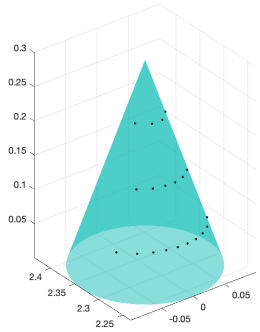
Name of Parameter	Value
Channels	16
Measurement Range	100m
Field of View(Vertical)	30°
Angular Resolution(Vertical)	2.0°
Field of View(Horizontal)	360°
Angular Resolution(Horizontal/Azimuth)	0.1°-0.4°
Rotation Rate	5 Hz - 20 Hz

Table 4.1: Specifications of Velodyne Puck

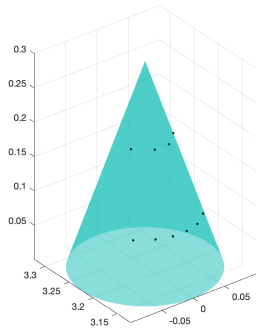
We selected vertical field of view as $82° < \phi < 112°$, angular resolution as 0.4°. For different distance between cone and LiDAR, we generated imitated 16-channel Velodyne LiDAR data, some samples are illustrated in figure 4.2. The whole catalog is provided in Appendix figure A.1



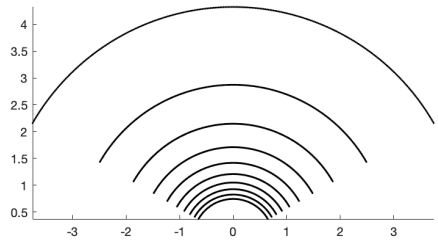
(a) Distance = 1 [m]



(b) Distance = 2.33 [m]



(c) Distance = 3.22 [m]



(d) Scanning of the Ground

Figure 4.2: Samples of LiDAR Scan

4.2 Data Augmentation

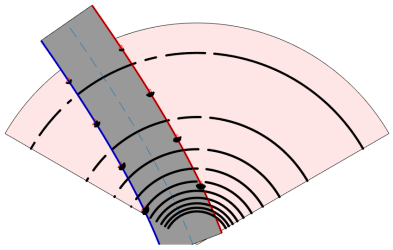
With imitated point cloud of cones at various distance and ground, we assembied each scene using scaling, translation and rotation transformation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.8}$$

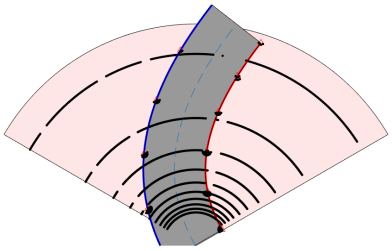
In order to create a rich and diverse dataset for training, we used 3rd order polynomials based on random parameters to generate a unique path for each scene, and each scene contains different number of cones.

4.3 Demonstration

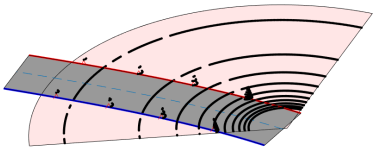
In figure 4.3, we present two set of scenes generated using the method we elaborated above. Where black points indicates point cloud generated by LiDAR mounted on the chasis; red sector is hotizontal scanning range; grey area is the track of Formula Student. We can notice the cones and track scanned by LiDAR.



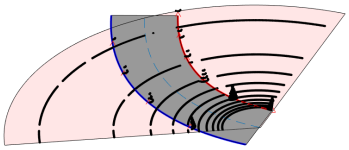
(a) Top view of scene 1



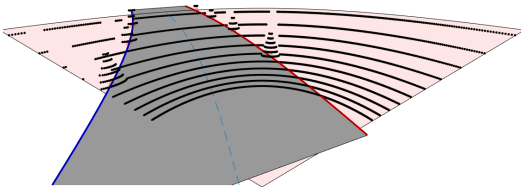
(b) Top view of scene 2



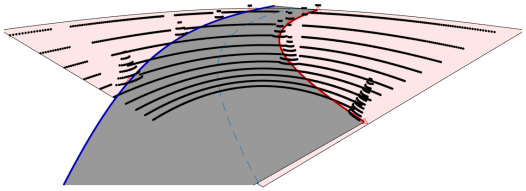
(c) Side view of scene 1



(d) Side view of scene 2



(e) Front view of scene 1



(f) Front view of scene 2

Figure 4.3: Samples of generated training data

In figure 4.3, we present ground truth of cone detection task.

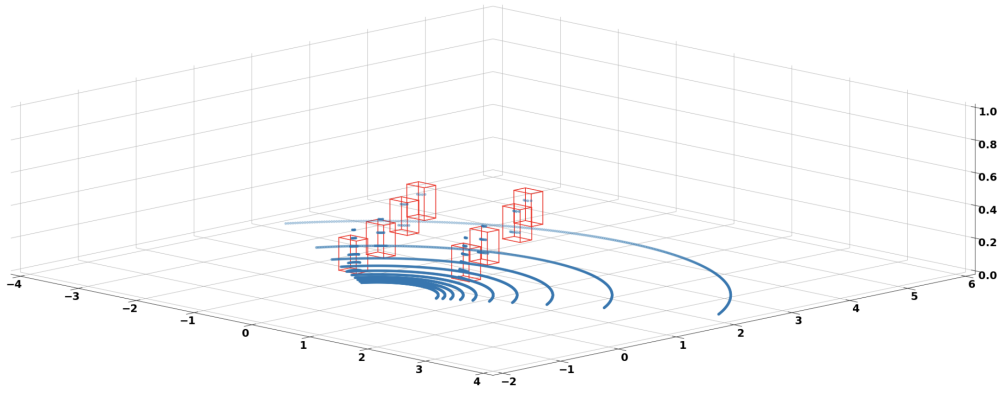


Figure 4.4: Ground Truth of one sample

4.4 Pre-processing

In this sub-section, we present our approaches of pre-processing point cloud data.

