# OBJECT LOCALIZATION IN 3D POINT CLOUD

**CHIN WAI LOK**

**A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Mechatronics Engineering**

**Lee Kong Chian Faculty of Engineering and Science**
**Universiti Tunku Abdul Rahman**

**April 2021**

# DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : CHIN WAI LOK

ID No. : 1602556

Date : 3 May 2021

# APPROVAL FOR SUBMISSION

I certify that this project report entitled **OBJECT LOCALIZATION IN 3D POINT CLOUD** was prepared by **CHIN WAI LOK** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Mechatronics Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : 

Supervisor : Dr. Ng Oon-Ee

Date : 3 May 2021

# ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Ng Oon-Ee for his invaluable advice, guidance and his enormous patience throughout the development of the research. In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement.

# ABSTRACT

Object localization in point clouds can help search for the target objects in the extensive 3D search space. It allows the post-operation of object recognition to operate on the objects more efficiently. There are many published works for object localization in 3D point clouds. Each approach has a unique architecture in its work. Thus, the frameworks used are not standardized like with 2D object localization frameworks. This work focuses on developing a method to locate objects in a point cloud and measure the objects' three primary dimensions accurately. The intra and inter-comparison and evaluation of the selected work are conducted to discuss its significance in 3D object localization. Comparison and evaluation of method(s) are standardized by average precision outputted using the same evaluation metrics, the KITTI offline evaluation dataset. Point-GNN is selected as the approach for 3D object localization. It works best when iterated twice in the edges and vertices' feature aggregation. Besides, Point-GNN scored second among the twelve 3D object localization approaches discussed. It achieves the AP predicted on the KITTI test 3D detection benchmark of 88.33 % for 'easy' car, 79.47 % for 'moderate' cars, and 72.29 % for 'hard' cars.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF SYMBOLS / ABBREVIATIONS

| | |
|---|---|
| 2D | two-dimensional |
| 3D | three-dimensional |
| 3DSSD | point-level 3D single-stage object localizer |
| 4D | four-dimensional |
| AP | average precision |
| API | application programming interface |
| BEV | bird's eye view |
| BirdNet | 3D object localization framework from BEV LiDAR |
| C-GCN | contextual graph convolutional network |
| CNN | convolutional neural network |
| ContFuse | deep continuous fusion framework |
| D-FPS | 3D Euclidean distance-based furthest point sampling |
| F-ConvNet | frustum-based convolutional neural network |
| F-FPS | feature distance-based furthest point sampling |
| F-PointNets | frustum-based PointNets |
| FCN | fully convolutional network |
| FPS | frames per second |
| GB | gigabytes |
| GCN | graph convolutional network |
| GNN | graph neural network |
| IoU | intersection-over-union |
| LiDAR | light detection and ranging |
| MLP | multilayer perceptron |
| MV3D | multi-view 3D object localization network |
| NMS | non-maximum suppression |
| PIXOR | oriented 3D object localization from pixel-level neural network predictions |
| Point-GNN | graph neural network for 3D object localization in point clouds |
| PointNet | deep learning on point sets network |
| PointRCNN | region-based convolutional neural network for 3D object localization |

| | |
|---|---|
| PointRGCN | region-based graph convolutional network for 3D object localization |
| R-GCN | residual graph convolutional network |
| RAM | random access memory |
| ReLU | rectified linear unit |
| RGB | red, green, and blue |
| RGB-D | red, green, and blue with depth |
| RoI | region of interest |
| RPN | region proposal network |
| UV | ultraviolet |
| VFE | voxel feature encoding |
| Vote3Deep | 3D object localization with a feature-centric voting scheme |
| VoxelNet | voxel-based object localization framework |

# LIST OF APPENDICES

# CHAPTER 1

# INTRODUCTION

## 1.1 General Introduction to Object Localization in Point Cloud

The point cloud is a type of representation of objects and scenes in three-dimensional (3D) space. There are many other representations to describe objects and scenes, including multi-view red, green, and blue with depth (RGB-D) images, volumetric, polygonal mesh, and primitive-based computer-aided design models. Along with point clouds, these representations are categorized into two classes: rasterized form and geometric form, which are regular and irregular in terms of the data's nature, respectively. Point cloud belongs to the geometric form of 3D data (Engelcke, et al., 2017). A point cloud is formed by a compilation of points in 3D space, where each point allocated in the space represents the X, Y, and Z geometric coordinates of the point (FME Community, 2020). Therefore, a point cloud can represent an object or a scene, where it can be built up from a collection of points. Many single spatial measurements are collated into a dataset to represent the object or the scene as a whole (Gray, n.d.). The dimensional complexity of the point cloud can be increased by adding new features to the points, such as colour information. Point clouds are the raw 3D data obtained from the 3D laser scanners and light detection and ranging (LiDAR) technology and techniques. LiDAR is usually implemented to obtain the data from a scene because it measures the distance between the sensor and the target object using ultraviolet (UV) rays (Thomson, 2019). It computes the distance by measuring the time lapse between emitting and receiving back the same UV pulse (Singh, 2018). For this reason, LiDAR is widely implemented in various emerging technologies to utilize its distance measurement mechanism, including autonomous vehicles and inspection at the building's surface.

Object localization is one of the two components in object detection, where it usually complements object recognition to localize and recognize the objects in space simultaneously (Brownlee, 2019). Object localization is used to locate the objects in the space accurately, wherein the object's coordinates are being determined. Object recognition, in contrast, is used to recognize the

located objects, where the label of the object is indicated. Object localization is the first step in the object detection process. It involves searching for an extensive search space to address the objects' profiles.

Object localization is achieved by determining bounding boxes that lie around the target objects to inform the locations and coordinates of the objects in space through visualization or integers (Brownlee, 2019). Bounding boxes regression is a method to describe the target object's dimensions. The lines of a two-dimensional (2D) bounding box illustrate an object's width and height in 2D space. This is where the measurement of the located object's primary dimensions is conducted. Object localization is usually performed with deep learning, where the features within the space are deeply abstracted (Zhao, et al., 2019). Feature abstraction is a proportion in the dimensionality reduction process that extracts only the essential features and neglect the less important ones without introducing much information loss (Alkabawi, Hilal and Basir, 2017).

The difference between object localization in 2D and 3D space is that the information density is higher in a 2D image while having low data volume. All information in the image is encoded into a frame, and each neighbouring pixel is meaningful for feature abstraction. Whereas in 3D space, the points are sparsely located. There exist regions with no points presence due to the absence of objects in that region. The empty regions are constituted as a part of the 3D data. It will be accounted as computational waste because there will be no output generated from these regions (Engelcke, et al., 2017). Since the objects in point clouds have 3D coordinates, the 2D bounding boxes required to concatenate an additional dimension of length to become 3D bounding boxes to cover and locate the objects in 3D space.

## 1.2 Importance of the Study

The study of object localization in 3D space, like point clouds, over 2D images is significant for several reasons. Point clouds have richer information context than 2D images to represent scenes and objects. Besides, occlusion is the primary drawback in a 2D image. It is computationally inefficient to synthesize the occluded objects when they can be found originally in a 3D point cloud. Furthermore, point clouds contain depth information that could

retain the object's original appearance in the actual situation of a 3D space. Object localization in point clouds can help search for the target objects in the extensive 3D search space. It allows the post-operation of object recognition to operate on the objects more efficiently.

There are many published works for object localization in 3D point clouds. Each approach has a unique architecture in its work. Thus, the frameworks used are not standardized like with 2D object localization frameworks. PointNet is the first significant point cloud processing work proposed by Qi, et al. (2017). Its architecture was widely implemented by many later works to develop new methods for object localization. Compared to the current state-of-the-art, PointNet's result in terms of the accuracy of localizing objects has been overtaken by newly emerged approaches. This work focuses on developing a method to locate objects in a point cloud and measure the objects' three primary dimensions accurately. In addition, intra and inter-comparison and evaluations of the selected work will be conducted to discuss its significance in 3D object localization.

## 1.3 Problem Statement

Even though point clouds are rich in information to represent objects and scenes, the points' overall density is generally low due to the smaller number of objects in certain regions in the 3D space. This phenomenon is significant in the data produced from the LiDAR sensor. LiDAR sensor produces sparse point clouds and can be occluded by objects as the 3D information behind a light-reflecting or non-reflective object cannot be detected, where the LiDAR UV lights cannot reach the target objects. This will cause computational inefficiency to perform object localization in the sparse point cloud. Besides, a point cloud is costly to an extent to be produced, and it is essential to acquire a 3D point cloud dataset for use in this project. Figure 1.1 shows a scene obtained from the KITTI dataset that is captured with a LiDAR sensor.

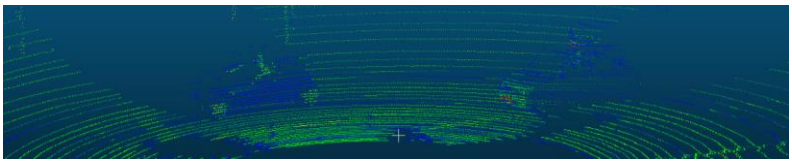


Figure 1.1: LiDAR Point Cloud with Cars (Geiger, et al., 2013).

Object localization is already implemented in the analysis of 2D images. It has reached maturity, where the architecture of the convolutional neural network (CNN) is leading the field of 2D object localization. CNN has a robust operation of detecting edges and sharp corners of objects in a 2D image. The first few convolutional layers detect the low-level features, and high-level features are detected by the last few layers. However, things changed when the dimension of the input data increases. The additional dimension increases the complexity in processing those high dimension data. The increased dimension should be treated as additional information to the subject instead of being ignored.

Object localization in 3D space is not as mature as in 2D images. This is because processing and using 3D space, such as point cloud, is computationally expensive. The increased dimensional complexity renders a different architecture design needed to handle the 3D point cloud data. Older techniques of CNN cannot be applied directly to point clouds. This is because the feature abstraction techniques have their dimensional preference to work on the data. Some feature engineerings should be performed earlier to change the structure and the characteristics of the point cloud before feature abstraction.

Apart from object localization, the measurement of the target object's primary dimensions is also essential for post-processing. These dimensions will help create a volumetric object where the object's size and volume can be estimated, rather than referring to the target object as a single point in 3D space. The measurement of the object's dimensions can be done by calculating the bounding box's dimensions. The bounding box is surrounding the target object securely and represents the object's size closely.

Deep learning on point sets network (PointNet) and PointNet++, both proposed by Qi, et al. (2017) are the basic deep learning techniques to process point clouds. It contains multiple functional architectures that can be implemented in future architecture design for feature abstraction from the point cloud. These two approaches can be applied in various point cloud applications, such as object localization, object classification, and point segmentation. The continuous grouping of points in the hierarchical convolutional layers from PointNet++ influences many object localization

frameworks proposed later to abstract the local features to detect fine-grained patterns and produce generalization to complex scenes. However, PointNet's architecture is rarely implemented by the current state-of-the-art, where its accuracy of localizing objects has been overtaken by newly emerged approaches. This causes the new approaches to have a unique architecture in their work. Therefore, the frameworks used are not standardized like with 2D object localization frameworks.

## 1.4 Aim and Objectives

This project aims to explore the methods and approaches for localizing objects and measuring their dimensions within 3D point clouds. The achievement of the aim will determine the best 3D object localization approach that localizes objects accurately. The method explored should be able to perform object localization in point clouds accurately and reliably. Hence, the following objectives need to be achieved to assist in achieving this aim:

1) Identify existing rich scenes of a 3D point cloud dataset for use in this project.
2) Develop a method for matching an object, either 2D or 3D, to objects within the point cloud.
3) Measure the primary dimensions of the located object.
4) Perform intra and inter-comparison and evaluations of the approach.

The first objective is to obtain a rich point cloud dataset with sufficient difficulty representing a scene but not an object. This objective allows object localization to operate on a scene rather than a fully segmented object. Next, the second and the third objective is to develop an object localization method with dimensions measuring capability. Many pieces of works of literature on related technology will be reviewed to study the existing frameworks. This will help develop the method in this project, where the feature engineering in the literature helps abstract most of the essential features in the scene to perform object localization. After that, the approach is compared internally and externally using a general standard.

## 1.5 Scope and Limitation of the Study

This project's scope focuses on the software component, whereas the hardware part is omitted and not part of the project's consideration, including the computation speed. Besides, no user interface is required to be designed in this project. In this study, object recognition is also ignored to focus on the object localization with the target object's coordinates and the bounding box predicted. By following the existing frameworks, the KITTI test dataset is used as the input data for this project and the evaluation metrics for the frameworks. This project was carried out for a year and concluded by this final report.

## 1.6 Contribution of the Study

This project outlines the existing 3D object localization frameworks and provides insights into each of them. Each framework is evaluated and discussed based on its accuracy in localizing the object and measuring the dimension, as well as its simplicity in delivering the outputs. The best framework is selected, and it was modified to improve the results further.

## 1.7 Outline of the Report

This project is further separated into several parts: literature review, research methodology, results and discussion, and recommendation. There are five chapters included in this final report. The first chapter: the introduction, discusses the relationship between object localization and point clouds and their significance. Problems faced by the current technologies are identified, and aim and objectives are stated to solve the problem. The introduction ends with an overview of the project. The second chapter then reviews multiple works of literature of different approaches to object localization in point clouds, including region proposal-based and single-shot approaches. This is to provide insights into current object localization technologies in developing a powerful object localization method. The third chapter then describes the project's methodology, where the procedures for achieving the objectives are stated, as well as the planning and time allocation of this project over a year. Problems encountered and their solutions during the project's execution are also included. The fourth chapter is the results and discussion of this project.

The results are presented and explained in-depth with supporting statements. Lastly, the fifth chapter will conclude this report and the project with the recommendations suggested for future work.

# CHAPTER 2

# LITERATURE REVIEW

There have been many 3D object detection approaches introduced, where each of them included object localization as part of the goal. Most of the approaches are proposed for implementation in autonomous vehicle applications, the current most popular technological advancement. Most of the approaches implement deep learning, specifically deep neural networks, to extract the features through complex feature abstraction architecture. Object localization is separated into two types of approaches, where the behaviour of the point cloud processing to regress 3D bounding boxes are different.

Region proposal-based approaches propose several regions in the point cloud with the potential of having interesting objects to be detected. Calculations of a significant number of 3D bounding boxes as proposals are being done in the first stage. After that, features are extracted from the region to refine the proposals' locations before proceeding to the final 3D bounding box refinement. This approach branches more into three specific main classes depending on the working principle: frustum-based approaches, multi-view-based approaches, and segmentation-based approaches (Guo, et al., 2020). These approaches generally have fully integrated computation. It increases the obsessive computation from conducting both proposal generation and bounding box regression, where the computational cost increases further with the number of proposals generated (Hyams and Malowany, 2020).

Another class of approaches for object localization is the single-shot approach. Instead of generating multiple proposals in the first stage, single-shot approaches directly jump to the later stage of object localization. 3D bounding boxes regression is carried out by constructing a network that regresses 3D bounding boxes without further refinement process. Since the post-processing is eliminated, the computational load and cost are reduced. Single-shot approaches are further separated into three main classes depending on the form of input data: bird's eye view (BEV) based approaches, discretization-based approaches, point-based approaches, and graph-based approaches (Guo, et al., 2020). Single-shot approaches are rapid processing

since the computational load relies on the number of anchors but not the number of objects in 3D space. However, it loses part of the accuracy. Single-shot approaches generally lack the refinement process for the object candidates. It removes most of the background instances, and thus more inaccurate, false-positive objects can be detected.

The accuracy of the localized target object is evaluated by average precision (AP). It integrates recall and precision, ranging between 0 and 1, as a measure for ranked restoration results (Zhang and Zhang, 2009). The recall is a measurement of the positive results obtained, while precision is the measurement of the actual results obtained based on recall (Hui, 2018). A high AP indicates most of the localized object is accurate compared to the ground truth data. AP is obtained by the mean of precision scores, where the area under the precision-recall curve is calculated. AP is a suitable measurement parameter to evaluate a method for object localization, where it relates the predicted bounding boxes with the ground truth labels.

The dataset applied by the approaches consists of the KITTI test BEV detection benchmark, and the AP result to be discussed is based on the easy class for the cars' category. The KITTI dataset consists of LiDAR point clouds embedded with the feature of reflection intensity. The BEV detection benchmark is preferred over the 3D object detection benchmark in the literature works' discussion because some approaches only utilize the BEV form of input rather than a raw point cloud. BEV is a compact type of point cloud representation. It is accomplished by transforming the raw point clouds into the BEV form to render it the 2D characteristics and increases the point cloud's density. The AP predicted using the BEV detection benchmark does scale on the AP predicted using the 3D object detection benchmark for every difficulty for the same approach; hence, the BEV input's results are relatable with the results using raw 3D point cloud input. The threshold for intersection-over-union (IoU) is set at 0.7 during the regression of 3D bounding boxes for all approaches. IoU is the measure of area overlapped between ground truth and prediction to classify the positive results as true or false. The IoU value greater than the threshold will be classified as true positive (Hui, 2018).

## 2.1 Region Proposal-based Approaches

Two-stage object localization frameworks create multiple proposals to define the regions which potentially contain the interesting object. The proposals could be in the form of 3D bounding boxes that act as preliminary results of object localization. The 3D bounding boxes are then refined to improve the result of accuracy and precision.

### 2.1.1 Frustum-based Approaches

PointNet is one of the pioneering works of point cloud for 3D classification and segmentation, proposed by Qi, et al. (2017) to resolve the irregularity of point cloud in the processing technique. They proposed a unified architecture that acquires raw point cloud as data for broad applications of classification, part segmentation, and semantic segmentation. It transforms features through a series of multilayer perceptron (MLP) layers and rectified linear unit (ReLU) non-linearity, and finally aggregate point-level features by a max-pooling operation. Later, Qi, et al. (2017) proposed an advanced version of PointNet, PointNet++, to improve the previous version. PointNet++ aggregates both global and local features, whereby it is accomplished by continuous sampling and grouping of points between two PointNet's feature aggregation layers. A sampling and grouping operation plus a PointNet operation forms a set abstraction layer. The abstraction layers are then applied by many object localization frameworks afterwards. However, PointNet and PointNet++ are unable to detect target objects and localize them.

To resolve the challenges of PointNet on how to efficiently propose potential locations of interesting 3D objects in 3D space like point cloud, Qi, et al. (2018) proposed frustum-based PointNets (F-PointNets), which PointNet inspires. It leverages existing and matured 2D object detectors, CNN, to generate 2D region proposals as candidates in the red, green, and blue (RGB) images by classifying the content. Each of the 2D region proposals in the RGB image is extruded from the centre of view to a 3D frustum. This is accomplished by integrating the 2D region proposals with depth information with the point cloud and form 3D frustum proposals. The 3D frustum proposals that potentially contain the 3D target objects are extracted and trimmed. After that, the points within the 3D frustum proposal are fed into two

variants of PointNet, namely 3D instance segmentation and amodal 3D box estimation. 3D instance segmentation is used to acquire the object's locations in the point cloud with the given 2D image space and the relevant 3D frustum structure. This predicts the probability for each point in the frustum belonged to the object by binary classification. The segmented object point cloud is then aligned by translation and light-weight regression of PointNet to predict the object's actual centre to match with the amodal box centre. Lastly, amodal 3D box estimation will predict and locates bounding boxes for objects with the segmented and aligned point cloud.

This approach is not a type of end-to-end learning to regress bounding boxes, where it heavily relies on other functional refinement modules to perform the tasks. Besides, the number of available foreground points in the 3D frustum is minimal, and false-positive points on 2D images could generate false 3D frustum proposals, thus producing inaccurate results with low precision. Since there are two refinements of the 3D frustum proposal network in this approach, the computational cost increases. The extra load of 3D instance segmentation is performed and requires several works to segment the object. The result of this approach is favourable, where it achieves 91.17 % AP for 'easy' cars when using the KITTI test BEV detection benchmark. Figure 2.1 shows the F-PointNet's architecture.

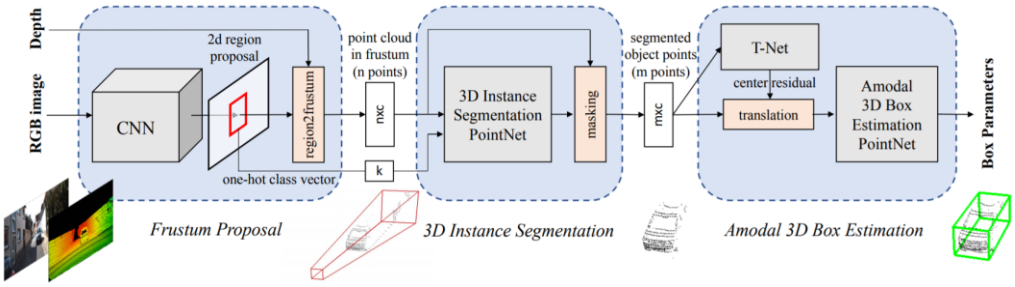



Figure 2.1: F-PointNet's Architecture (Qi, et al., 2018).

Frustum-based convolutional neural network (F-ConvNet) is another approach of frustum-based object localization proposed by Wang and Jia (2019) using frustum proposal generation architecture. Like F-PointNet, 2D region proposals are generated from RGB image and map into the given depth to produce proposals in 3D space. The 2D region proposals then extrude to

produce the 3D frustum proposals. Instead of one 3D frustum proposal generated from a 2D region proposal, F-ConvNet generates an array of frustum throughout the length of the frustum axis for each of the 2D region proposals, where the frustum axis is perpendicular to the RGB image plane. The 3D frustum proposals are produced by segmenting the parallel frustum planes offset from the image plane throughout the frustum axis's length with similar spacing and strides. Then, one 3D frustum is produced from a couple of parallel planes. Each of the 3D frustum proposals generated is then applied with PointNet to group the local points within the frustum to extract the point-level features to generate a frustum-level feature vector. PointNet groups the points by feeding the points into an MLP series and performing max pooling to aggregates the point features. The frustum-level features are transformed to become a 2D feature space, and this will become suitable for a fully convolutional network (FCN) to work on the frustum-level features. In the FCN, the features are consecutively abstracted to reduce the feature map's size and eventually deconvoluted later to perform 3D bounding box regression. Since more 3D frustum proposals are processed from one 2D region proposal, the computational load is heavier than F-PointNet. However, the result shows a minor improvement compared to F-PointNet, with 91.51 % for 'easy' cars' AP when tested with the KITTI test BEV detection benchmark. Figure 2.2 shows the F-ConvNet's architecture.

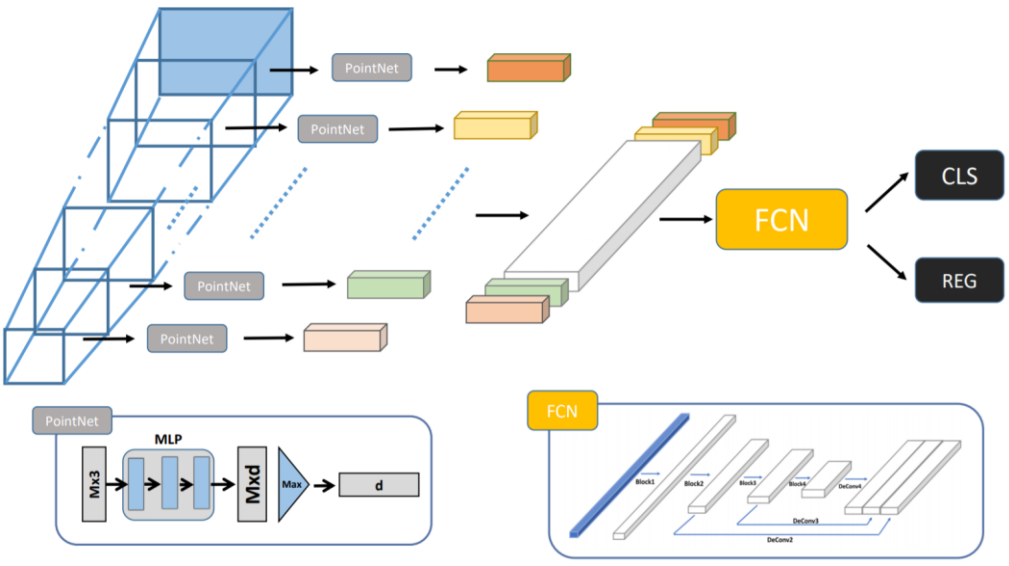



Figure 2.2: F-ConvNet's Architecture (Wang and Jia, 2019).

Both F-PointNet and F-ConvNet exploit the mature 2D detector in the early stage to construct the backbone of the architecture. This brings up the problem of heavily relying on the 2D detection's performance, and it does not take advantage of the rich 3D information in point clouds. 2D images have lesser information describing the objects than 3D point clouds. A false 2D region proposal generated at the beginning stage of the detection will carry forward the error to the framework's subsequent modules and produce inaccurate results. These approaches generally score well among the 2D image-based approaches detection but are not classified as 3D point cloud-based.

### 2.1.2 Multi-view-based Approaches

Multi-view-based approaches exploit the highest number of input modalities for processing among the 3D object localization techniques. Instead of taking the RGB image or point cloud as the primary detector's input, these approaches fuse the features extracted from both RGB images and point clouds to regress 3D bounding boxes. A multi-view 3D object localization framework (MV3D) proposed by Chen, et al. (2017) functions as a sensory-fusion network that takes multiple data modalities as input and leverages the multimodal information to execute region-based feature fusion. The first subnetwork, the 3D proposal network, takes advantage of the point cloud's regular-grid BEV representation as to the primary detector. It undergoes 2D FCN, and the point cloud is being discretized into a 2D grid before performing 3D bounding boxes regression that generates 3D proposals. BEV is chosen among other point cloud representations due to its conservation of physical sizes of objects that do not vary much with the camera's distance and are less likely to be occupied by obstacles when viewed on top. The 3D proposals generated are projected to each of the three views of BEV point cloud, front-view point cloud, and RGB image after each of them being fed into FCN and feature-abstracted from their respective unique representations. The projection will concatenate the extra information of depth to other modalities for better delivery of contextual information. After that, in the second subnetwork, the region-based fusion network, a fusion of the combined proposals and extracted features for each input modality is then conducted. The multiple views'

features are effectively fused and fed into a deep fusion network. It extracts the features from each late fusion's intermediate layer to perform feature concatenation with an element-wise mean operation for its high flexibility when integrated with drop-path training. The output feature of the region-based fusion network is used to regress 3D bounding boxes.

Since the fusion of features of multiple views through a region of interest (RoI) pooling occurs at a course level, it produces a significant loss of resolution and geometric information during the quantization, and all of the information directed into the region-based fusion network is depending on the fused's output. The resulting AP for 'easy' cars is only 86.62 % when using the KITTI test BEV detection benchmark, which is considered low compared with other approaches. Besides, the computational load is high, where each of the input modalities has to be discretized into a 2D grid and feed into FCN individually. A deep region-based fusion network also increases the computational load, where the tasks are sequential and cannot be accomplished in parallel computing. Thus, it requires improvement in effectively fuse the different modalities and extract reliable input data representations. Figure 2.3 shows the MV3D's architecture.

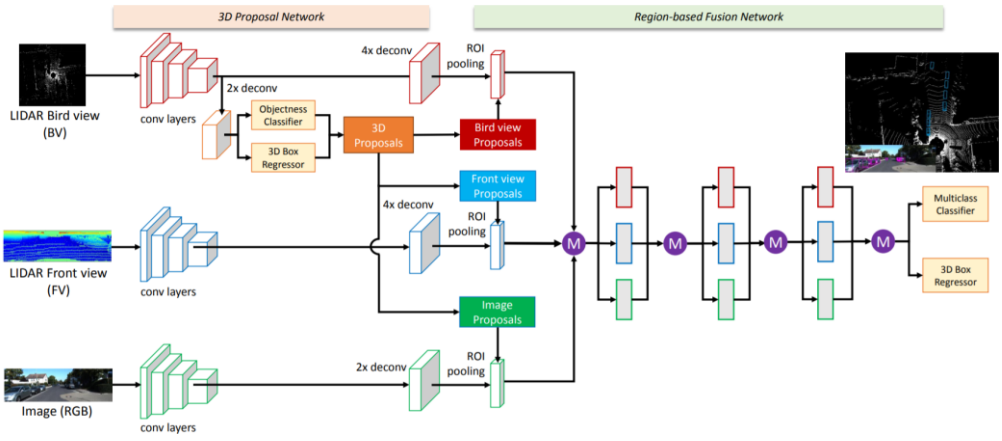



Figure 2.3: MV3D's Architecture (Chen, et al., 2017).

Therefore, Liang, et al. (2018) proposed a deep continuous fusion framework (ContFuse) for multiple modality 3D object localization. Like MV3D, ContFuse exploits both BEV point cloud and RGB image to predict 3D bounding boxes. Nevertheless, the way of ContFuse generates 3D proposals is different. Instead of projecting proposals generated from the BEV

point cloud to other modalities, ContFuse projects proposals generated from the RGB images into the BEV point cloud. This is because it is challenging to acquire 3D detections from the projected 2D outputs. A convolutional network extracts the features from the RGB image before projecting them to the BEV point cloud. It is then fused with the convolutional layers of 3D based detector. To increase the effectiveness, continuous convolution is applied to extract features from the nearest relevant image for every point in the BEV point cloud for every resolution, and bilinear interpolation is implemented to create BEV feature vectors. The BEV feature vector has a higher density than an ordinarily BEV point cloud, which resolves the drawbacks of the discretized image's feature loss. Continuous convolution allows connections of multiple intermediate layers of resolutions on both image and BEV point cloud streams and performs multi-scale fusion for multiple sensors. The output detection produced is in BEV space, which is more reliable for 3D bounding boxes regression. This is because the detection header is created from the output BEV feature map instead of discrete image space and sparse point cloud feature maps.

ContFuse has region-based fusion integrated into the proposal projection network. It saves many computational loads from performing 3D bounding box regression with the elimination of deep fusion layers. The prediction accuracy is also increased and achieved 94.07 % AP for 'easy' cars when using the KITTI test BEV detection benchmark. The geometric relationship between multiple modalities is densely encoded during the continuous convolution layers at different resolutions. However, the fusion of data is still limited by LiDAR BEV's sparsity, which is the native of the representation, and producing smaller object recall when the object's size decreases. The number of positive results for small objects will be lesser. Figure 2.4 shows the ContFuse's architecture.

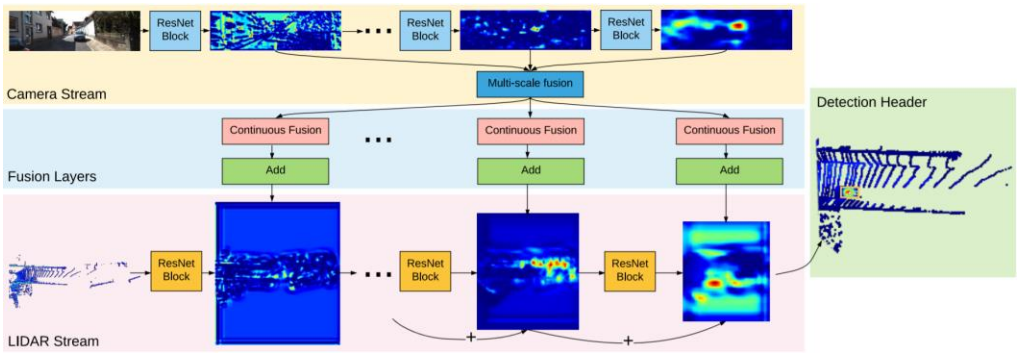



Figure 2.4: ContFuse's Architecture (Liang, et al., 2018).

The smaller object recall usually limits multi-view-based approaches. The quality of the 3D proposals depends on the consistency of features abstracted from the 2D FCN for each modal. It can be affected by the scene's complexity and highly occluded and crowded objects, where the information might not be sufficient and consistent to generate proposals on them confidently. This is because the noise presence could dominate the features in the 2D RGB image.

### 2.1.3 Segmentation-based Approaches

These approaches exploit semantic segmentation mechanisms like 3D instance segmentation on point cloud or 2D segmentation on image to remove most background points, leaving essential foreground points in the data. These essential points are the interest of object localization and are used to generate superior proposals, which is better than ordinary proposals in terms of quality and accuracy. The proposals are generated on the lower density points, where the computational load can be reduced. Shi, Wang and Li (2019) proposed a region-based convolutional neural network for 3D object localization (PointRCNN), which segments 3D point clouds during the framework's first stage. This is to acquire essential foreground points in 3D space using a point segmentation network with PointNet++'s backbone, while point cloud is the only input modality the approach used. The points in 3D space are relatively sparsed from each other, where overlapping or occlusion rarely occurs, rendering effective segmentation to extract valuable foreground points and features. The semantic masks are provided from the training data itself, namely 3D bounding box annotations, where it specifies which object of

points in 3D space is required to be segmented. In the first stage of the framework, 3D bounding box proposals are generated in a bottom-up manner. It utilizes the segmented foreground points and generates 3D proposals for each foreground point in parallel computation. This will reduce the computational load, where lesser proposals are generated due to the removal of background points as noise and thus increasing the proposals' quality. Whereas in the second stage of the framework, the proposals are being refined in canonical coordinates. The proposals generated in the previous stage are feature-abstracted and transformed into canonical coordinates. The transformed local spatial features will fuse with semantic features from the point-level feature vector of the segmented point cloud and the segmentation mask by using the grouping approach in PointNet++. This will effectively refine the 3D bounding boxes for further increase in quality with a higher confidence level.

Canonical transformation offers the advantage of a higher recall from 3D proposals. It also allows better learning of local spatial features for each proposal in the box refinement stage. It is combined with each point's global semantic feature in the previous stage, and the merged features are used for 3D bounding box refinement and confidence prediction. Thus, segmentation-based approaches generally have a higher recall over most region proposal-based approaches, where they can accurately localize smaller objects. However, the geometric information could be lost due to the imperfect point cloud region RoI pooling scheme for different resolutions. The AP of PointRCNN is affirmative of 92.13 % for 'easy' cars when using the KITTI test BEV detection benchmark while having the computational load that is almost evenly distributed into the two stages of the framework. Figure 2.5 shows the PointRCNN's architecture.

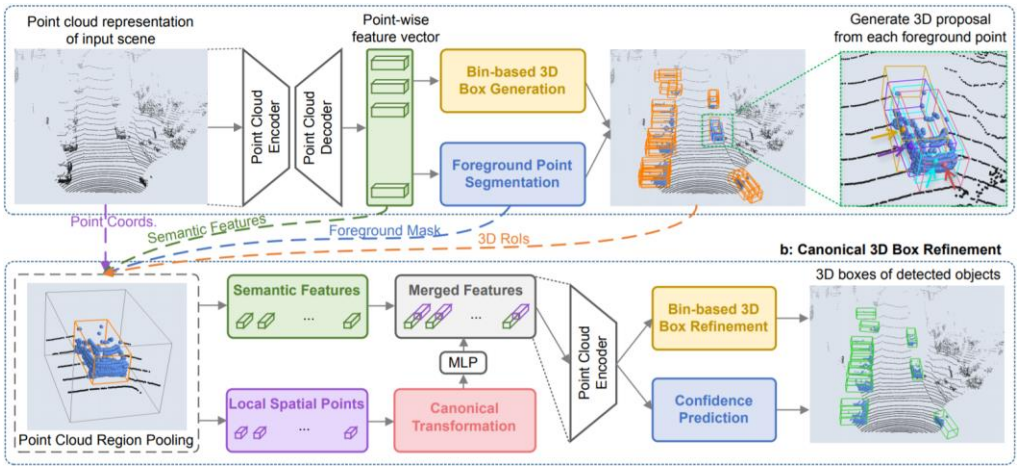



Figure 2.5: PointRCNN's Architecture (Shi, Wang and Li, 2019).