Our goal is to create BEV pseudo images from LiDAR scannings.
First, we need to voxelize the point cloud, the resolution is chosen as 0.1 for every axis, meaning every point in the same grid would be considered as a group.

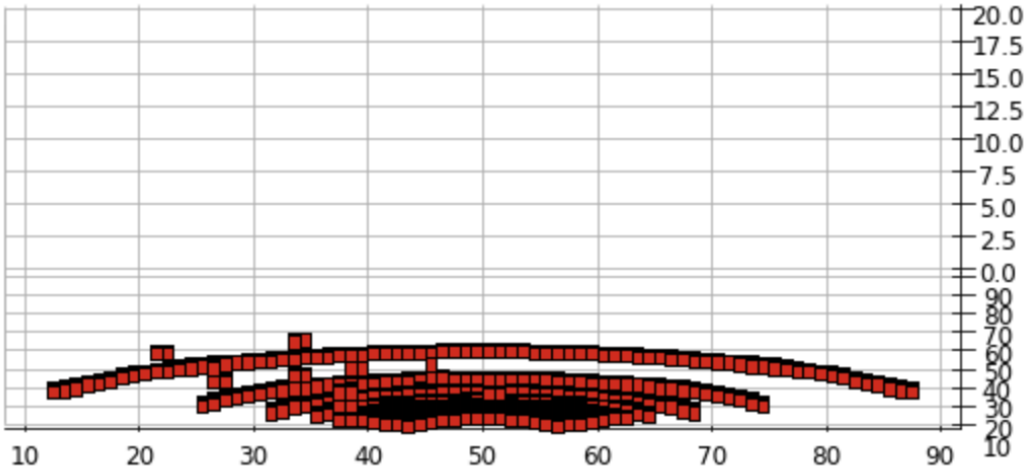


Figure 4.5: Voxelize point cloud

Then, based on voxels obtained, we create pseudo images by stacking voxels along z axis and highlighting voxels with larger z values.

As we can see from figure 4.6, the highlighted areas are where cones located.

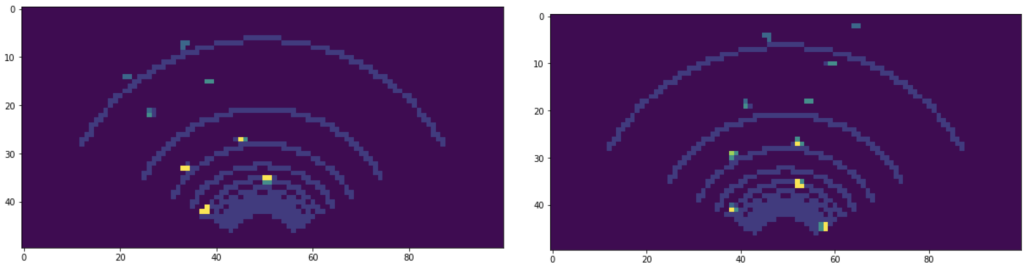


Figure 4.6: pseudo images

4.5 Training

Model

In this section, we present our training model and training results.

We used `pytorch` to train and build the model. Our model is a variational autoencoder, pseudo images are processed through three convolution and two max-pooling layers and then decoded through two transposed convolution layers. The final regression layer is a linear layer that outputs 200 features (coordinate candidates). The loss function is chosen as the mean squared error (squared L2 norm):

$$\ell(x, y) = L = l_1, \ldots, {l_N}^T, \; l_n = (x_n - y_x)^2$$

The model is shown in detail as:

```
Net(
(encoder): Sequential(
   (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1))
   (1): ReLU()
   (2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1))
   (3): ReLU()
   (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
   (5): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
   (6): ReLU()
   (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1)
   (8): Dropout(p=0.2, inplace=False)
)
(decoder): Sequential(
   (0): ConvTranspose2d(32, 16, kernel_size=(3, 3), stride=(2, 2))
   (1): BatchNorm2d(16, eps=1e-05, momentum=0.1)
   (2): ReLU()
   (3): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(2, 2))
   (4): BatchNorm2d(8, eps=1e-05, momentum=0.1)
)
```

```
(regression): Sequential(
   (0): Linear(in_features=37224, out_features=200, bias=True))
)
```

Results

We trained on NVIDIA® Tesla™ T4 GPU with 400 epochs, training time was 214[sec].