Zarzar, Giancola and Ghanem (2019) proposed a region-based graph convolutional network for 3D object localization (PointRGCN) that possesses certain similarities with PointRCNN but differ in the box refinement stage. PointRGCN also works exclusively on point cloud input modal only. The initial stage of high recall 3D proposal generation is identical to PointRCNN. It leverages the segmentation approach in PointNet to extract essential foreground points by removing most of the background points and generates proposals based on them. When performing 3D box refinement, PointRGCN utilizes a graph convolutional network (GCN) to perform proposal feature and context aggregation instead. It consists of two modules, namely residual GCN (R-GCN) and contextual GCN (C-GCN). The first module, R-GCN, utilizes the point clouds' geometric information by constructing a graph representation for points within each proposal to extract features for classification and 3D bounding box regression. The point cloud is first transformed into canonical coordinates. The canonical coordinate of each point is then projected into a larger space, and it concatenates with the corresponding 3D proposal features to obtain a per-point feature vector. R-GCN processes the vectors to obtain global point features by projecting the vector layers into a higher dimensional space and then max-pooled. The second module, C-GCN, then aggregates the contextual information among multiple proposals in the frame. Both global and local information within each proposal are being encoded as nodes in a global frame graph representation. Global features are computed by concatenating the local feature vector from one proposal to another. The global

features are then used to regress 3D bounding boxes and perform confidence prediction.

PointRGCN exploits a two-stage 3D object localization approach with two graphs neural network. The performance surprisingly dropped when compared to PointRCNN, where it achieves 91.63 % AP when tested for ‘easy’ cars using the KITTI test BEV detection benchmark. Apart from that, it contains two GCNs in the second stage of 3D box refinement that severely increases the system’s computational load, where both R-GCN and C-GCN leverage a graph neural network (GNN) individually that extensively aggregates the information across nodes via edges. The size of graph representations of the proposal point cloud in R-GCN and the number of nodes in C-GCN is highly dependent on the number of 3D proposals generated in the initial stage of the region proposal network (RPN). Many proposals could be generated as it is built on every segmented foreground point, yet high quality with high recall. However, the number of epoch of GCN can be limited to prevent a heavier computational load. Figure 2.6 shows the PointRGCN’s architecture.

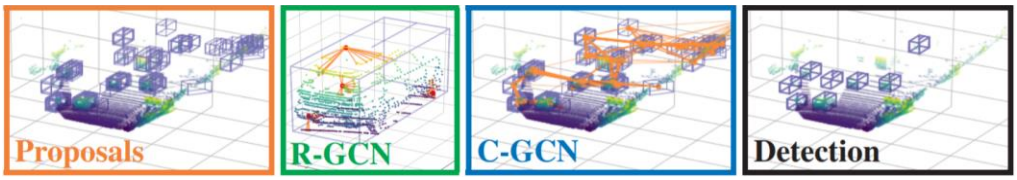



Figure 2.6: PointRGCN’s Architecture (Zarzar, Giancola and Ghanem, 2019).

### 2.1.4 Summary

Region proposal-based approaches generally have a more significant computational load since they involve two stages in the object localization pipeline. A significant number of 3D bounding boxes are generated in the first stage that acts as proposals to the object localization, suggesting the regions with a higher possibility of containing the interesting objects.

The proposal generation can be separated into two approaches. The grid-based approach encodes the irregular point clouds to a regular representation, such as 2D BEV maps. This includes the multi-view-based and frustum-based approaches, where it allows matured 2D object detectors of

CNN or FCN to work on it effectively. The second proposal generation approach would be a point-based method that directly works on the irregular point cloud without transformation or feature engineering, with the example of segmentation-based approaches. Working on a raw point cloud will enhance the computational load, where the computer deals with irregular data inefficiently and has to be supported by certain architecture backbones. However, point-based approaches can utilize point set abstraction to achieve a larger receptive field. The object recall is higher and retains more significant geometric information that increases the proposals' accuracy with a trade-off of increased computational load. Conversely, the grid-based approaches are computationally efficient. However, they could end with inevitable information loss due to the inappropriate use of 2D detection techniques and loss of information from different resolutions in the convolution layers.

The proposals generated by region proposal-based approaches are refined to reduce the number of bounding boxes and improve the accuracy by reducing false-positive results. The box refinement methods leveraged vary greatly, where it has no fixed pattern or approach of architecture to be applied. PointNet and PointNet++ proposed feature abstraction approaches that preserve accurate geometric information with flexible receptive fields. It allows implementation in most of the box refinement approach to effectively extract feature-level vectors from 3D proposals to regress 3D bounding boxes and perform confidence prediction.

Among all the region proposal-based approaches discussed, the segmentation-based method, PointRCNN, outperforms all other approaches and is most suitable for object localization. This is because the computational load is less concerned in this project, while the accuracy is the primary focus. PointRCNN has a higher recall, and the ability to retain geometric information preserves the features well and generates a higher quality of proposals in the first stage of the framework. Besides, the canonical 3D box refinement submodule takes multiple point cloud coordinates, semantic features, foreground mask, and 3D RoIs to merge the local spatial points and semantic features effectively.

## 2.2 Single-shot Approaches

The single-stage object localization frameworks omit the process of generating a massive number of preliminary proposals in the RPN while still accurately detecting target objects. This reduces the computational load created from the proposal refinement stage. Single-shot approaches directly regress the 3D bounding boxes as final detection. Hence, the approach of preprocessing the input modal and the feature abstraction method will be crucial to the pipeline's performance.

### 2.2.1 BEV-based Approaches

As expressed with the class's name, these approaches utilize the unique characteristics of the BEV representation of the point cloud to regress 3D bounding boxes. Yang, Lup, and Urtasun (2018) have proposed oriented 3D object localization from pixel-level neural network predictions (PIXOR). It projects the raw point cloud into a 2D feature map by discretization and compute the pixel values of the projected point cloud and treating it as a 2D vector. The resulting projected point cloud is known as the BEV representation of the point cloud. The 2D feature map is suitable to leverage FCN to extract the feature-level vectors from the BEV point cloud to perform object localization by regressing 3D bounding boxes. Geometric information could be lost during discretization, where it can no longer exploit the natural characteristic of the raw point cloud. Instead, it is being transformed into a compact representation, where the point cloud becomes denser with the depth information being embedded into the 2D feature maps. This approach is too simple to be implemented in 3D object localization, where it has the minimum feature engineering to abstract features for 3D bounding boxes regression. Besides, the 2D framework working on an originally 3D point cloud without optimum processing does induce inaccuracy because the complexity increases when changing from 2D to 3D. Thus, PIXOR's result is 83.97 % AP for 'easy' cars when tested with the KITTI test BEV detection benchmark. However, this approach has a smaller computational load than complex architecture in region proposal-based approaches. Figure 2.7 shows the PIXOR's architecture.

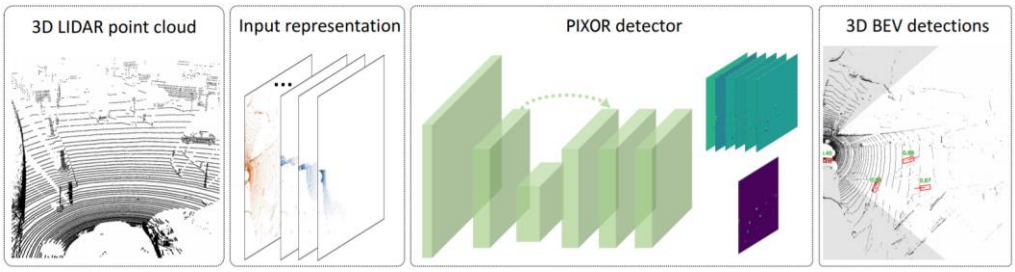



Figure 2.7: PIXOR's Architecture (Yang, Lup and Urtasun, 2018).

3D object localization framework from BEV LiDAR (BirdNet) proposed by Beltrán, et al. (2018) is another class of BEV-based object localization approach. It tries to reduce the information loss during point cloud projection or BEV formation by presenting the BEV map's normalization. Limiting the height at each cell and computing the average intensity of all points within the cell will produce homogeneous pixel values among all cells. This produces the capability to encode the maximum number of points within a cell, where the normalized BEV map has a denser and more compact representation. Therefore, the computation effectiveness is increased. Each cell contains the highest available number of points to allow the feature extractor to acquire more output from cellular level operation. The normalization map has a similar resolution as the 2D BEV map. Besides, the implementation of normalization is reliable when importing data from different LiDAR sensors, where the number of points collected varies with the parameters built-in. The normalized map is fed into 2D CNN to extract features, and regression of 3D bounding boxes is carried out using RPN. Homogeneity in pixel values will eliminate the objects' unique characteristics. Thus, the BEV point cloud representation's density increases as well as the dissimilarity between the objects in the point cloud versus the objects in the BEV map. The resulting AP is 76.88 % for 'easy' cars when tested with the KITTI test BEV detection benchmark. Figure 2.8 shows the BirdNet's architecture.

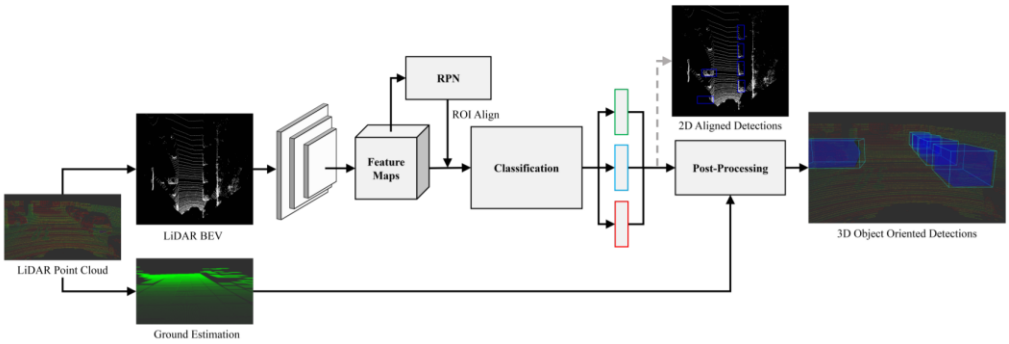



Figure 2.8: BirdNet's Architecture (Beltran, et al., 2018).

The BEV-based single-stage detector is exploiting the BEV representation of the point cloud to regress 3D bounding boxes. This type of approach is limited to an extent. It does not support other input modality types, not even utilizing the raw point cloud effectively. Although BEV is a typical representation of point cloud for 3D object localization, there is not much computation that can be worked on to improve the results' overall accuracy. This is because the BEV point cloud is overloaded with information, while some are lost during the BEV transformation. The high density of the BEV point cloud makes it less flexible to be further feature-engineered.

### 2.2.2 Discretization-based Approaches

Discretization-based object localization is similar to BEV-based approaches, where it also encodes the point cloud into a regular grid. Instead of encoding the point cloud into a 2D grid to become BEV representation, discretization-based approaches encode the points into a 3D voxel grid to become discrete representation. These approaches also allow CNN to be applied to the voxelized point cloud's regular representation to abstract features. However, some empty cells may exist, where it contains no points as a result of the point cloud's sparsity. The computation worked on the empty cells will be accounted for nothing but just increasing the computational load. Engelcke, et al. (2017) proposed 3D object localization with a feature-centric voting scheme (Vote3Deep) to effectively localize objects in 3D point clouds. It utilizes the feature-centric voting mechanism to take advantage of the point cloud's sparsity. Each input feature vector with several points in the cell will cast votes to its neighbouring cells, and the accumulation of votes in each cell

will become the output of the voting mechanism. The process is known as sparse convolution, where only specific cells are being computed to vote and feature-abstracted. This process is repeated multiple times, and it appears like a convolutional layer, which is the basic building block for CNN. The process is stacked convolutionally to extract features from layers to layers. Then, the point cloud's sparsity is preserved by exploiting the ReLU non-linearity after each convolutional layer. This is to prevent non-empty cells from being dilated during the convolution. This approach is highly dependent on the number of cells containing points, where it will be less efficient if the overall point density in the cells is low. The computation cost will increase as the scene become greater without modifying the size of the cells. There is no available accuracy result for this approach using the KITTI test BEV detection benchmark. Figure 2.9 shows the Vote3Deep's architecture.

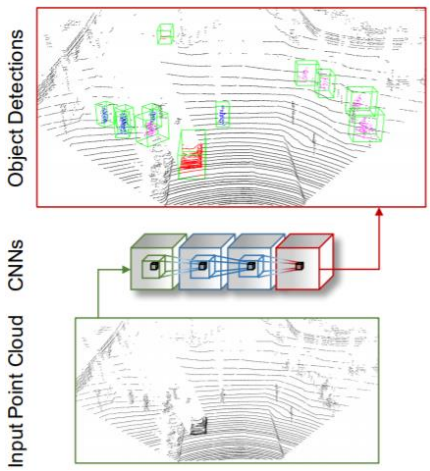



Figure 2.9: Vote3Deep's Architecture (Engelcke, et al., 2017).

Zhou and Tuzel (2017) presented a voxel-based object localization framework (VoxelNet). It utilizes RPN to generate 3D bounding boxes; however, it is limited by the sparse point cloud's low density. Hence, they intended to produce a denser representation of the point cloud known as the sparse four-dimensional (4D) tensor. The point cloud is encoded in a 3D voxel grid initially. The voxel feature encoding (VFE) layer is then designed to allow interaction of points within the voxel to aggregates point-level features. The VFE layers appear in a stacked circumstance, where it enables aggregation of complex local features before interacting with other voxels.

This will produce a descriptive volumetric representation for the point cloud. It creates a new dimensional feature into the 3D voxel to increase the complexity of the voxelized point cloud representation. Initially, each voxel is fed into the stacked VFE layers to obtain a point-level feature. After that, FCN is applied to the point-level features to aggregate the voxel-level features. The voxel-level features are being concatenated into the respective voxel in the grid, producing sparse 4D tensors. Middle convolutional layers further refine the sparse 4D tensors to extract spatial contexts among the voxels for information transfer before regressing 3D bounding boxes via RPN. This approach results in an AP of 89.35 % for 'easy' cars when using the KITTI test BEV detection benchmark. Due to the sparse voxel's convolution in the beginning and 3D convolution on the sparse 4D tensor at the end, the computational load is enormous. Figure 2.10 shows the VoxelNet's architecture.

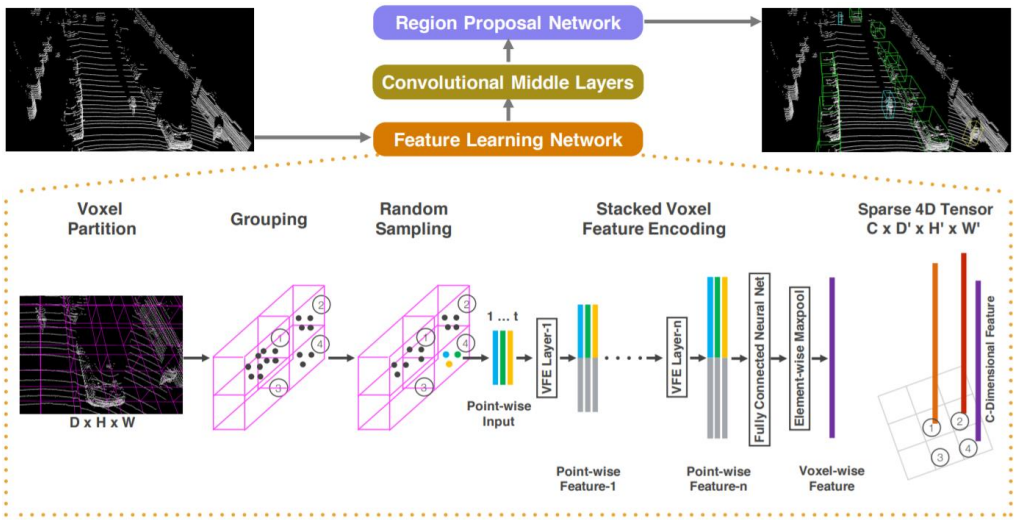



Figure 2.10: VoxelNet's Architecture (Zhou and Tuzel, 2017).

Discretization-based approaches are generally inclusive in terms of architecture design. It opens the area for the point cloud's post-processing after it is being discretized into a 3D voxel grid. The computational load mainly depends on the design of the feature abstraction from the voxelized point cloud. It can be as efficient as Vote3Deep to perform computation on point-present voxel only or be as complex as VoxelNet to work on every voxel and further increase the computation by constructing sparse 4D tensors and abstract features from it.

### 2.2.3 Point-based Approaches

This approach explicitly takes raw point cloud into 3D bounding boxes regression for 3D object localization without performing any feature engineering to convert the point cloud into other denser representations like BEV or voxelized point cloud. Yang, et al. (2020) proposed a point-level 3D single-stage object localizer (3DSSD). It fuses the feature distance-based furthest point sampling (F-FPS) with 3D Euclidean distance-based furthest point sampling (D-FPS) in the set abstraction layers. These set abstraction layers will serve as the network's backbone to prevent point loss after sampling. It eliminates pure feature propagation layers for the upsampling of points and refinement modules in ordinary point-based object localization architecture high in computational load. A candidate generation layer is then concatenated after the set abstraction layers to regress 3D bounding boxes. The sampled points from F-FPS are used to generate candidate points with geometrical coordinates, and shifting operation is performed with features abstracted from F-FPS. This is to provide supervision to the geometrical location between the points. As the F-FPS points are treated as the centres, F-FPS and D-FPS's neighbouring points are detected as a whole set of representations. MLP networks then abstract the features from the points to regress 3D bounding boxes using an anchor-free head regression approach. The feature propagation layers are adapted with fusion sampling, where the computational load is optimized while having a similar accuracy of the detection result. Since the foreground points' information is retained by implementing fusion sampling, the result's AP is generally as high as 92.66 % for 'easy' cars when using the KITTI test BEV detection benchmark. Figure 2.11 shows the 3DSSD's architecture.