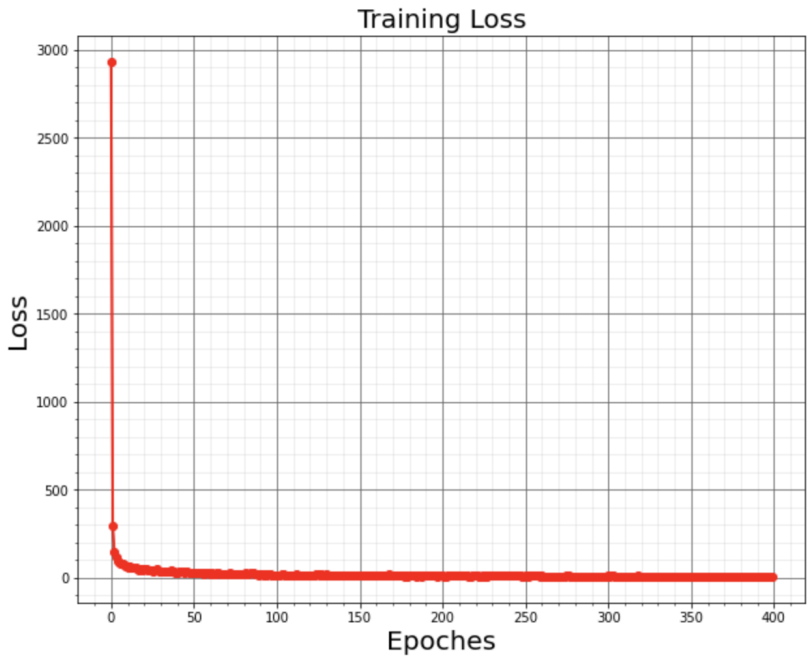


Figure 4.7: Model loss

The training loss quickly converged from thousands to less than `15` after `100` epochs. The average training loss of last `50` epochs was `7.8176`, testing loss was `45.3561`. After around `150` epochs, there was certain degree of overfitting.

After deep learning tasks, we perform a Non-maximum Suppression(NMS) algorithms on the results received and outputs final coordinates and number of cones detected. In a nut shell, NMS is a technique used in numerous computer vision tasks. It is a class of algorithms to select one entity (e.g., bounding boxes) out of many overlapping entities. We can choose the selection criteria to arrive at the desired results. The criteria are most commonly some form of probability number and some form of overlap measure (e.g. Intersection over Union). Before applying

NMS algorithm, our network output 100 candidate bounding boxes, as illustrated in figure 4.8

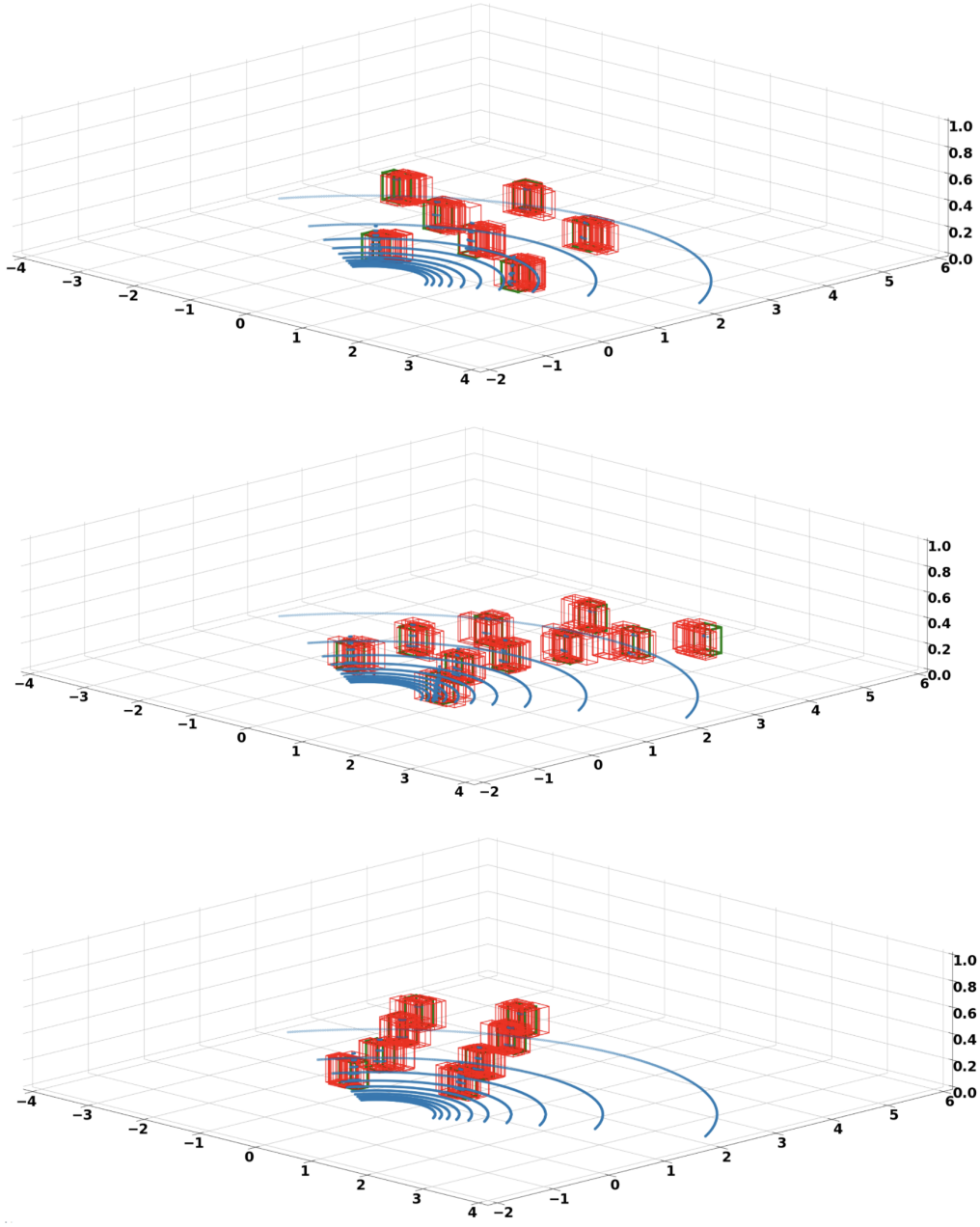


Figure 4.8: Before applying NMS algorithm, red boxes are outputs from linear regression layer, green ones are ground truth.

The algorithm of NMS is attached in appendix.