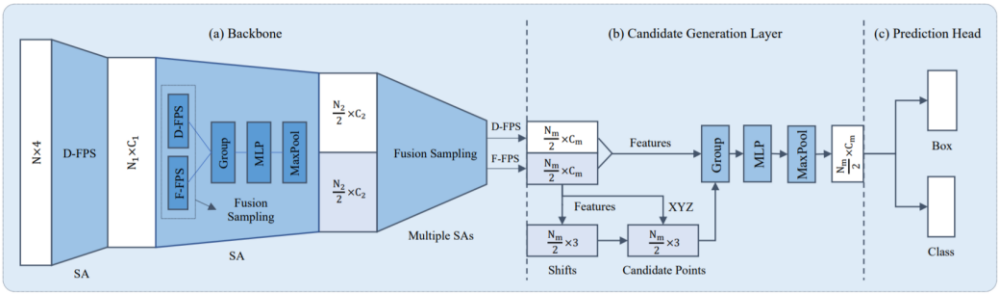



Figure 2.11: 3DSSD's Architecture (Yang, et al., 2020).

### 2.2.4 Graph-based Approaches

The graph-based approach leverages the graph representation of the point cloud for the object localization framework. Shi and Rajkumar (2020) proposed a GNN for 3D object localization in point clouds (Point-GNN). A GNN transforms a point cloud into graphs before abstracting its features. It transforms a group of the points in the point cloud into a graph vertex, where each vertex contains the information of both 3D coordinates and the feature state encoded. The point cloud is voxel downsampled during vertices' graph construction to reduce the computational burden. The density of the point cloud is reduced to a fair amount for efficient vertex formation and computation. The vertices within a fixed neighbouring radius are joined together by graph edges. The fixed neighbouring radius is a parameter to adjust the trade-off between computational load and accuracy. The larger the radius, the more edges will be connected to vertices in the graph. This will allow more information transfer to a vertex that is far from the origin vertex and increase the receptive field. Therefore, more computation has to be accomplished to refine the features during the iteration process.

The edges are used as the bridge connection between two vertices to transfer information in a bidirectional way, where it contains the features of both vertices. The vertex's feature is refined during each iteration by aggregating the edges' features, while the edges are computing the features from the two vertices it linked. The features of vertices and edges are aggregated continuously throughout the iterations. The information of one vertex obtained from its neighbouring vertex is being transferred to another neighbouring vertex with a longer relative distance with the first vertex via edges. The series connection of vertices allows transferring information from a smaller fixed starting radius to the entire graph after several iterations. The receptive field of vertices is therefore increased. Features used for 3D object localization are aggregated as vertices' features. The vertex's state feature is refined by its neighbouring vertex's state through the edge. Spatial information between vertices is exchanged to increase the sense of belonging in the point cloud's graph representation. The final iterated vertices' features are used to regress 3D bounding boxes.

Continuous grouping of points in convolutional layers is eliminated in Point-GNN to reduce the computational cost. Instead, it is only grouped and sampled once at the beginning of graph construction. The overall performance of the Point-GNN is good, where it adapts the characteristic of a point cloud of sparsity just like the point-based approach does. It is not altering it by projecting the point cloud into a regular representation that may cause data loss. The computational load can be adjusted via parameters of fixed neighbouring radius and number of iterations, with the trade-off of the accuracy of average precision. Point-GNN achieved an AP of 93.11 % for 'easy' cars when using the KITTI test BEV detection benchmark under normal circumstances. Figure 2.12 show the Point-GNN's architecture.

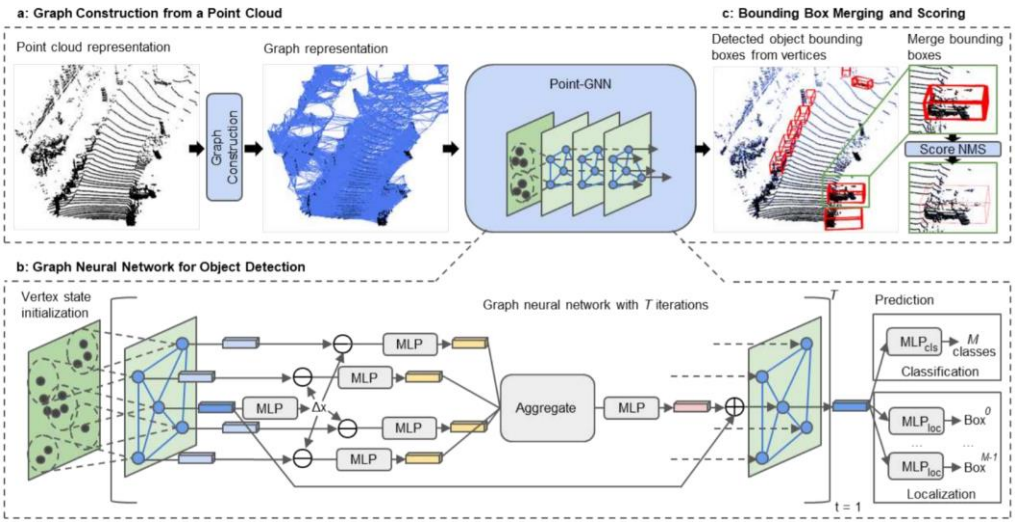



Figure 2.12: Point-GNN's Architecture (Shi and Rajkumar, 2020).

### 2.2.5 Summary

Single-shot approaches are generally having a smaller computational load than region proposal-based approaches. It has a more straightforward architecture design, where there is no massive number of proposal generations and complex proposal refinement modules that saturate the object localization's computation. While having a more straightforward architecture design, the framework's accuracy does not compromise, where some of the single-shot approaches are having AP higher than the region proposal-based approach.

The BEV-based approach transforms the raw point cloud into BEV representation to match the existing and matured 2D detector's requirement to

abstract feature from it. Although it leverages the 3D information of the point cloud to an extent, utilizing the 2D representation is limited since the BEV feature map is compact and highly dense, limiting further complex processing to aggregate features. Next, the discretization-based approach transforms the irregular point cloud into a 3D regular representation of the voxelized point cloud. The point cloud partitioning will allow precise computation to work on it if the empty voxels are neglected. Many types of computation can be designed to handle voxels, whether contained with points.

Furthermore, the point-based approach takes the raw point cloud as the input for feature abstraction to preserve the raw data's complete information. The AP obtained from this approach is generally high, and the optimization applied to the point-based approach is mainly to reduce the computational load. Lastly, the graph-based approach takes graph representation of point cloud as input to perform feature abstraction. Since this approach is somehow similar to the point-based approach, it has relatively high AP as results, but computational load remains a concern to apply real-time monitoring.

Overall, Point-GNN from the graph-based approach has the best architecture design among the single-shot approaches. This is because Point-GNN preserves the point cloud's sparsity to prevent information loss from feature engineering. The constructed graphs have a structural similarity with the raw point cloud to utilize its sparse characteristic. The vertex representation of a small group of points forms bondings with each other via edges to transfer information and local features. Graph representation preserves most of the information points without complex pre and post-processing features. Along with high AP, simplicity is achieved as well, where it has no complex mechanism.

## 2.3 Overall Summary

In conclusion, single-shot approaches are more preferred than region proposal-based approaches, where it has a higher simplicity of architecture design for 3D object localization. Although it is only one-stage detection, single-shot approaches do not compromise its accuracy, and it achieves a better result than some of the region proposal-based approaches. Accuracy is mainly depending on the computational load, not the type of approach.

Therefore, Point-GNN is chosen among all object localization frameworks. It best suits the sparse characteristic of a point cloud, where it is inherited to the structure of the graphs constructed in the first place. The graph representation of the point cloud appears like the raw point cloud, where the points are being connected with their neighbouring points via linkages. The connection of vertices via edges in Point-GNN allows information transfer between vertices to aggregate local features in the beginning and global features at the end of the iterations. Finally, the comparison of accuracy among all approaches discussed above is listed in Table 2.1 with their respective input modalities.

Table 2.1: Comparative 3D Object Localization Results of Various Approaches on the KITTI Test BEV Detection Benchmark, with an IoU Threshold of 0.7 for 3D Bounding Box (Guo, et al., 2020).

| Approaches | Modality | Average Precision for Cars | | |
|---|---|---|---|---|
| | | Easy (%) | Moderate (%) | Hard (%) |
| F-PointNet | LiDAR, Image | 91.17 | 84.67 | 74.77 |
| F-ConvNet | LiDAR, Image | 91.51 | 85.84 | 76.11 |
| MV3D | LiDAR, Image | 86.62 | 78.93 | 69.80 |
| ContFuse | LiDAR, Image | 94.07 | 85.35 | 75.88 |
| PointRCNN | LiDAR | 92.13 | 87.39 | 82.72 |
| PointRGCN | LiDAR | 91.63 | 87.49 | 80.73 |
| PIXOR | LiDAR | 83.97 | 80.01 | 74.31 |
| BirdNet | LiDAR | 76.88 | 51.51 | 50.27 |
| Vote3Deep | LiDAR | - | - | - |
| VoxelNet | LiDAR | 89.35 | 79.26 | 77.39 |
| 3DSSD | LiDAR | 92.66 | 89.02 | 85.86 |
| Point-GNN | LiDAR | 93.11 | 89.17 | 83.90 |

# CHAPTER 3

# METHODOLOGY AND WORK PLAN

## 3.1 Introduction

In this chapter, the methodology of the project is discussed. All the necessary details to accomplish the project, repeat it, reproduce the results following the objectives are stated. Besides, the planning and managing of the project activities, which constitute for a year since the beginning of the project, will be illustrated with two Gantt charts with explanations. Furthermore, the problems encountered during the project's execution will be discussed with the solutions carried out.

## 3.2 Environment Setup

This section describes the development environment used for this project.

### 3.2.1 Python Programming

This project was completed using Python, an open-source programming language that contains a lot of backup support and higher-quality documentation. This is because the project consists of deep learning. Python has an extensive selection of open-source libraries and frameworks for deep learning projects, such as Keras, PyTorch, and TensorFlow (Protasiewicz, 2018). This promotes coding capability, especially in this deep learning project that consists of heavy programming activity. Besides, the Python programming language's simplicity allows the code writer and reader to understand the code concisely. Thus, a programmer can invest more time in complex problem-solving instead of cognitive overhead on syntax errors. The testing of the algorithm can be performed without implementation into hardware components. Anaconda Spyder is chosen as the programming platform due to its functionality in coding and ease in installing packages.

### 3.2.2 TensorFlow Library

Since object localization (part of object detection) is the primary concern in this project, the TensorFlow library is chosen over Keras and PyTorch. This is

due to its high-quality object detection application programming interface (API), facilitating the construction, training, and deployment of object localization models (Keshari, 2020). Furthermore, it works well with a large dataset like the KITTI dataset, which consists of about 7 500 point cloud scenes for both training and testing of the model. TensorFlow's performance is generally high, where it has both high and low-level APIs for deep learning solutions. However, TensorFlow is harder to debug, and its readability is lower when compared to Keras due to the higher complexity of the architecture in TensorFlow (Sayantini, 2020).

### 3.2.3 The KITTI Dataset

The KITTI dataset is used for researching point cloud processing techniques for autonomous vehicles (Geiger, et al., 2013). It consists LiDAR point cloud that represents the scenes on the road in Karlsruhe, Germany. The main objects present in the scene are cars, pedestrians, and cyclists. This project's focus is cars where it has the most significant dimensions, thus having better accuracy results in 3D object localization. The other objects in the scene are less meaningful and significant than the main objects due to the lesser ground truth data. Each point cloud scene is accompanied by its respective RGB image and the object's labels for training various object localization approaches. It is optimal for use in this project due to the comprehensive implementation in the literature works above. It has moderate and sufficient difficulty to be applied in practical object localization solutions.

The KITTI dataset's files arrangement is suggested to follow the file structure shown in Figure 3.1. This produces ease in executing Point-GNN, the selected 3D object localization approach, to extract data from the same structure. 'Image', 'velodyne', and 'calib' in Figure 3.1 represent the dataset of RGB image of the scene in the left direction, a point cloud of the scene, and the calibration data for internal data conversion during processing. Each of the image, velodyne, and calib dataset consists of both training and testing data, where it will be applied when executing model training and inferencing, respectively. Label dataset only consists of ground truth labels for the training dataset, which also can be used for AP prediction of the model during evaluation. 3DOP_splits is the validation split for the dataset. It lists the file

indexes that contain particular interesting objects, such as cars and pedestrians, and cyclists, as mentioned in the text file's name.

```
DATASET_ROOT_DIR
├── image                    #  Left color images
│   ├── training
│   │   └── image_2
│   └── testing
│       └── image_2
├── velodyne                 # Velodyne point cloud files
│   ├── training
│   │   └── velodyne
│   └── testing
│       └── velodyne
├── calib                    # Calibration files
│   ├── training
│   │   └──calib
│   └── testing
│       └── calib
├── labels                   # Training labels
│   └── training
│       └── label_2
└── 3DOP_splits              # split files.
    ├── train.txt
    ├── train_car.txt
    └── ...
```

Figure 3.1: Recommended Dataset File Structure (Shi and Rajkumar, 2020).

### 3.2.4 Executing Terminal and Version of Libraries

This project was executed on two platforms: Anaconda terminal (Windows) and UTAR University research server (Linux). The primary Point-GNN execution is accomplished using the UTAR research server as well as the data collection and evaluation, while the Anaconda terminal is used to generate graphical results of Point-GNN. This is because both platforms have their limitation where some of the actions cannot perform well, and the limitation is resolved when using another platform. The detailed discussion is included in the problem-solving section of this chapter.

This project uses the TensorFlow version of 1.15. Besides, it requires five python packages to be installed, which are OpenCV-python, open3d-python version of 0.7.0.0, scikit-learn, tqdm, and shapely. There are two principal codes for Point-GNN's execution, namely run.py and train.py. Run.py is used for inferencing the data, in order words, produce the output from the input dataset using the pre-trained Point-GNN model. The output of

Point-GNN consists of 3D coordinates and the three primary dimensions of the objects, which origin at the camera point. By obtaining the result of Point-GNN, the second and third objective of this project will be accomplished. Train.py is the Point-GNN's training algorithm, where it allows retrain of the Point-GNN model to produce different results during inferencing. The AP prediction can be performed using kitti_native_evaluation, which KITTI sources to evaluate the models using the KITTI training dataset and the ground truth labels. AP is the main parameter for evaluating this project's approaches, where all approaches discussed in the previous chapter are using the KITTI dataset for consistency.

## 3.3 Point-GNN Development

This section describes the flow of Point-GNN in inferencing the output.

### 3.3.1 Graph Construction

Point-GNN is chosen as the main approach for object localization after reviewing multiple literary works. It has multiple steps to perform object localization and only utilizes the LiDAR point cloud as the input modality. After the point cloud data is read, the graph representation of point cloud is constructed by converting the point cloud of $N$ points into a set $P$, where each subset of $P$ is a point with both 3D coordinates $x_i$ and the state value $s_i$, hence $p_i = (x_i, s_i)$ and $P$ is representing a set of vertices. The state value is represented by the reflection intensity of the LiDAR point cloud. Then, a graph, $G = (P, E)$ is constructed with vertices and the fixed neighbouring radius $r$, where $E$ is the edges of the graph:

$$E = \left\{ (p_i, p_j) \;\middle|\; \|x_i - x_j\|_2 < r \right\} \text{——(1)}$$

The vertices' state features are initialized with embedding LiDAR reflection intensity and relative coordinates using MLPs, which are then aggregated by the max-pooling function. After that, the vertices' state features are refined by aggregating features along the graph edges. In the $(t+1)^{th}$

iteration, the vertex feature $v^t$ and edge feature $e^t$ in a general graph are updated in the form of:

$$\begin{matrix} {v_i}^{t+1} = g^t\left( p\left(\left\{e_{ij}^t \,\middle|\, (i,j) \in E\right\}\right), v_i^t \right) \\ e_{ij}^t = f^t\left(v_i^t, v_j^t\right) \end{matrix} \quad --(2)$$

where $g^t$ is a function to aggregate edges' features to refine a vertex's feature while $f^t$ is a function to compute the edge's feature with two vertices the edge is connected. $p^t$ is a set function that aggregates the edges' feature for each vertex.

### 3.3.2 Auto Registration for Offset Coordinates

To specialize for object localization approach, equation 2 is rewritten to involve the neighbour's state during the refinement of vertex's state to include belonging information of the vertex:

$${s_i}^{t+1} = g^t\left( p\left(\left\{f^t\left(x_j - x_i, s_j^t\right) \,\middle|\, (i,j) \in E\right\}\right), s_i^t \right) --(3)$$

where $x_j - x_i$ represents the relative coordinates of the neighbours. The relative coordinates approach will induce translational invariance against the point cloud's global shift, where the global shift is a type of data augmentation to prevent overfitting the model. However, it remains sensitive in the local neighbourhood area. The relative coordinates of its neighbours will be altered when introducing a small translation and causes an increase in translational variance, which is a disadvantage to the training function $f^t$. An auto-registration mechanism is proposed to reduce the translational variance. An alignment offset can be predicted by utilizing centre vertex coordinates and aligning the neighbours' coordinates by their structural features. Therefore, the equations are rewritten as:

$$\begin{matrix} {\Delta x_i}^t = h^t(s_i^t) \\ {s_i}^{t+1} = g^t\left(p\left(\left\{f^t\left(x_j - x_i + \Delta x_i^t, s_j^t\right) \,\middle|\, (i,j) \in E\right\}\right), s_i^t\right) \end{matrix} --(4)$$

where $\Delta x_i^t$ is the coordination offset for the coordinate registration of vertices and $h^t$ is used to compute the offset using the centre vertex's features from the previous iteration. In overall, $f^t$, $g^t$, and $h^t$ are modelled using MLPs, and max-pooling is used for vertices' state aggregation:

$$\begin{aligned} {\Delta x_i}^t &= {MLP_h}^t(s_i^t) \\ {e_{ij}}^t &= {MLP_f}^t\left(x_j - x_i + {\Delta x_i}^t, s_j^t\right) \\ {s_i}^{t+1} &= {MLP_g}^t\left(Max\left(\left\{{e_{ij}}^t \middle| (i,j) \in E\right\}\right), s_i^t\right) \end{aligned} \quad --(5)$$

### 3.3.3 Bounding Boxes Prediction

After $t$ iterations, the final vertices' state values are used to predict 3D bounding boxes using the localization branch of $MLP_{loc}$. The bounding box is predicted for each vertex and is determined by seven parameters, thus possessing seven degree-of-freedom: $b = (h, w, l, x, y, z, \theta)$ . $(x, y, z)$ represents the 3D coordinates of the 3D bounding box, while $(h, w, l)$ represents the height, width, and length of the bounding box predicted, respectively, and $\theta$ is the yaw angle rotates around the Y-axis, which is the axis of the height in point clouds. By utilizing $(x, y, z)$, the target objects will be localized in a 3D point cloud by referring to the location of the bounding box that surrounds the object. Besides, the primary dimensions of the target object can be measured by taking $(h, w, l)$, where the assumption is made on the target object fully enclosed by the bounding box. This is accomplished by increasing the bounding box's size 10 % larger to prevent cutting the object's edges. To measure the object's dimension accurately in terms of alignment, yaw angle $\theta$ will align the bounding box with the object's centre axis to perform enhanced 3D object localization and primary dimensions measurement.

### 3.3.4 Bounding Boxes Regression

Some large objects might have multiple vertices present, and a bounding box is predicted for each vertex that belonged to the same object. Box merging operation is conducted to regress the bounding boxes into one. The traditional non-maximum suppression (NMS) approach that determines the box with the highest classification score and expels the others does not improve the

localization accuracy. Instead, it merely reduces the number of boxes. In order to improve the localization quality, the median position and size of the whole bounding box cluster are computed, along with the confidence score as the summation of classification score weighted by the IoU and occlusion factors:

$$i = argmax(D)$$
$$L_i = \sum_{b_{j=0}}^{B} IoU(b_i, b_j) > T_h \quad --(6)$$

where $D$ is a set of detection scores, $B$ is a set of bounding boxes, and $T_h$ is the overlapping threshold. IoU threshold of 0.7 is set rather than 0.5 to increase the result's confidence level by reducing the precision value. For each successive mapping of IoU by the given threshold value, $D$ and $B$ is removed with the corresponding index, and this process will repeat until the bounding boxes are emptied, $length(B) = 0$. Equation 6 outputs a set of the bounding boxes, which fulfiled the threshold requirement. Regression of the bounding box is carried out by the following equation:

$$m_i = median(L_i)$$
$$o_i = occlusion(m_i) = \frac{1}{h_i w_i l_i} \prod_{v \in \{v_i^h, v_i^w, v_i^l\}} \max_{p_j \epsilon b_i}(v^T x_j) - \min_{p_j \epsilon b_i}(v^T x_j) \quad --(7)$$
$$z_i = (o_i + 1) \sum_{b_k \epsilon L_i} IoU\,(m_i, b_k) d_k$$

where $m_i$ is the selected bounding box, $o_i$ is the occlusion factor computed, and $z_i$ is the corresponding bounding box's confidence score. After the bounding boxes are regressed, output data is generated for the localized objects in the point cloud. Figure 3.2 shows the overall flow of the processes in the Point-GNN inferencing algorithm.

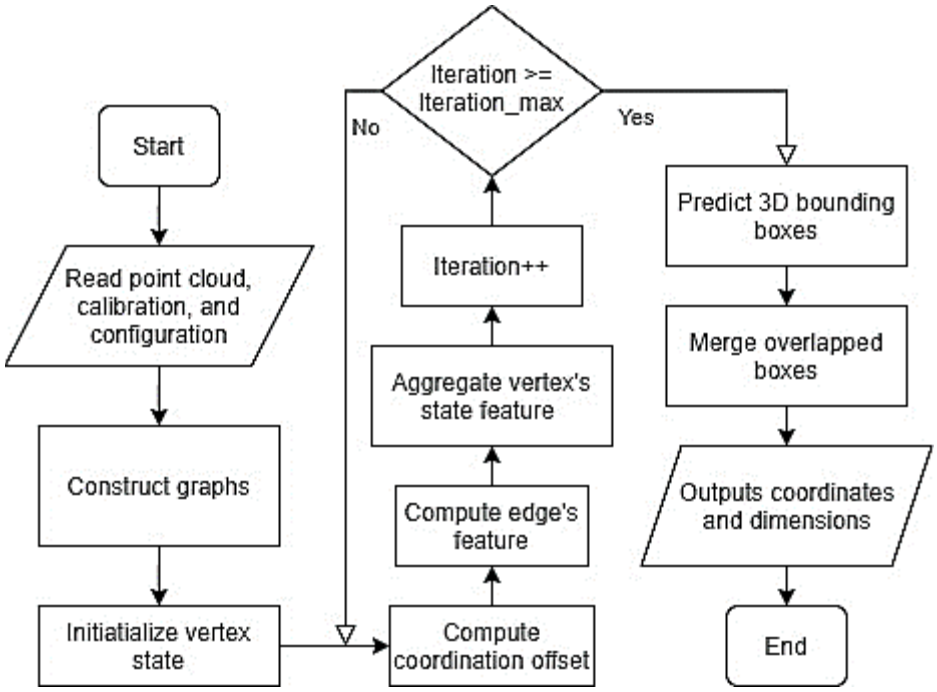



Figure 3.2: Flowchart of Point-GNN Inferencing Algorithm.

### 3.4 Adjustment Made to Point-GNN

During the Point-GNN's execution, it was found the algorithm only focuses on the objects within the range of the RGB image taken, which is the region in front of the camera. Point-GNN did this way because the 'label' dataset only consists of the objects in front of the camera, and AP prediction only evaluates the localized object in the front point cloud. This is not a good practice in implementing 3D object localization in an autonomous car. The car's control system should consider all aspects around it, not limited to the front of the camera but also behind it. For instance, an autonomous car shall notice the vehicles behind it before switching lanes on the pavement. Therefore, the code is rewritten by disabling the `front_cam_points` tuple, consisting of points and attributes for the points that have the Z3D of greater than 0.1. The points with Z3D greater than 0.1 indicating the points lie in front of the image. The subsequent functions with the input of `front_cam_points` tuple are also substituted with the pre-processed `cam_points` tuple, which consists of all points in the point cloud. However, this creates syntax error in the later algorithm that converts the camera points to the image plane via calibration, where the algorithm divides the Z3D after determining the interesting points:

```
kitti_dataset.py:1050: RuntimeWarnig: invalid value encountered in
true_divide: img_points_xy1 = img_points_xyz/img_points_xyz[:,[2]]
```

where `img_points_xyz[:,[2]]` represent the Z3D of the `img_points` tuple. `cam_points` tuple does consist of points that have a Z3D value of zero. This will cause the algorithm's imperfect execution. Infinity value is obtained, and data loss occurred due to data invalidation. Therefore, the code is further modified to include the points of having Z3D greater than 0.1 or lesser than $-0.1$ to prevent invalid data:

$$p = p_i \, \epsilon \, (Z3D > 0.1 \, || \, Z3D < -0.1) - -(8)$$

This change in coding will allow all points in the point cloud to be processed without introducing significant data loss from excluding points between $-0.1$ to 0.1 in the Z-axis. With the introduction of points behind the camera, the computational cost increased. The time required to generate the output becomes longer due to the increased number of vertices and edges.

### 3.5 Performing Intra and Inter Approach Comparisons

The Point-GNN approach is compared with the different number of layers in the Point-GNN model to verify the difference in AP predicted and the computation cost tradeoff. The number of layers in the Point-GNN model represents the number of iteration for graph vertices in aggregating features. An inference to the intra-comparison is that the higher the number of iteration, the more information is gained for each vertex from its neighbourhood, and result in higher AP. Thus, all Point-GNN results used for intra-comparison will be executed on the same platform, the UTAR research server. The number of iterations tested is 0, 1, 2, and 3, and the evaluation for each output data is accomplished by the KITTI native offline evaluation metrics, with the recall position of 11. Evaluation is conducted by generating the output data using the training dataset to compute the recall and precision based on the ground truth data.

Apart from that, the inter-approach comparison will be conducted to compare Point-GNN with other approaches discussed in the literary works in

chapter 2. The main and the only parameter used for discussion is also the evaluated AP of the approach. The AP for the approaches discussed is obtained from the KITTI Vision Benchmark Suite. All of the approaches were submitted to KITTI by their authors, and KITTI outputs the APs for three difficulties for each object class the approach is working. Therefore, the KITTI reported APs are based on the same evaluation metrics and use the recall position of 40.

## 3.6 Planning and Managing of Project Activities

This section describes the planning of the project activities with time management, where time is the only resource in this project.

### 3.6.1 Project Part I

Project part I consisted of four major activities to be accomplished by the end of the project part. In the first two weeks, a discussion with the supervisor was conducted to plan for this project regarding the project's output, project duration, possible problems encountered, and the design for the Gantt chart shown in Table 3.1. After that, comprehensive literature reviews were conducted to cover many object localization works relevant to the 3D point cloud as input modal. The sources were analyzed and criticized professionally by reviewing them in-depth and discussing all necessary aspects to highlight the current gaps in the 3D object localization research. The literature review was started right after registration of this project title in the first week, and eight weeks were spent to accomplish it. There were twelve 3D object localization approaches collected and reviewed, where region proposal-based approaches and single-shot approaches both consisted of six works of literature.

Then, a research methodology was conducted to set up the preferred programming environment. Python was chosen over C++ due to its simplicity and readability of codes that promote ease in programming. Point-GNN was to be executed; however, it failed due to the different operating systems in the computer used in accomplishing this project versus the Point-GNN's written code. The KITTI test dataset was obtained, consisting of 7 500 point cloud scenes with the respective RGB image and label for both the training and

testing dataset. This was to increase the flexibility of input modalities to be used in this project. This activity was started in the third week, overlapped 87.5 % with the literature review activity, and was ended in the twelfth week. The second last part of this project was the writing of the progress report. The literature review section, which is the heaviest part of the progress report, was first to be completed for the supervisor's inspection in terms of the report's quality. It was arranged in a coherent structure to expand the reader's more profound understanding of 3D object localization in point clouds. After that, the introduction was written to identify the current problem and the aim and objectives. The methodology section then described the experimental details, where the facilities to be used were elaborated and justified and is ended with the project planning. Progress report writing was started in the ninth week, and six weeks were spent completing it. The project part I was concluded with the presentation to the supervisor and the moderator, which held in the fourteenth week. In conclusion, all activities were accomplished as planned. Table 3.1 shows the Gantt chart for project part I.

Table 3.1: Gantt Chart for Project Part I.

| No. | Activities \ Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M1 | Project planning | ✔ | ✔ | | | | | | | | | | | | |
| M2 | Literature review | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | | | | |
| M3 | Research methodology | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| M4 | Report writing & oral presentation | | | | | | | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

### 3.6.2 Project Part II

Project part II consisted of six activities to be accomplished by the end of this project part. In the first five weeks, the Point-GNN's code was executed using the UTAR research server. The research server consisted of remote access that allows the user to execute the algorithm using the research server's specifications, which are more robust in performance than an ordinary computer. Numerical results of Point-GNN were generated using the research server, while the graphical results of Point-GNN were generated using a student's computer. Next, research methodology was carried out from the second week onwards, and have spent nine weeks researching the possible improvements to fill up the technological gaps in Point-GNN. It was found out that Point-GNN is limiting itself to localize the objects within the range of the RGB image taken, which is in front of the camera while leaving others objects in the point cloud scene behind and unlocalized. Changes were made to the code to improve the output result in terms of point's coverage. This project's model development was carried out along with the research methodology to resolve the error on programming, where coding was performed in a Python environment. Another set of inference results was generated with the new code. The KITTI dataset with 7 481 training samples and 7 518 testing samples of point cloud was used as the project's input modal. Model development was started in the third week and ended in the twelfth week.

Result and discussion were performed in the middle of the model development, which was the eighth week of project part II. The results obtained from the improvement of the method and the original Point-GNN were analyzed and discussed. Besides, a poster regarding this project was prepared, which began in the ninth week. Both result and discussion and poster preparation were ended in the twelfth week due to the poster's submission. The result and discussion were essential for the poster to promote the model developed for 3D object localization in point clouds. Lastly, the final report writing was started in the seventh week, two weeks earlier than the progress report writing in project part I, to improve writing skills and early problem spots. This was to improve the report's quality by including essential information in the report to effectively expand the reader's knowledge. This activity was continued until the submission of the final report in the fourteenth

week. The final presentation was prepared and conducted to present this project's results to both the supervisor and the moderator. Table 3.2 shows the Gantt chart for project part II.

Table 3.2: Gantt Chart for Project Part II.

| No. | Activities / Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M1 | Execution of Point-GNN's code | ✔ | ✔ | ✔ | ✔ | ✔ | | | | | | | | | |
| M2 | Research methodology | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | | | |
| M3 | Model development | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| M4 | Result and discussion | | | | | | | | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| M5 | Poster preparation | | | | | | | | | ✔ | ✔ | ✔ | ✔ | | |
| M6 | Report writing & oral presentation | | | | | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

## 3.7 Problems Encountered and Solutions

During the project's execution, three problems have been encountered, and most of them have been resolved and achieved a better improvement in the project execution.

### 3.7.1 The Computational Expense of Executing Point-GNN

The execution of Point-GNN was decided to run on the student's computer, which is the model of Acer Nitro 5 with the specifications of 12 gigabytes (GB) of random access memory (RAM). When the code was executed in the Anaconda terminal, which is the initial primary code execution terminal, it outputs nothing but mentions the insufficient available device memory and causes zero output of Point-GNN result. The condition was not solved even after optimizing and defragmenting the disk. The number of iterations of Point-GNN executed was three: the maximum iteration number and the default value suggested by Shi and Rajkumar (2020). This is because the higher the number of iterations, the farther the vertex's information could reach in the neighbourhood. One possible inference to the inability of execution is the inappropriate hardware execution. Point-GNN is generally running on CPU, and Python programming itself is also increasing the computation cost, where it is a high-level programming language with higher complexity. It is estimated that the minimum requirement to execute Point-GNN is 20 GB of RAM in the computer. This problem was resolved by changing the computation hardware by shifting the task of executing Point-GNN from a student's computer to the UTAR research server, which has a bigger RAM of 64 GB that runs on Linux. Numerical results of Point-GNN were outputted in the research server and stored in text files, including the 3D coordinates and the three primary dimensions of the localized objects. In addition, the research server allows remote access from the student's computer where the student's computer can be used to perform other helpful project activities other than code execution. The comparison between the specifications of the student's computer and the UTAR research server is listed in Table 3.3.

Table 3.3: Comparison Between Specifications of the Student's Computer and the Research Server.

| | Student's computer | UTAR research server |
|---|---|---|
| Operating system | Windows | Linux |
| RAM (GB) | 12 | 64 |
| GPU | NVIDIA GeForce GTX 1050 | NVIDIA GP104GL |
| On-screen (FPS) | 60.0 | 13.0 |

### 3.7.2 Shifting Primary Programming Environment to Linux

During the shifting of the primary programming environment from the student's computer to the UTAR research server, problems were encountered during the environment setup. Since the execution of Point-GNN has been shifted onto the UTAR research server, the KITTI dataset has to be placed inside the server for Point-GNN's data acquisition. The KITTI dataset was obtained during project part I, which is stored in the student's computer. A Linux command called 'rsync' allows synchronizing files across the local server to the remote server. However, this approach is slow due to the requirement of prolonged connection between the servers to transfer the files. Hence, the 'wget' command was used to download the files directly online, which speed up the dataset set up in the research server. However, some files are not available online directly, such as the folder of 3DOP_splits; thus, the 'rsync' command was applied to gather them into the research server before executing Point-GNN's code.

The disk quota kept saturated during the dataset set up in the research server due to limited volume. This is because the KITTI dataset is enormous, where it has 7 481 training samples and 7 518 testing samples. The entire dataset consumes a size on disk of 38.6 GB, and it contains 52 486 files in total. Therefore, the research server's user-allocated disk space was updated to input the whole dataset into the research server. Most of the dataset folder such as 'image', 'velodyne', 'calib', and 'labels' are setup using the 'wget' command; The 'wget' command outputs a '.zip' folder for each dataset into the research server and requires further unzip operation. The unzipping operation of Linux is outputting an extracted folder that consumes the same amount of disk space to the research server before the zipped folder can be

removed. However, the disk space was saturated again upon the completion of the dataset download. Due to the disk space's saturation in the research server, the unzipping operation could not be done. A solution to this was removing some zipped dataset folders before unzipping other zipped folders that remained. This solution allocates space for the extracted files to occupy and, thus, allows the unzipping process. The zipped-extracted folders were then removed to allocate disk space for further unzipping processes. At the end of the dataset setup, the files in the initially removed zipped folder were input to the research server via the 'rsync' command, where no zip file or 'wget' command is involved. 'Image' zipped folder was removed in this aspect due to its medium-sized occupied disk space; therefore, the 'rsync' command is not consuming a very high computation cost in importing files from the 'image' dataset at the end.