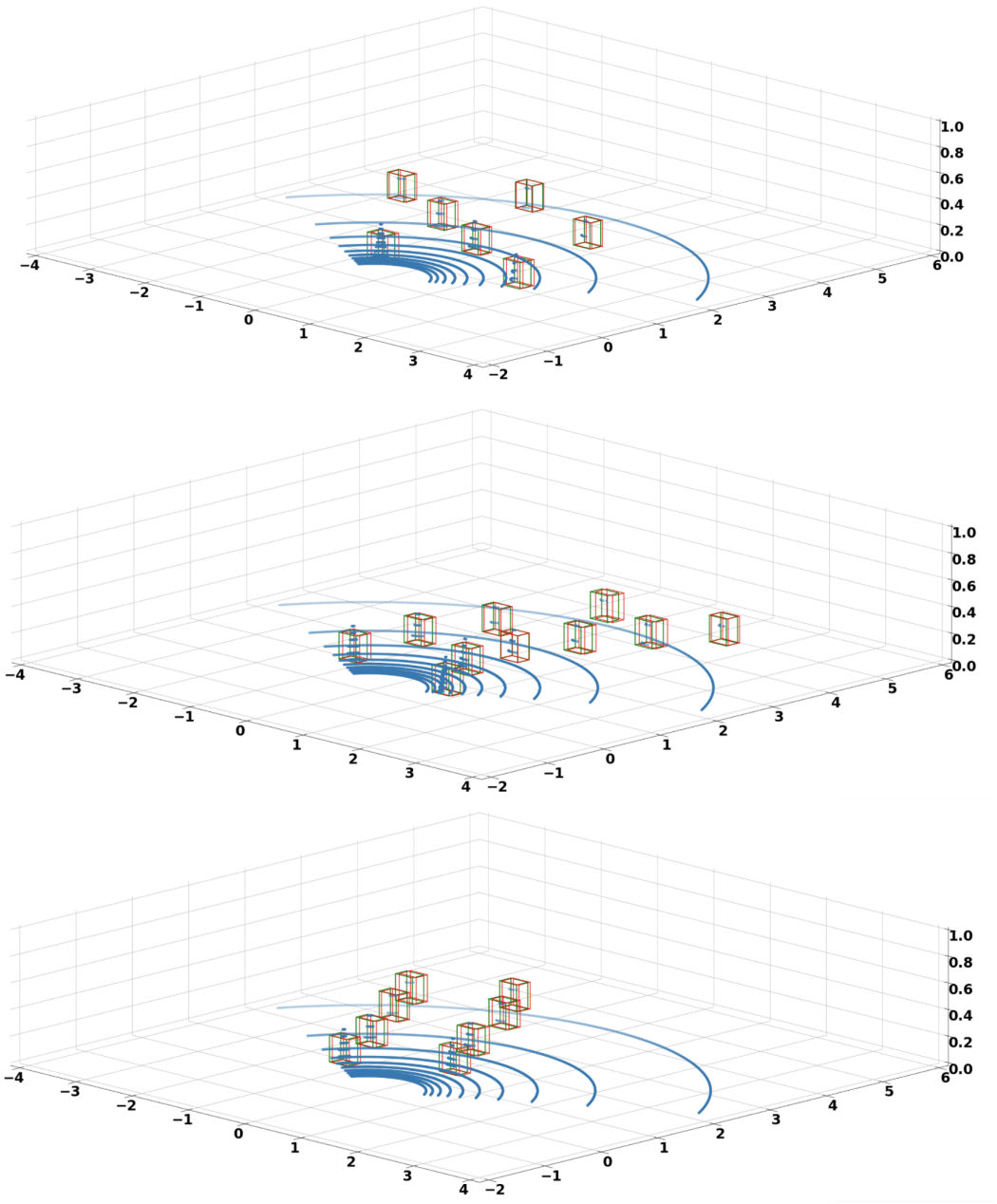


Figure 4.9: After applying NMS algorithm, red boxes are outputs from linear regression layer, green ones are ground truth.

Chapter 5

PERFORMANCE ANALYSIS

In this section, we are going to analyze our network's performance under noise and scale of the cones. We are still going to use the l2 loss function as a quantitative criteria for the comparasion.

5.1 Effect of noise

In the training stage, we used clean point cloud data. Here we tested the model performance when several noise level applied.

The noise we used for analysis is random number with normal distribution by `numpy.random.rand()` module. Mean value is set as `0` for all.

Model performance under normal-distributed noise		
Standard deviation	Training loss	Testing loss
0.01	12.8774	50.2453
0.02	13.1443	63.4528
0.03	15.91024	98.6002
0.04	14.0121	120.5060
0.05	16.1527	105.6436
0.06	16.3919	121.8947
0.07	15.5278	131.3905
0.10	22.4595	166.3808

Table 5.1: Table of Model Performance under noise.

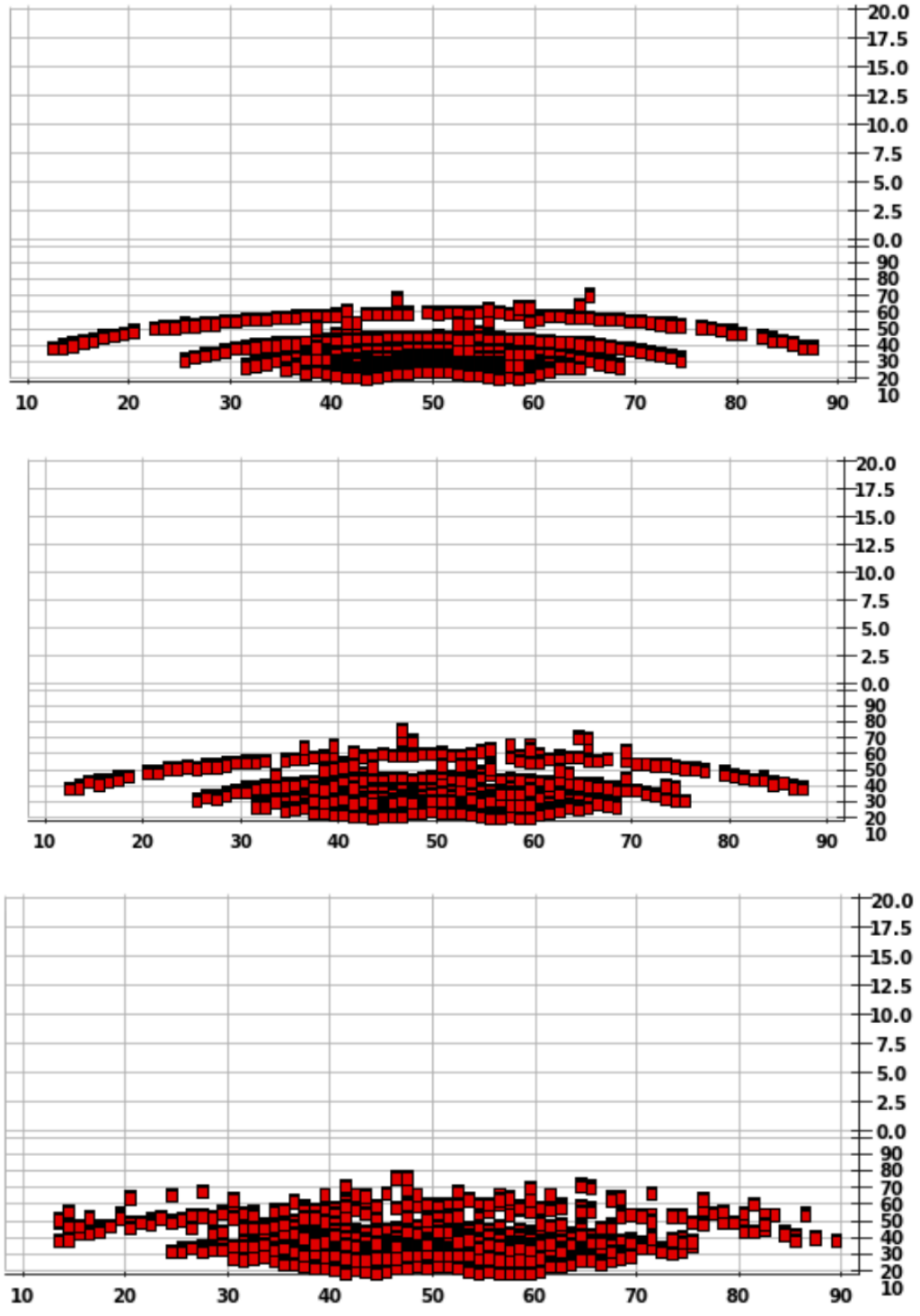


Figure 5.1: From top to bottom, voxels generated under noise level [0.02, 0.05, 0.10]