### 3.7.3 The Inability of Research Server to Visualize Point-GNN Results

Point-GNN's code has an argument that allows the visualization of the results. However, the UTAR research server faced difficulty displaying graphical results, which contain both image and point cloud with bounding boxes enclosing the objects. The system core has dumped frequently and causes the operation aborted. Introducing graphical application while executing Point-GNN has caused the several numerical results produced before the abortion worse than the results produced without displaying the graphics. One possible reason may be that the GPU of the research server of NVIDIA GP104GL has difficulty displaying such graphical results. It has low on-screen frames per second (FPS) displayed and causes interruption on the code execution. Thus, the graphical result was then produced using the Anaconda terminal on the student's computer. The GPU of the student's computer of NVIDIA GeForce GTX 1050 has a higher on-screen FPS displayed as shown in Table 3.3, and it successfully displayed the graphical results.

However, due to the student's computer's limitation, Point-GNN can only be executed in zero iteration in the Anaconda terminal. This produces the results that the vertex's state information is not shared among its neighbourhood. Therefore, the validated result's AP is the lowest compared to other results with the number of iterations greater than zero. The low AP is

due to the introduction of false-positive results. Point-GNN can barely localize the true-positive objects only when the objects are fully visible in the point cloud and the scene's complexity is low for a non-iterated model. In other words, the objects are not occluded by any other less meaningful objects, such as trees or walls, during the point cloud acquisition. Although Point-GNN's visualization is limited to a zero iterated model, the numerical results' usability is far more superb than the graphical results. This is because the numerical data can be used as input to an autonomous car's control system to carry out decisive actions. Simultaneously, the graphical result is limited to human use for visually evaluating the Point-GNN.

### 3.7.4 Missing System Package

The KITTI offline evaluation metrics is encoded in a '.cpp' file and other directories. It requires compilation to make it executable for intra and inter-approach evaluation. However, the UTAR research server lacks the 'cmake' system package with the minimum requirement of version 2.6 to compile the files. Therefore, a request is sent to the supervisor to install the system package into the research server environment for Point-GNN evaluation. The KITTI offline evaluation metrics are compiled, and it is executed with the inputs of validation result from Point-GNN using the training dataset and the ground truth label dataset. The KITTI offline evaluation metrics outputs a series of evaluation data with the car object: 2D detection AP, 2D orientation AOS, BEV detection AP, BEV orientation AHS, 3D detection AP, and 3D orientation AHS. Each evaluation data is accompanied by a graph displaying the data. The focuses of evaluation data in this project are 3D detection AP and BEV detection AP.

## 3.8 Summary

The environment setup of the project is explained. A Python programming platform is required to modify and execute segments of Point-GNN's code. Besides, the TensorFlow library is required for the deep learning programming sessions in the code. The KITTI dataset is obtained and is successfully loaded into Point-GNN to generate output. Point-GNN's development is explained in details with numerical equations included and the overall flowchart. Then, the

adjustment made to Point-GNN is described and analyzed in this chapter, too. Also, the method of performing intra and inter-approach comparison is discussed. The planning and management of the project's activities are explained, where it is separated into part I and part II for two long semesters that constitute a year in total. Problems encountered and the solutions selected are also included in this chapter, followed by the results of the solutions executed.

# CHAPTER 4

# RESULTS AND DISCUSSION

## 4.1 Introduction

This chapter describes the result obtained from Point-GNN execution and the code's adjustment made. Intra and inter-approaches comparison will be discussed and evaluated further in this section.

## 4.2 Point-GNN

The KITTI files input to the code consist of the velodyne point cloud, image, label, and calibration. The collection of the KITTI dataset has marked the accomplishment of the first objective in this project. These four datasets are sufficient to reproduce the Point-GNN both numerical and graphical results. The image and calibration files are used to produce graphical results, where the points will map to the image plane and camera plane through and forth during the processing. The 'label' dataset consists of the ground-truth label of the objects, and it is used to predict AP from the output results generated from the training samples. The primary dataset used by Point-GNN processing is the velodyne point cloud, which consists of X, Y, and Z coordinates as well as the LiDAR reflection intensity. The LiDAR reflection intensity represents the vertex's state value during the vertex initialization. Next, the calibration data is also used to convert the points from velodyne coordinate to camera coordinate, where there is a change in the X, Y, and Z coordinates of the points to produce the generalized points regardless of the camera's setting. The configuration file is also input to Point-GNN, which consists of the parameters to execute the code. The parameters include the keyword arguments for graph generation and model construction. The graph generation keyword argument consists of the configurations to generate graphs, such as the graph type and the radius of neighbour searching for generating edges' graph. Then, the model keyword argument consists of the Point-GNN layer type for each layer in the Point-GNN's iteration. The MLP depth parameters are also included in the layer configuration, as well as the normalization and activation type for the MLPs.

After executing the Point-GNN, it outputs both the numerical and graphical results. The graphical result output is controlled by an argument, '--level' during the code's execution to allow visualization of the result. Figure 4.1 shows the graphical result of the Point-GNN for the testing sample 000001. The graphical result consists of an RGB image of the scene with two green 2D bounding boxes covered on the two cars localized. The graphical result also consists of a 3D point cloud of the scene with three red 3D bounding boxes covered on three cars localized. The coloured lines intersecting at the bounding box represent the graph's edges, where each edge is connected to two vertices in the graph. The visible lines in the point cloud are the edges with one of the vertex lying within the 3D bounding box. Therefore, the vertex at the other end of the edges is delivering its state's value to the bounding box's vertex for aggregating local features. The green lines represent the object localized is classified as 'car', where cars are the target object for this project. Meanwhile, the grey lines represent the object localized is classified as 'do not care'. The 'do not care' class is implemented to prevent false-positive results in the evaluation, where it is not included in the localization result. Therefore, there are only two target objects, cars, localized in the testing sample 000001.

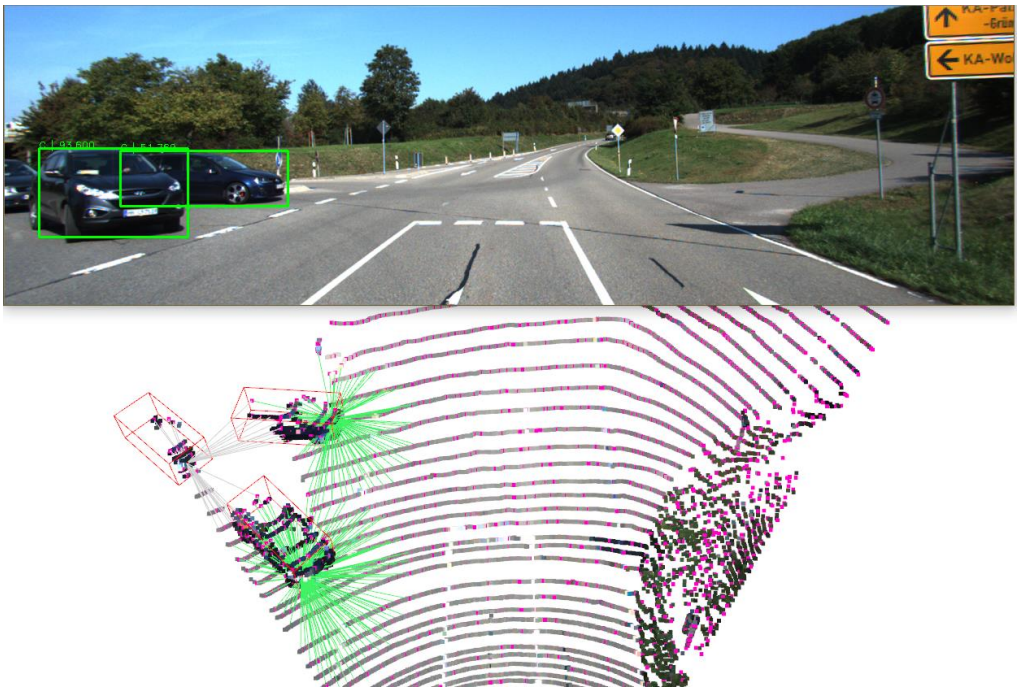


Figure 4.1: Graphical Result for the Testing Sample 000001.

Apart from the graphical result, Point-GNN also outputs the numerical result simultaneously. The first three data: truncation, occlusion, and alpha, are not the interesting parameters in this project, where it refers to 2D image processing. The clip for $X_{min}, Y_{min}, X_{max},$ and $Y_{max}$ represent the maximum limits for the particular object in the 2D RGB image. Similarly, these four parameters are not interested in the project's discussion due to the involvement of the 2D region. The 'label' KITTI dataset included all these parameters because there are approaches utilizing the 2D detector in localizing objects. The significant numerical result parameters are the X3D, Y3D, Z3D, height, width, and length. The X3D, Y3D, and Z3D lie at the bounding box's origin, computed in the MLP of the localization branch. The bounding box's origin lies on the top XZ bounding box plane's centre point. This indicates the bounding box's origin lies in the top centre of an object to reference the object from the global origin, the LiDAR camera's coordinate at $(0,0,0)$. Hence, this project's second objective is achieved by determining the X, Y, and Z coordinates of the located object in a 3D point cloud.

The height, width, and length represent the three primary dimensions of the object in meters, which are also computed in MLP of the localization branch. The height, width, and length are displaying in the form of bounding box lines. Each bounding box line is drawn across two of the eight corners surrounding the object to form a cube. The bounding box corners are positioned by the height, width, and length parameters computed, originating (X3D, Y3D, Z3D). The upper four corners are positioned at $\left(\pm {}^{w}/_{2}, 0, \pm {}^{l}/_{2}\right)$, while the bottom four corners are positioned at $\left(\pm {}^{w}/_{2}, -h, \pm {}^{l}/_{2}\right)$, in respect to (X3D, Y3D, Z3D). The bounding box's orientation is determined by the second last parameter in the numerical result, yaw angle, representing the rotation angle of the object around the Y-axis, which is the axis of the height. Thus, this project's third objective is accomplished by displaying the height, width, and length of the located object in both numerical and graphical form. The last parameter in the numerical result is the confidence score computed for the bounding box regressed to indicate performance for the particular bounding box predicted. Table 4.1 shows the numerical result for the testing sample 000001.

Table 4.1: Numerical Result for the Testing Sample 000001.

| **Object class** | Car (1) | Car (2) |
|---|---|---|
| **Truncation** | -1 | -1 |
| **Occlusion** | -1 | -1 |
| **Alpha** | 0 | 0 |
| **Clip_Xmin** | 141.20734621538205 | 36.41744711056066 |
| **Clip_Ymin** | 176.2229789093736 | 174.03262110345074 |
| **Clip_Xmax** | 351.68137221248566 | 232.33337647692284 |
| **Clip_Ymax** | 243.90982579877945 | 289.8086621292917 |
| **Height (m)** | 1.5215358 | 1.5745287 |
| **Width (m)** | 1.6863366 | 1.6601882 |
| **Length (m)** | 4.283223 | 4.113779 |
| **X3D** | -8.645287 | -7.8946705 |
| **Y3D** | 1.6074694 | 1.5971005 |
| **Z3D** | 17.363804 | 11.835676 |
| **Yaw** | 0.09412821 | 0.72221255 |
| **Score** | 82.2164862876902 | 151.07463956706272 |

### 4.2.1 Adjustment Made to Point-GNN

Based on the changes made in section 3.4, the point cloud input into the Point-GNN model is different, where it also consists of the points behind the camera. Figure 4.2 shows the point cloud scene of the testing sample 000001 viewed from the top that consists of the front and the rear point cloud.

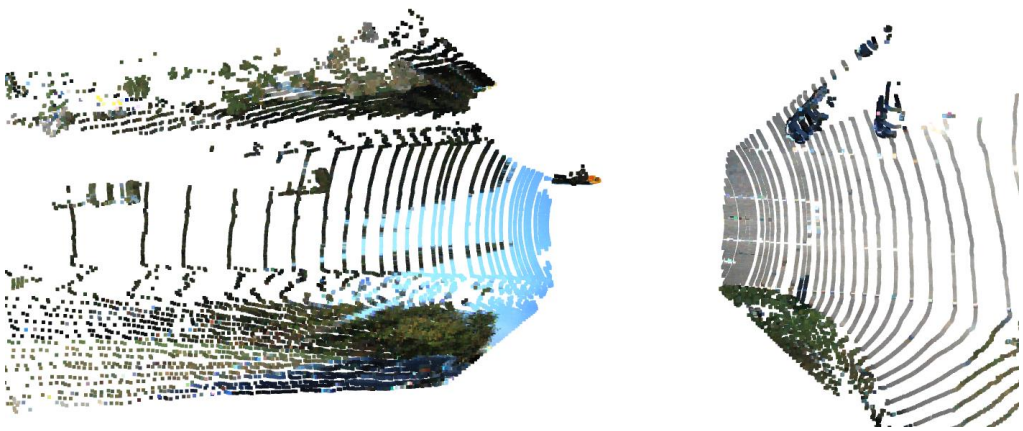


Figure 4.2: Point Cloud of the Testing Sample 000001 Viewed from the Top.

Figure 4.2 shows that the point cloud is separated into two regions. The point cloud at the left consists of the points that lie behind the camera, while the point cloud at the right consists of the points that lie in front of the camera, which is identical to the point cloud shown in Figure 4.1 but viewed from the top. It is worth noticing that there are objects in the left point cloud; therefore, 3D object localization should be worked on those objects to obtain their 3D coordinates and three primary dimensions. With the inclusion of rear points into the vertices and edges' graph construction by equation 8, the combined numerical result for the testing sample 000001 is generated and is shown in Table 4.2.

Table 4.2: Combined Numerical Result for the Testing Sample 000001.

| **Object class** | Car (1) | Car (2) | Car (3) |
|---|---|---|---|
| **Truncation** | -1 | -1 | -1 |
| **Occlusion** | -1 | -1 | -1 |
| **Alpha** | 0 | 0 | 0 |
| **Clip_Xmin** | 141.2073462 | 640.9306218 | 36.4174471 |
| **Clip_Ymin** | 176.2229789 | 118.7164306 | 174.0326211 |
| **Clip_Xmax** | 351.6813722 | 711.8680651 | 232.3333765 |
| **Clip_Ymax** | 243.9098258 | 166.1110190 | 289.8086621 |
| **Height** | 1.5215358 | 1.5051302 | 1.5745287 |
| **Width** | 1.6863366 | 1.6225216 | 1.6601882 |
| **Length** | 4.283223 | 3.9107132 | 4.113779 |
| **X3D** | -8.645287 | -2.2776604 | -7.8946705 |
| **Y3D** | 1.6074694 | 1.7630962 | 1.5971005 |
| **Z3D** | 17.363804 | -25.551062 | 11.835676 |
| **Yaw** | 0.09412821 | 1.4244215 | 0.72221255 |
| **Score** | 82.2164863 | 53.6548599 | 151.0746396 |

Table 4.2 shows that one more object is localized, the second car in the table, from the point cloud behind the camera compared to the results in Table 4.1. The second car detected has the Z3D of -25.551062, indicating the object lies behind the camera. Thus, the 3D coordinates and three primary

dimensions of the second car is successfully computed by the MLP of the localization branch. Point-GNN can now localize the objects in point clouds in a more comprehensive manner. Nevertheless, another car in the rear point cloud is observed in Figure 4.2 but is not localized by Point-GNN. One possible inference to this is the Point-GNN is only trained with the front point cloud but not both the front and the rear due to the absence of labelling of objects in the rear point cloud. Besides, the front point cloud is observed to have a higher point density than the rear point cloud, where the points in the rear point cloud are sparsely located in 3D space. This may be accounted for by the specification of the LiDAR camera, where it focuses on the front direction it is facing. The car that Point-GNN does not localize is locating further away from the LiDAR camera compared to the car localized in the rear point cloud. Although it is located within the Point-GNN's localization range, low point density causes insufficient information to categorize it as an interesting object in the point cloud. Hence, it possesses a lesser chance to be classified as a 'car', the target object for localization, by Point-GNN due to lesser spatial information of vertices and edges generated in the graph. Figure 4.3 (a) shows the front point cloud of the testing sample 000001, which has a higher point density, while Figure 4.3 (b) shows the rear point cloud of the testing sample 000001, which has a lower point density and the undetected car is indicated by a box.

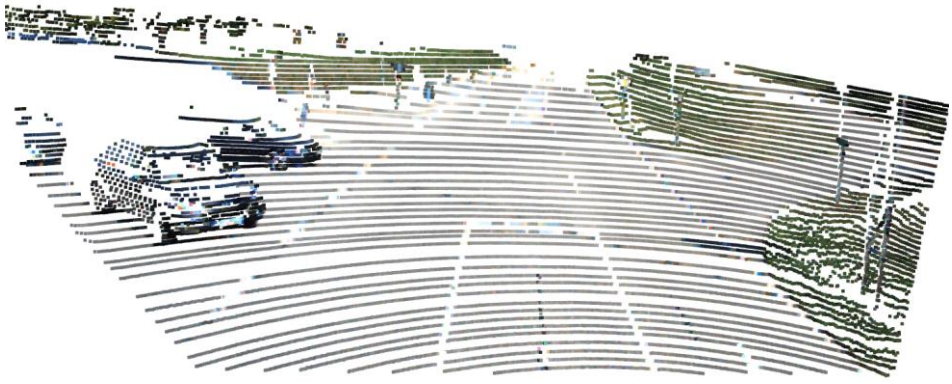


Figure 4.3: (a) Front Point Cloud of the Testing Sample 000001.

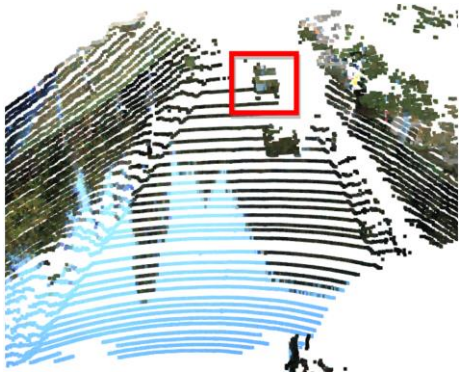


Figure 4.3: (b) Rear Point Cloud of the Testing Sample 000001 with the Labelled Unlocalized Object.