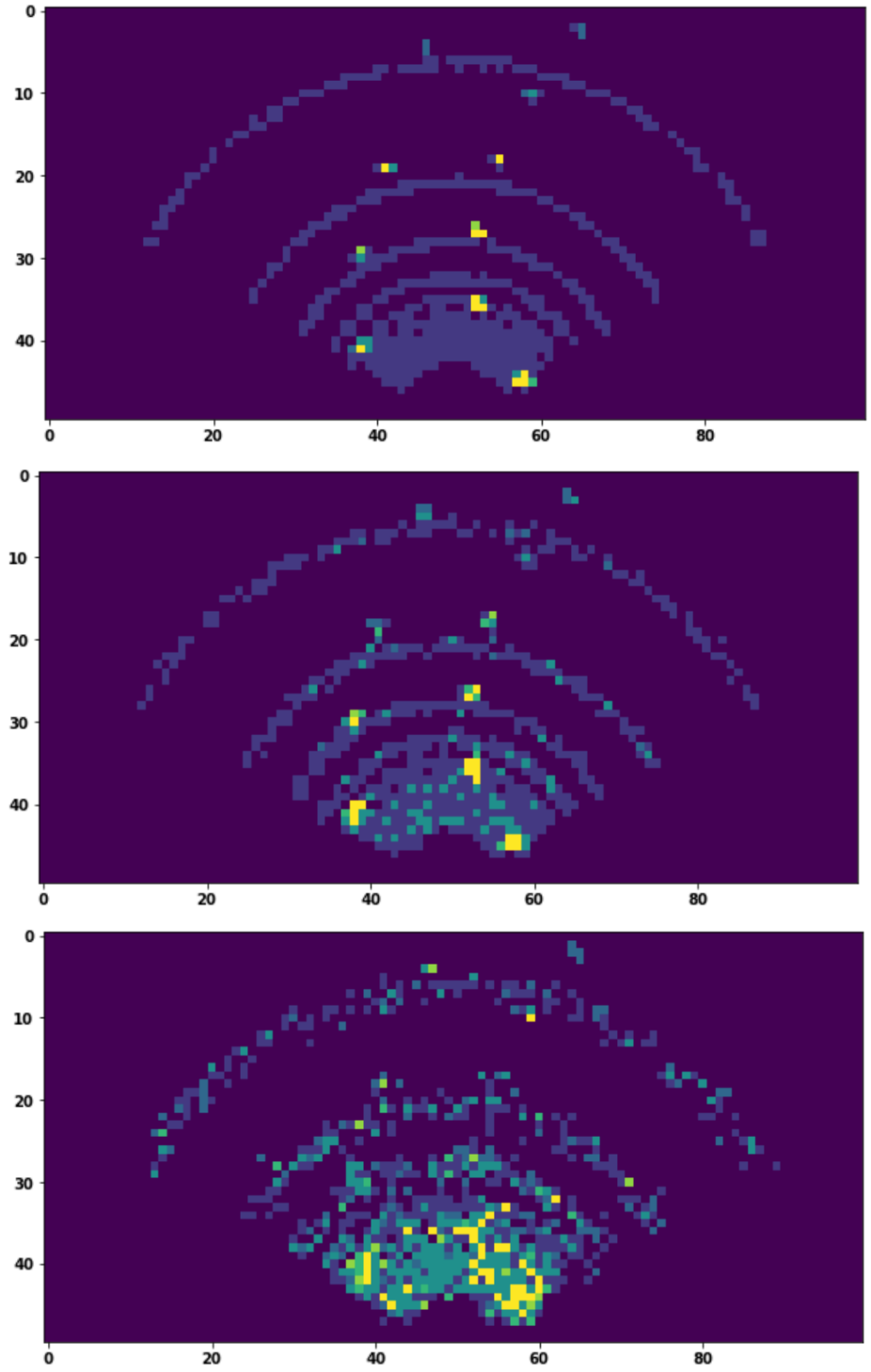


Figure 5.2: From top to bottom, pseudo BEV images generated under noise level [0.02, 0.05, 0.10]

It is more of a qualititative than quantitative explanation:
From simulation it is found that when the loss of model is over 130, the final locations and number of cones generated loss validality. In other words, the model is robust to noises with standard deviation < 0.07.
In principle, our model has high robustness in nature because of the voxelization process: The point cloud with small deviation did not create a different voxel map because the noise compared with resolution is small. Thus, the pseudo maps are also unchanged. The figures above 5.2, 5.1 illustrated that only when noise is larger than the resolution of voxels, there are qualititative change of both maps.
This implies that we may decrease resolution of voxels to suppress the noise. However, there's a trade-off: decreasing resolution would results impairing accuracy of localization, because the information of cone positions is impaired
This finding tells us that in real practice, we can suppress noise and improve the model performance by tunning the voxel resolution.

5.2 Effect of cone sizes

Although in real-life usage, cone size is fixed and known for the formula student competition, we still tested the model's performance under the effect of cone sizes.

Model performance with different cone sizes		
Size factor	Training loss	Testing loss
0.4	54.4626	234.2451
0.6	34.2451	189.3553
0.8	15.6548	70.4524
1.0	12.1474	45.5773
1.2	12.4637	42.2467
1.4	11.2473	38.1351
1.6	9.3534	35.2452
1.8	9.1245	32.3556

Table 5.2: Table of Model Performance with Different Cone Sizes.

From the results we found that, the detection performs better for larger cones and worse for smaller cones. As implied previously, the resolution of the voxel grids would affect the locatization of cones. For smaller cones, the resolution set for our original model would be too coarse and its position is undertermined. Solutions can be increasing resolution of voxels and reducing the subsampling layers (pooling, striding).

Besides the former mentioned influence of noise and scale, density of point cloud is also a common factor for performance analysis, however, it is not for our case and we can explain the reason immediately: Our model is first to convert the point cloud data to voxels and then create a pseudo map. Even the point cloud is diluted, as long as there are existance of one point in the space, it would be treated as filled voxel. So, the model's performance won't be affected.

Chapter 6

CONCLUSION

This project is motivated by driveless formula student challenges in 2017, our goal is to build a detection model for cone detection and localization using point cloud data from LiDAR.

We reviewed autonomous vehicle systems and various object detection models for autonomous driving, including 2D methods and 3D methods.

In order to get life-like training data, we parameterized the detection scene and computed the coordinates of points on cone models to simulate LiDAR scans. We pre-processed the point cloud to create voxels, then pseudo images for later detection.

The variational autoencoder model was designed and built, due to the uniqueness of the detection problem we used a fully connect layer for the final output of cone coordinate candidates. In the end, we used Non-maximum suppression algorithm to select the most promising candidates.

The model gave us relatively good results with low loss. We also analyzed the model's performance with training noise and cones of different scales.

Future work can be trying different detection methods: region purposal network (known as RCNN), and Single Shot MultiBox Detector (SSD). These methods has shown good performance in real-time detection applications.

BIBLIOGRAPHY

[1] D. Zadok, T. Hirshberg, A. Biran, K. Radinsky, and A. Kapoor, "Explorations and lessons learned in building an autonomous formula sae car from simulations," *Proceedings of the 9th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, 2019. DOI: 10.5220/0008120604140421. [Online]. Available: http://dx.doi.org/10.5220/0008120604140421.

[2] F. T. team. (). "Picture of ft2016," [Online]. Available: https://www.formulatechnion.com/2016.html.

[3] h. TIAN, J. NI, and J. HU, "Autonomous driving system design for formula student driverless racecar," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1–6. DOI: 10.1109/IVS.2018.8500471.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[5] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, "The apolloscape dataset for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 954–960.

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[7] P. L. Liu. (). "Monocular 3d object detection in autonomous driving — a review," [Online]. Available: https://towardsdatascience.com/monocular-3d-object-detection-in-autonomous-driving-2476a3c7f57e.

[8] Y. Kim and D. Kum, "Deep learning based vehicle position and orientation estimation via inverse perspective mapping image," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 317–323. DOI: 10.1109/IVS.2019.8814050.