## 4.3 Comparison and Evaluation

The Point-GNN generated several results using the different number of iterations, known as the different number of layers in the Point-GNN model. The first layer in the Point-GNN model is always the graph layer of aggregating the LiDAR reflection intensity by MLP to initialize the vertices' state, where `max_point_set_pooling layer` type is introduced to the layer. The last layer in Point-GNN is always the output layer, where the vertices' state features are abstracted using the `class aware predictor` approach. The intermediate layers that lay between the first and the last layer of the Point-GNN model are the iteration layers. All layers are configured in the configuration file input to the code, as discussed previously.

The results in Table 4.1 and 4.2 are produced from the three-time-iterated Point-GNN model, representing three iteration layers in the model. Every iteration layer is identical to each other in terms of configuration. The iteration layer's configuration includes the activation type, normalization type, and depth parameter for each operating MLP. The layer type that every iteration layer used is `scatter_max_graph_auto_center_net`, where the auto offset feature is activated in every iteration of vertices' state aggregation to reduce translational variance. The depth parameter for MLP of edge and vertices update are identical, which is the list of [300, 300], indicating the number of nodes in the perceptron. All layers in the Point-GNN's model have the same activation function: ReLU non-linearity. It is because ReLU can preserve the sparsity of vertices in the graph while increasing the computation speed.

The maximum iteration number, three, is the iteration number that Point-GNN used to generate the validation result and submitted it to KITTI for evaluation. It has the greatest computation load, where the information of a vertex has reached a neighbourhood vertex with the relative distance of three edges or two vertices. The information cross does provide a larger receptive field around the vertices. Therefore, it has a higher tendency to localize objects accurately in 3D space. The results used for evaluation is only the objects localized in the front point cloud, where the labelled ground truth objects in the KITTI dataset only consist of the objects in the front.

The AP predicted for Point-GNN's result is separated into 'easy', 'moderate', and 'hard' classes, representing the difficulty of the objects being localized. This includes the factor of occlusion in LiDAR point cloud, which is shadows that result from geometric triangulation. The interesting objects may be occluded by the non-reflective object. This is because the hidden objects that are not visible to the LiDAR camera will not be measured. At the same time, multiple scanning from different angles is not implemented during KITTI dataset construction. The three difficulties represent the occlusion level of objects in the scene, from fully visible to significantly occluded. As the object's difficulty increases, the number of points and vertices representing the object is reduced. Therefore, the occluded car's vertices require more information from its neighbourhood vertices, where it needs to aggregate more local features to classify itself as an interesting object and localize it.

### 4.3.1 Intra-Approach Comparison

The AP predicted for the three-time-iterated Point-GNN's result is 87.890259 % for 'easy' cars, 78.342728 % for 'moderate' cars, and 77.377190 % for 'hard' cars using the 3D detection benchmark. It is worth noticing the KITTI evaluation used for intra-approach comparison has the recall position of 11. The result shows that the AP reduces from 'easy' to 'moderate' followed by 'hard' object's difficulty. As the number of vertices representing the interesting object decreases, the predicted bounding box does not accurately reflect the object's 3D coordinates. Translation of bounding box occurs, where the bounding box is minimally shifted to other nearby positions within the object's neighbourhood as a result of the localization MLP. This

reduces the IoU between the ground truth labelled and the predicted bounding box and results in lower AP.

Apart from the AP predicted using the 3D detection benchmark, the BEV detection benchmark is also used and predicted an AP value for each difficulty class. The AP predicted using BEV detection benchmark for the three-time-iterated Point-GNN's result is 89.823723 % for 'easy' cars, 88.308357 % for 'moderate' cars, and 87.155525 % for 'hard' cars. It shows that the AP predicted using the BEV detection benchmark is higher than 3D detection for every difficulty classes. The BEV detection benchmark included transforming the raw point cloud into BEV representation, which produces the BEV point cloud with higher point density in a 2D regular map and uses it as the input to Point-GNN. The BEV transformation preserves the object's size consistent with respect to distance, which provided a strong antecedent for inferencing because it does not suffer from occlusion. The AP predicted using the 3D detection and BEV detection benchmark is illustrated in Figure 4.4 (a) and (b), in terms of the area under the graph.

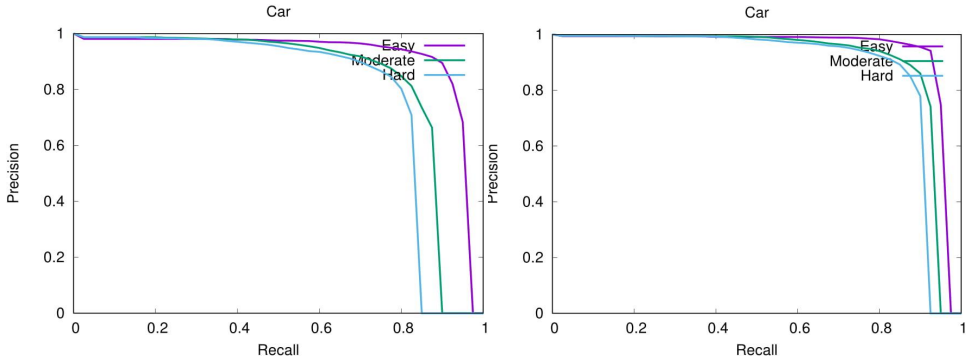



Figure 4.4: (a) AP Predicted using the 3D Detection Benchmark for the Three-time-iterated Point-GNN and (b) AP Predicted using the BEV Detection Benchmark for the Three-time-iterated Point-GNN.

As the Point-GNN's iteration number reduces, a vertex's information cannot be transferred far away from the origin. Therefore, a vertex's receptive field will be smaller and is anticipated a drop in AP predicted for all classes of difficulty. The AP predicted for Point-GNN's result with iteration number of 0, 1, 2, and 3 are listed in Table 4.3 and 4.4 using the 3D detection and the BEV detection benchmark, respectively. Besides, the AP predicted for Point-GNN's

result is also illustrated in Figure 4.5 (a), (b), and (c) that show the graph of AP using the 3D detection benchmark with the iteration number of 2, 1, and 0, respectively.

Table 4.3: AP Predicted for Point-GNN using the 3D Detection Benchmark with Different Iteration Numbers.

| Number of Iteration | Average Precision (3D Detection Benchmark) | | |
|---|---|---|---|
| | Easy (%) | Moderate (%) | Hard (%) |
| 0 | 73.896561 | 64.421539 | 59.909767 |
| 1 | 88.003227 | 77.887131 | 76.144211 |
| 2 | 88.336563 | 78.510040 | 77.671188 |
| 3 | 87.890259 | 78.342728 | 77.377190 |

Table 4.4: AP Predicted for Point-GNN using the BEV Detection Benchmark with Different Iteration Numbers.

| Number of Iteration | Average Precision (BEV Detection Benchmark) | | |
|---|---|---|---|
| | Easy (%) | Moderate (%) | Hard (%) |
| 0 | 87.241478 | 77.389450 | 75.839111 |
| 1 | 89.833923 | 87.672905 | 86.303192 |
| 2 | 89.995453 | 88.374153 | 87.218575 |
| 3 | 89.823723 | 88.308357 | 87.155525 |

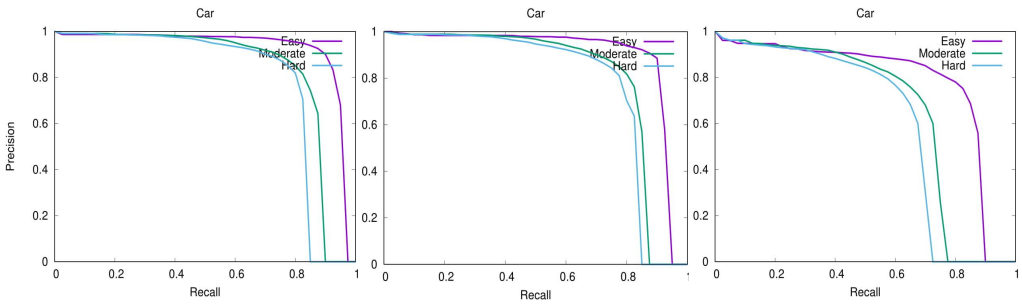



Figure 4.5: (a) AP Predicted for Two-time-iterated Point-GNN, (b) AP Predicted for One-time-iterated Point-GNN, and (c) AP Predicted for Non-iterated Point-GNN.

The result shows that the AP predicted for non-iterated and one-time-iterated Point-GNN have an increment of AP as the iteration number increases

when using both the 3D detection and the BEV detection benchmark. It has the AP difference of 14.106666 % for ‘easy’ cars, 13.465592 % for ‘moderate’ cars, and 16.234444 % for ‘hard’ cars when using the 3D detection benchmark. A vertex’s receptive field increased significantly when the iteration number increased from zero to one. This is because information exchange in Point-GNN is essential for feature abstraction, where it refines the vertices’ state feature before predicting the bounding boxes. A non-iterated Point-GNN will produce a result similar to the region proposal-based approach, where a massive number of 3D proposals will be generated prior to any feature engineering. The massive number of 3D proposals lower the model’s accuracy, where many false-positive results are introduced.

When Point-GNN is iterated at least one time, the AP increased significantly. This is due to the presence of a receptive field around vertices, where the edges deliver the information from neighbourhood vertices to a vertex. The edges are formed with the radius for searching neighbourhood vertices at 4.0 m originating from a vertex, with the maximum number of edges of 256 that indicate the neighbour numbers for each vertex. The wide range of searching and great neighbour numbers causes a vertex’s receptive field to change dramatically when the iteration number is changed by one. Therefore, the AP predicted for non-iterated Point-GNN is the lowest, followed by one-time-iterated and two-time-iterated Point-GNN.

The AP does not change significantly when the iteration number changes from one to two. This is because most of the essential and meaningful features are aggregated during the first iteration. The iterations afterwards refine the minor features of the vertices to bring the result closer to the ground truth. Therefore, the AP increased from one-time-iterated Point-GNN to two-time-iterated Point-GNN, which is 0.333336 % for ‘easy’ cars, and 0.622909 % for ‘moderate’ cars, and 1.526977 % for ‘hard’ cars when using the 3D detection benchmark.

Surprisingly, the two-time-iterated Point-GNN achieves the predicted AP higher than the three-time-iterated Point-GNN for all difficulty classes, with the difference of 0.446304 % for ‘easy’ cars, 0.167312 % for ‘moderate’ cars, and 0.293998 % for ‘hard’ cars when using the 3D detection benchmark. This is because the three-time-iterated Point-GNN model has trained with the

highest iteration number, which results in a deeper neural network. Training difficulty occurs, where the accuracy is saturated, followed by degradation as the number of layers in the neural network increases. The Point-GNN has successfully determined a compelling set of state's values that optimize the object localization at the second iteration. Therefore, the third iteration of Point-GNN produces noise to the vertices' state by creating an identity function from the ReLU non-linearity activation function (He, et al., 2015). This problem requires further improvement, such as adding a residual learning layer to the third Point-GNN iteration to skip over the activation and improve the deeper neural network's quality (He, et al., 2015).

### 4.3.2 Inter-Approach Comparison

The Point-GNN's result was submitted to KITTI for general evaluation, and they use a recall position of 40. The AP predicted for three-time-iterated Point-GNN's result using the 3D detection benchmark is 88.33 % for 'easy' cars, 79.47 % for 'moderate' cars, and 72.29 % for 'hard' cars. Compared to the result in Table 4.3, which is also tested with the 3D detection benchmark, the AP predicted by KITTI is higher. This is because the increase in the number of recall positions will increase the AP predicted. KITTI has claimed that the use of 40 recall positions during the evaluation will produce a more fair comparison among the approaches. Therefore, many approaches have submitted their results to KITTI for general evaluation, including all approaches discussed in chapter two. Table 4.5 shows the 3D object localization results of various approaches on the KITTI test 3D detection benchmark.

Table 4.5: Comparative 3D Object Localization Results of Various Approaches on the KITTI Test 3D Detection Benchmark, with an IoU Threshold of 0.7 for 3D Bounding Box (Guo, et al., 2020).

| Approaches | Modality | Average Precision for Cars | | |
|---|---|---|---|---|
| | | Easy (%) | Moderate (%) | Hard (%) |
| F-PointNet | LiDAR, Image | 82.19 | 69.79 | 60.59 |
| F-ConvNet | LiDAR, Image | 87.36 | 76.39 | 66.69 |

| | | | | |
|---|---|---|---|---|
| MV3D | LiDAR, Image | 74.97 | 63.63 | 54.00 |
| ContFuse | LiDAR, Image | 83.68 | 68.78 | 61.67 |
| PointRCNN | LiDAR | 86.96 | 75.64 | 70.70 |
| PointRGCN | LiDAR | 85.97 | 75.73 | 70.60 |
| PIXOR | LiDAR | - | - | - |
| BirdNet | LiDAR | 13.53 | 9.47 | 8.49 |
| Vote3Deep | LiDAR | - | - | - |
| VoxelNet | LiDAR | 77.47 | 65.11 | 57.73 |
| 3DSSD | LiDAR | 88.36 | 79.57 | 74.55 |
| Point-GNN | LiDAR | 88.33 | 79.47 | 72.29 |

The results show that the AP predicted using the 3D detection benchmark is lower than when using the BEV detection benchmark, as discussed above. When comparing Table 4.5 with Table 2.1, where Table 2.1 consists of the 3D object localization results of various approaches on the BEV detection benchmark, the AP's ranking has changed. PointRCNN achieves the highest AP for 'easy' cars among all approaches when using the BEV detection benchmark, but it ranked four among the twelve approaches when using the 3D detection benchmark. This could be due to the preference of the input modal's form. For instance, the AP predicted by BirdNet's result drops significantly when changing from the BEV detection benchmark to 3D detection, where the BirdNet's model is explicitly designed for BEV point cloud input. PointRCNN segments the BEV point cloud more efficiently compared to the raw point cloud, where the absence of occlusion factor allows segmenting of near-perfect points for high-quality proposals generation and bounding box refinement. As discussed in chapter two, the performance of PointRCNN depends on the points segmented in the first stage.

Nevertheless, the 3D detection benchmark is more general than the BEV detection benchmark to be evaluated and discussed, where it reflects the actual situation in the scene. Point-GNN does prefer using the 3D detection benchmark, although the AP predicted using the BEV detection benchmark is higher. Point-GNN ranked second among the twelve approaches, and the AP predicted is slightly lesser than the 3DSSD approach. The AP difference for

Point-GNN and 3DSSD is 0.03 % for ‘easy’ cars, 0.10 % for ‘moderate’ cars, and 2.26 % for ‘hard’ cars when using the 3D detection benchmark. Both 3DSSD and Point-GNN utilize the preservation of the point cloud’s sparsity to maintain its geometrical features during processing. This reduces the data loss that occurs in many feature engineering, whereby the data is being transformed into other originality-altered representations. Although 3DSSD surpasses Point-GNN for the AP prediction, Point-GNN has the highest AP predicted when iterated twice, while the result submitted to KITTI is the three-time-iterated Point-GNN’s result. Therefore, Point-GNN’s actual performance is more excellent than 3DSSD when localizing ‘easy’ and ‘moderate’ cars due to slight AP difference. However, Point-GNN might not be as reliable as 3DSSD when localizing ‘hard’ cars, given that the significant AP difference between the two approaches of 2.29 %. This may be due to 3DSSD performs better in feature abstraction than Point-GNN for the lower number of available points. 3DSSD utilizes two furthest point sampling layers to sample the points and is capable of sampling points without introducing data loss.

## 4.4 Summary

The KITTI dataset that consists of the velodyne point cloud, image, label, and calibration files are obtained. The KITTI dataset is used as the input to the Point-GNN algorithm. Point-GNN has outputted numerical and graphical results, displaying the X, Y, and Z coordinates of the localized object as well as its three primary dimensions. The Point-GNN algorithm has been modified to include the points and objects behind the camera, which causes the result produced more comprehensive for the scene. Comparison and evaluation for intra and inter-approach have been conducted. Point-GNN performs the best when the model is iterated twice. Besides, Point-GNN surpasses most of the 3D object localization approaches when using the 3D detection benchmark, but its AP predicted for ‘hard’ cars is lower than the 3DSSD approach.

# CHAPTER 5

# CONCLUSIONS AND RECOMMENDATIONS

## 5.1 Conclusions

The extensive and comprehensive KITTI dataset consists of 7481 training samples and 7518 testing samples, which are costly if the dataset is constructed individually. It is identified and used to generate the results and used for intra and inter-approach comparison. Besides, twelve 3D object localization approaches are deeply reviewed and compared to select the best among them. Point-GNN is a distinctive 3D object localization approach and is selected due to its high AP result and its simplicity in the architecture design. It preserves the point cloud's sparsity and takes advantage of the natural geometrical information of the points. Therefore, Point-GNN can accurately localize the objects in the point cloud and obtain the objects' three primary dimensions. Point-GNN is further modified to include the computation for the rear point cloud to perform a comprehensive 3D object localization. Point-GNN performs the best when the model is iterated twice, which the AP predicted for the model is the highest among all iteration numbers, with the value of 88.336563 % for 'easy' cars, 78.510040 % for 'moderate' cars, and 77.671188 % for 'hard' cars when using the 3D detection benchmark. Apart from that, Point-GNN outperforms most of the other literary works, where it achieves state-of-the-art among the approaches along with 3DSSD. Therefore, all objectives are achieved.