[9] T. Roddick, A. Kendall, and R. Cipolla, "Orthographic feature transform for monocular 3d object detection," *arXiv preprint arXiv:1811.08188*, 2018.

[10] S. Srivastava, F. Jurie, and G. Sharma, "Learning 2d to 3d lifting for object detection in 3d for autonomous vehicles," *arXiv preprint arXiv:1904.08494*, 2019.

[11] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8445–8453.

[12] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teulière, and T. Chateau, "Deep MANTA: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image," *CoRR*, vol. abs/1703.07570, 2017. arXiv: 1703.07570. [Online]. Available: http://arxiv.org/abs/1703.07570.

[13] A. Kundu, Y. Li, and J. M. Rehg, "3d-rcnn: Instance-level 3d object reconstruction via render-and-compare," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3559–3568.

[14] F. Manhardt, W. Kehl, and A. Gaidon, "ROI-10D: monocular lifting of 2d detection to 6d pose and metric shape," *CoRR*, vol. abs/1812.02781, 2018. arXiv: 1812.02781. [Online]. Available: http://arxiv.org/abs/1812.02781.

[15] I. Barabanau, A. Artemov, E. Burnaev, and V. Murashkin, "Monocular 3d object detection via geometric reasoning on keypoints," *CoRR*, vol. abs/1905.05618, 2019. arXiv: 1905.05618. [Online]. Available: http://arxiv.org/abs/1905.05618.

[16] T. He and S. Soatto, "Mono3d++: Monocular 3d vehicle detection with two-scale 3d hypotheses and task priors," *CoRR*, vol. abs/1901.03446, 2019. arXiv: 1901.03446. [Online]. Available: http://arxiv.org/abs/1901.03446.

[17] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," *CoRR*, vol. abs/1612.00496, 2016. arXiv: 1612.00496. [Online]. Available: http://arxiv.org/abs/1612.00496.

[18] L. Liu, J. Lu, C. Xu, Q. Tian, and J. Zhou, "Deep fitting degree scoring network for monocular 3d object detection," *CoRR*, vol. abs/1904.12681, 2019. arXiv: 1904.12681. [Online]. Available: http://arxiv.org/abs/1904.12681.

[19] A. Naiden, V. Paunescu, G. Kim, B. Jeon, and M. Leordeanu, "Shift R-CNN: deep monocular 3d object detection with closed-form geometric constraints," *CoRR*, vol. abs/1905.09970, 2019. arXiv: 1905.09970. [Online]. Available: http://arxiv.org/abs/1905.09970.

[20] H. M. Choi, H. Kang, and Y. Hyun, "Multi-view reprojection architecture for orientation estimation," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, IEEE, 2019, pp. 2357–2366.

[21] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang, "Gs3d: An efficient 3d object detection framework for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1019–1028.

[22] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2147–2156.

[23] A. Simonelli, S. R. Bulò, L. Porzi, M. López-Antequera, and P. Kontschieder, "Disentangling monocular 3d object detection," *CoRR*, vol. abs/1905.12365, 2019. arXiv: 1905.12365. [Online]. Available: http://arxiv.org/abs/1905.12365.

[24] E. Jörgensen, C. Zach, and F. Kahl, "Monocular 3d object detection and box fitting trained end-to-end using intersection-over-union loss," *CoRR*, vol. abs/1906.08070, 2019. arXiv: 1906.08070. [Online]. Available: http://arxiv.org/abs/1906.08070.

[25] G. Brazil and X. Liu, "M3D-RPN: monocular 3d region proposal network for object detection," *CoRR*, vol. abs/1907.06038, 2019. arXiv: 1907.06038. [Online]. Available: http://arxiv.org/abs/1907.06038.

[26] M. Ding, Y. Huo, H. Yi, Z. Wang, J. Shi, Z. Lu, and P. Luo, "Learning depth-guided convolutions for monocular 3d object detection," *CoRR*, vol. abs/1912.04799, 2019. arXiv: 1912.04799. [Online]. Available: http://arxiv.org/abs/1912.04799.

[27] Z. Qin, J. Wang, and Y. Lu, "Triangulation learning network: From monocular to stereo 3d object detection," *CoRR*, vol. abs/1906.01193, 2019. arXiv: 1906.01193. [Online]. Available: http://arxiv.org/abs/1906.01193.

[28] F. Amzajerdian, "Role of lidar technology in future nasa space missions," *MRS Online Proceedings Library*, vol. 1076, no. 1, pp. 1–6, 2008.

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

[30] J. Zhou, X. Tan, Z. Shao, and L. Ma, "Fvnet: 3d front-view proposal generation for real-time object detection from point clouds," in *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, IEEE, 2019, pp. 1–8.

[31] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, "Lasernet: An efficient probabilistic 3d object detector for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 677–12 686.

[32] K. Minemura, H. Liau, A. Monrroy, and S. Kato, "Lmnet: Real-time multiclass object detection on cpu using 3d lidar," in *2018 3rd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, IEEE, 2018, pp. 28–34.

[33] B. Li, T. Zhang, and T. Xia, "Vehicle detection from 3d lidar using fully convolutional network," *arXiv preprint arXiv:1608.07916*, 2016.

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

[35] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1–8.

[36] J. Deng and K. Czarnecki, "Mlod: A multi-view 3d object detection based on robust feature fusion method," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 279–284.

[37] H. Lu, X. Chen, G. Zhang, Q. Zhou, Y. Ma, and Y. Zhao, "Scanet: Spatial-channel attention network for 3d object detection," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 1992–1996.

[38] D. Z. Wang and I. Posner, "Voting for voting in online point cloud object detection.," in *Robotics: Science and Systems*, Rome, Italy, vol. 1, 2015, pp. 10–15.

[39] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 1355–1361.

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

[41] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

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

[43] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3d object detection from rgb-d data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 918–927.

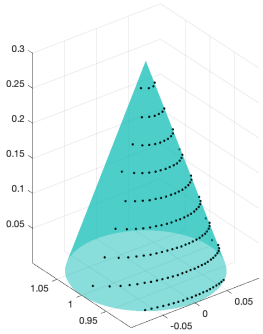
[44] Z. Wang and K. Jia, "Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 1742–1749.

[45] K. Shin, Y. P. Kwon, and M. Tomizuka, "Roarnet: A robust 3d object detection based on region approximation refinement," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2019, pp. 2510–2515.

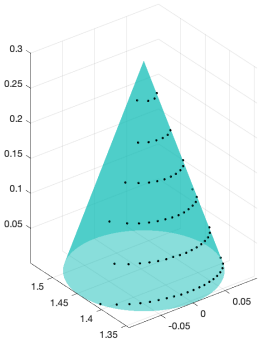
[46] N. Messikommer, S. Schaefer, R. Dubé, and M. Pfeiffer, "Cone detection using a combination of lidar and vision-based machine learning," *arXiv preprint arXiv:1711.02079*, 2017.

Appendix A

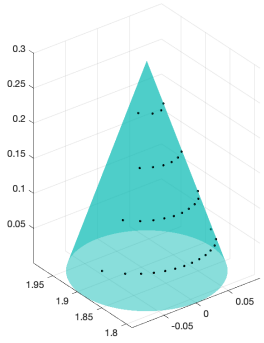
LIDAR SCAN OF CONE AT VARIOUS DISTANCE



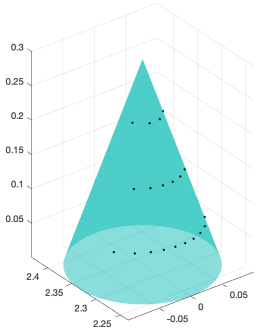
(a) Distance = 1 [m]



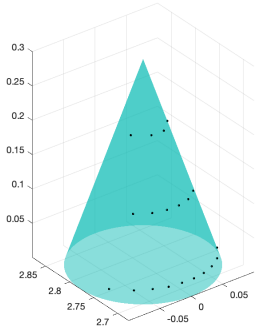
(b) Distance = 1.44 [m]



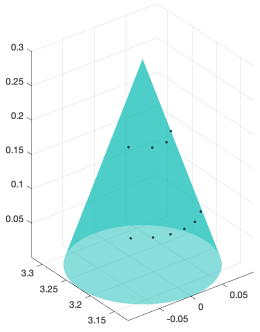
(c) Distance = 1.89 [m]



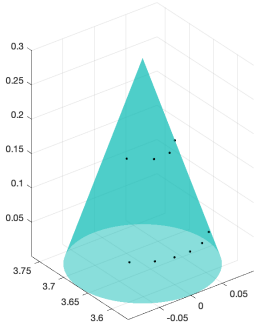
(d) Distance = 2.33 [m]



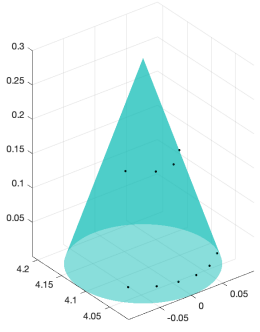
(e) Distance = 2.78 [m]



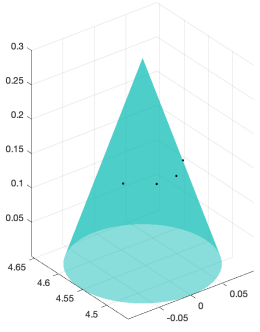
(f) Distance = 3.22 [m]



(g) Distance = 3.67 [m]



(h) Distance = 4.11 [m]



(i) Distance = 4.56 [m]

Figure A.1: LiDAR Scan of Cone at Various Distance

```
def non_max_suppression(boxes, overlapThresh):
# initialize the list of picked indexes
pick = []
# grab the coordinates of the bounding boxes
x1 = boxes[:,0]
y1 = boxes[:,1]
x2 = boxes[:,2]
y2 = boxes[:,3]
# compute the area of the bounding boxes and sort the bounding
# boxes by the bottom-right y-coordinate of the bounding box
area = (x2 - x1 + 1) * (y2 - y1 + 1)
# print(area)
idxs = np.argsort(y2)
# keep looping while some indexes still remain in the indexes
# list
while len(idxs) > 0:
   # grab the last index in the indexes list and add the
   # index value to the list of picked indexes
   last = len(idxs) - 1
   i = idxs[last]
   pick.append(i)
   # find the largest (x, y) coordinates for the start of
   # the bounding box and the smallest (x, y) coordinates
   # for the end of the bounding box
   xx1 = np.maximum(x1[i], x1[idxs[:last]])
   yy1 = np.maximum(y1[i], y1[idxs[:last]])
   xx2 = np.minimum(x2[i], x2[idxs[:last]])
   yy2 = np.minimum(y2[i], y2[idxs[:last]])
   # compute the width and height of the bounding box
   w = np.maximum(0, xx2 - xx1 + 1)
   h = np.maximum(0, yy2 - yy1 + 1)
   # compute the ratio of overlap
   overlap = (w * h) / area[idxs[:last]]
   # delete all indexes from the index list that have
   idxs = np.delete(idxs, np.concatenate(([last],
      np.where(overlap > overlapThresh)[0])))
# return only the bounding boxes that were picked using the
# integer data type
return boxes[pick].astype("float")
```

The end. Thanks for review.