## 5.2 Recommendations for Future Work

This project can be improved in many ways due to the project's limitation. Due to the insufficiently powerful computer available, Point-GNN cannot output the graphical result using the three-time-iterated Point-GNN's model. Currently, Point-GNN can only display the graphical result for 'easy' cars detection, and the numerical result itself cannot illustrate the robustness of Point-GNN in localizing highly occluded objects. Therefore, enhancing the specification of the computer used to execute Point-GNN is necessary, where

at least the enhanced specifications can meet the minimum requirement to execute Point-GNN at three iterations that are anticipated at 20 GB of RAM and 60 FPS on-screen graphic display. By doing so, Point-GNN can display the 'hard' class occluded objects being localized by 3D bounding boxes in the point cloud.

Next, Point-GNN is only trained with the front point cloud due to the absence of label for objects in the rear point cloud. This reduces the effectiveness of Point-GNN in localizing the objects behind the camera. Another extensive and comprehensive dataset should be obtained, which consists of the label for all objects in the point cloud. Therefore, Point-GNN can be trained with the new dataset to include objects in the rear point cloud during the training process to effectively localize all objects in the point cloud regardless of the object's position during inferencing. This is used to develop a robust 3D object localization approach that localizes objects in all directions.

Besides, Point-GNN's third iteration does not improve the AP compared to the two-time-iterated Point-GNN's model. This is because of the noise introduced into the vertices' state during the third iteration as a result of a very deep neural network. Hence, a residual learning layer should be added to the third Point-GNN iteration to skip over the ReLU activation during training, as discussed in chapter 4. There will be a branch from the two-time-iterated model to the third iteration output and forms an identity mapping. This will tell the training error at the third iteration should not be greater than the two-time-iterated model. Adding this layer will improve the deeper neural network's quality, and the model can be trained with the same iterative method, stochastic gradient descent (He, et al., 2015). Figure 5.1 shows the residual learning's building block.

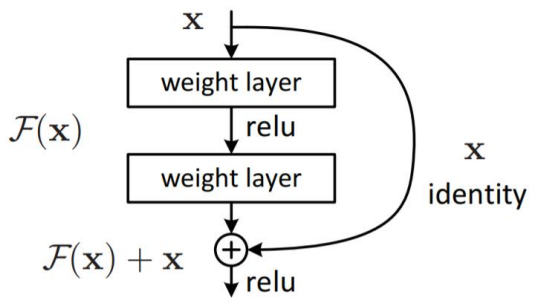



Figure 5.1: Residual Learning's Building Block (He, et al., 2015).

# REFERENCES

Alkabawi, E.M., Hilal, A.R. and Basir, O.A., 2017. Feature abstraction for early detection of multi-type of dementia with sparse auto-encoder. *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*, 2017-January, pp.3471–3476.

Beltrán, J., Guindel, C., Moreno, F.M., Cruzado, D., García, F. and Escalera, A. de la, 2018. BirdNet: A 3D Object Detection Framework from LiDAR Information. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*. [online] Available at: <http://arxiv.org/abs/1911.12236> [Accessed 22 August 2020].

Brownlee, J., 2019. *A gentle introduction to object recognition with deep learning*. Machine Learning Mastery, [blog] 22 May. Available at: <https://machinelearningmastery.com/object-recognition-with-deep-learning/> [Accessed 15 July 2020].

Chen, X., Ma, H., Wan, J., Li, B. and Xia, T., 2017. Multi-view 3D object detection network for autonomous driving. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*.

Engelcke, M., Rao, D., Wang, D.Z., Tong, C.H. and Posner, I., 2017. Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. *Proceedings - IEEE International Conference on Robotics and Automation*.

FME Community, 2020. *What is a point cloud? What is LiDAR?* [online] Available at: <https://community.safe.com/s/article/what-is-a-point-cloud-what-is-lidar> [Accessed 16 July 2020].

Geiger, A., Lenz, P., Stiller, C. and Urtasun, R., 2013. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research. *The International Journal of Robotics Research*, (October), pp.1–6.

Gray, D., n.d. *What are point clouds? 5 easy facts that explain point clouds*. [online] Available at: <https://info.vercator.com/blog/what-are-point-clouds-5-easy-facts-that-explain-point-clouds> [Accessed 1 September 2020].

Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L. and Bennamoun, M., 2020. Deep Learning for 3D Point Clouds: A Survey. [online] Available at: <http://arxiv.org/abs/1912.12033> [Accessed 11 August 2020].

He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December, pp.770–778.

Hui, J., 2018. *mAP (mean Average Precision) for object detection*. Medium, [online] 7 Mar. Available at: <https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173> [Accessed 12 September 2020].

Hyams, G. and Malowany, D., 2020. *The battle of speed vs. accuracy Single-shot vs two-shot detection meta-architecture*. Clear | ML, [blog] 8 March. Available at: <https://allegro.ai/blog/the-battle-of-speed-accuracy-single-shot-vs-two-shot-detection/> [Accessed 31 August 2020].

Keshari, K., 2020. *Object detection tutorial in TensorFlow: Real-time object detection*. Edureka, [blog] 14 May. Available at: <https://www.edureka.co/blog/tensorflow-object-detection-tutorial/> [Accessed 2 September 2020].

Liang, M., Yang, B., Wang, S. and Urtasun, R., 2018. Deep Continuous Fusion for Multi-sensor 3D Object Detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Protasiewicz, J., 2018. *Why is python so good for AI, machine learning and deep learning?*. Netguru, [blog] 31 August. Available at: <https://www.netguru.com/blog/python-ai> [Accessed 2 September 2020].

Qi, C.R., Liu, W., Wu, C., Su, H. and Guibas, L.J., 2018. Frustum PointNets for 3D Object Detection from RGB-D Data. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Qi, C.R., Su, H., Mo, K. and Guibas, L.J., 2017. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua, pp.77–85.

Qi, C.R., Yi, L., Su, H. and Guibas, L.J., 2017. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 2017-Decem, pp.5100–5109.

Sayantini, 2020. *Keras vs TensorFlow vs PyTorch: Comparison of the deep learning frameworks*. Edureka, [blog] 24 June. Available at: <https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/> [Accessed 2 September 2020].

Shi, S., Wang, X. and Li, H., 2019. PointRCNN: 3D object proposal generation and detection from point cloud. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Shi, W. and Rajkumar, R., 2020. Point-GNN: Graph Neural Network for 3D Object Detection in a Point Cloud. [online] Available at: <http://arxiv.org/abs/2003.01251> [Accessed 24 August 2020].

Singh, A.R., 2018. *What is LiDAR and how does it work?*. Geospatial world, [blog] 2 June. Available at: <https://www.geospatialworld.net/blogs/what-is-lidar-and-how-does-it-work/> [Accessed 1 September 2020].

Thomson, C., 2019. *LiDAR vs point clouds: Learn the basics of laser scanning, 3D surveys and reality capture.* Vercator, [blog] 27 May. Available at: <https://info.vercator.com/blog/lidar-vs-point-clouds> [Accessed 14 July 2020].

Wang, Z. and Jia, K., 2019. Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal. *IEEE International Conference on Intelligent Robots and Systems*.

Yang, B., Luo, W. and Urtasun, R., 2018. PIXOR: Real-time 3D Object Detection from Point Clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Yang, Z., Sun, Y., Liu, S. and Jia, J., 2020. 3DSSD: Point-based 3D Single Stage Object Detector. [online] Available at: <http://arxiv.org/abs/2002.10187> [Accessed 25 August 2020].

Zarzar, J., Giancola, S. and Ghanem, B., 2019. PointRGCN: Graph Convolution Networks for 3D Vehicles Detection Refinement. [online] Available at: <http://arxiv.org/abs/1911.12236> [Accessed 20 August 2020].

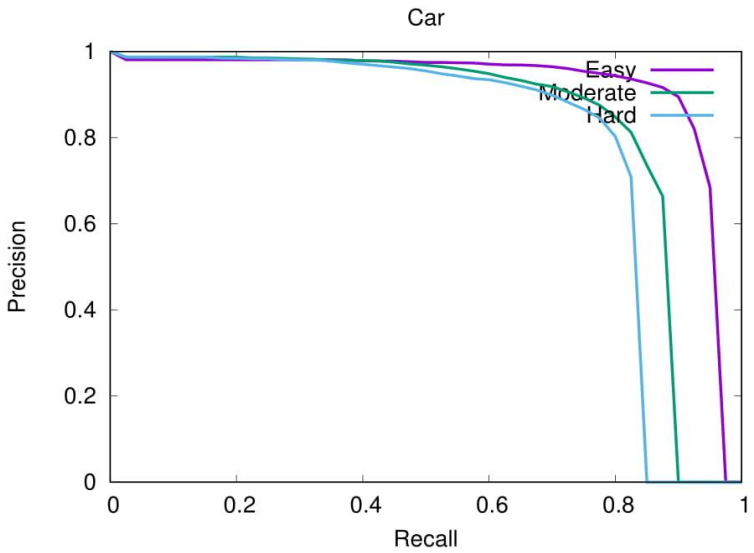
Zhang, E. and Zhang, Y., 2009. Average Precision. In: L. LIU and M.T. ÖZSU, eds. *Encyclopedia of Database Systems*. [online] Boston, MA: Springer US.pp.192–193. Available at: <https://doi.org/10.1007/978-0-387-39940-9_482> [Accessed 26 August 2020].

Zhao, Z.Q., Zheng, P., Xu, S.T. and Wu, X., 2019. Object Detection with Deep Learning: A Review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11), pp.3212–3232.

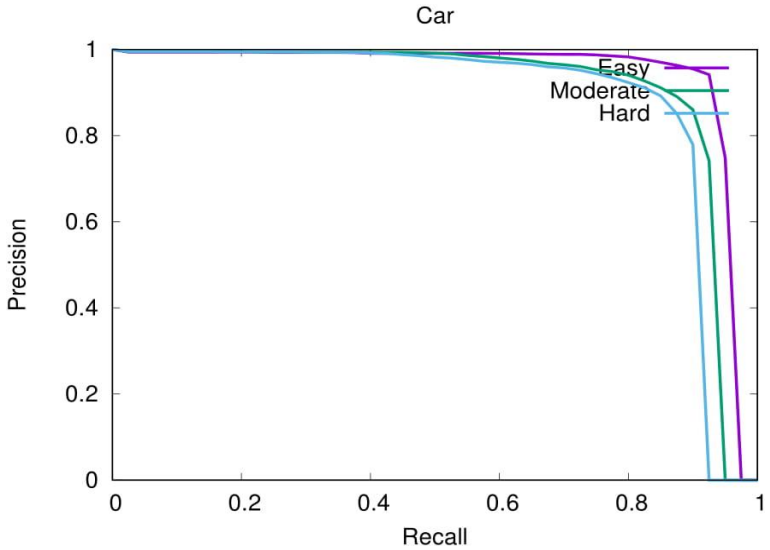
Zhou, Y. and Tuzel, O., 2017. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. *Computers in Education Journal*.

# APPENDICES

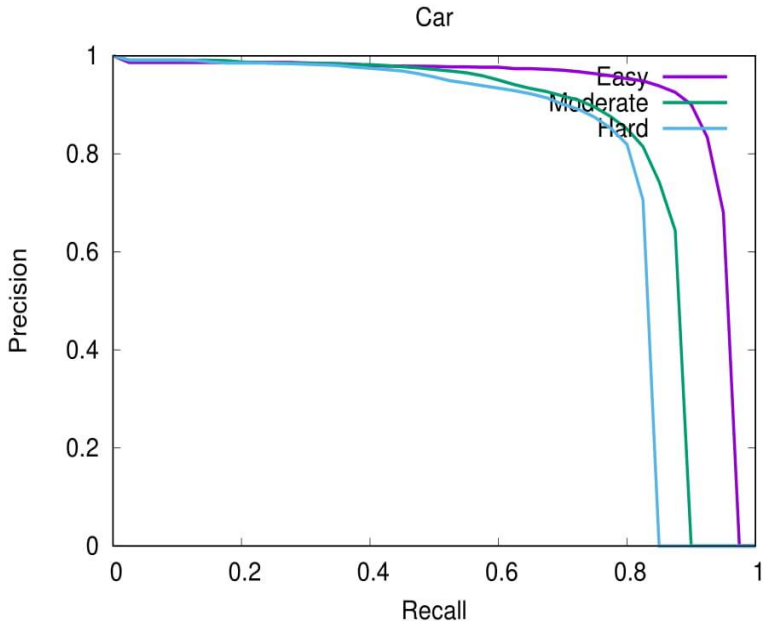
## APPENDIX A: Graphs



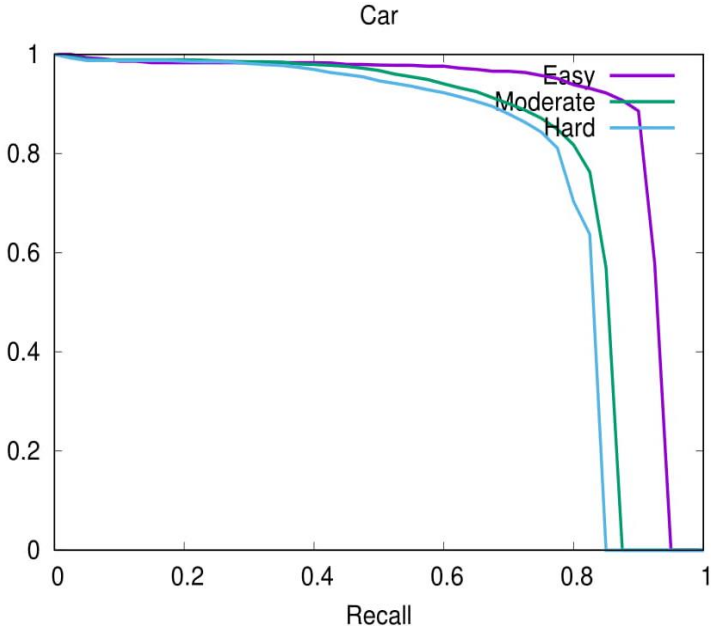

GraphA-1: AP Predicted using the 3D Detection Benchmark for the Three-time-iterated Point-GNN.



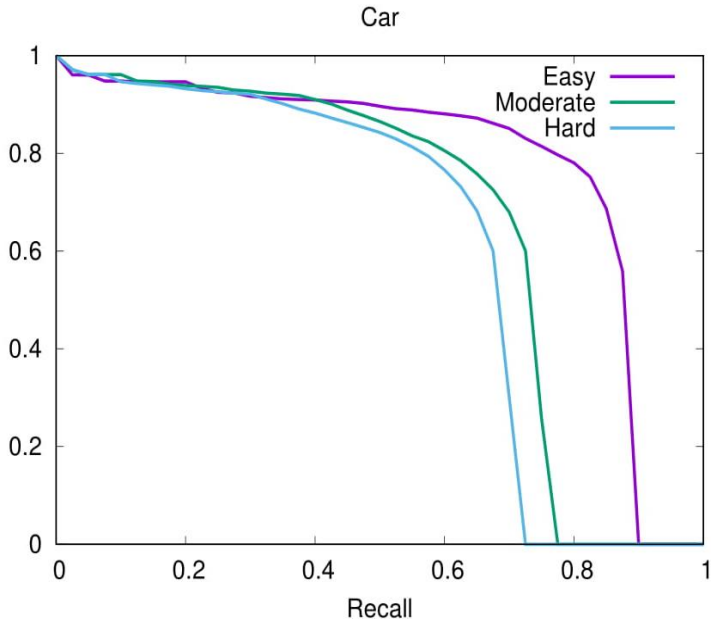

GraphA-2: AP Predicted using the BEV Detection Benchmark for the Three-time-iterated Point-GNN.




GraphA-3: AP Predicted for Two-time-iterated Point-GNN using the 3D Detection Benchmark.




GraphA-4: AP Predicted for One-time-iterated Point-GNN using the 3D Detection Benchmark.




GraphA-5: AP Predicted for Non-iterated Point-GNN using the 3D Detection Benchmark.

# APPENDIX B: Tables

TableB-1: Comparative 3D Object Localization Results of Various Approaches on the KITTI Test BEV Detection Benchmark, with an IoU Threshold of 0.7 for 3D Bounding Box (Guo, et al., 2020).

| Approaches | Modality | Average Precision for Cars | | |
|---|---|---|---|---|
| | | Easy (%) | Moderate (%) | Hard (%) |
| F-PointNet | LiDAR, Image | 91.17 | 84.67 | 74.77 |
| F-ConvNet | LiDAR, Image | 91.51 | 85.84 | 76.11 |
| MV3D | LiDAR, Image | 86.62 | 78.93 | 69.80 |
| ContFuse | LiDAR, Image | 94.07 | 85.35 | 75.88 |
| PointRCNN | LiDAR | 92.13 | 87.39 | 82.72 |
| PointRGCN | LiDAR | 91.63 | 87.49 | 80.73 |
| PIXOR | LiDAR | 83.97 | 80.01 | 74.31 |
| BirdNet | LiDAR | 76.88 | 51.51 | 50.27 |
| Vote3Deep | LiDAR | - | - | - |
| VoxelNet | LiDAR | 89.35 | 79.26 | 77.39 |
| 3DSSD | LiDAR | 92.66 | 89.02 | 85.86 |
| Point-GNN | LiDAR | 93.11 | 89.17 | 83.90 |

TableB-2: Gantt Chart for Project Part I.

| No. | Activities \ Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M1 | Project planning | ✔ | ✔ | | | | | | | | | | | | |
| M2 | Literature review | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | | | | |
| M3 | Research methodology | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| M4 | Report writing & oral presentation | | | | | | | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

TableB-3: Gantt Chart for Project Part II.

| No. | Activities \ Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M1 | Execution of Point-GNN's code | ✔ | ✔ | ✔ | ✔ | ✔ | | | | | | | | | |
| M2 | Research methodology | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | | | |
| M3 | Model development | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| M4 | Result and discussion | | | | | | | | ✔ | ✔ | ✔ | ✔ | ✔ | | |
| M5 | Poster preparation | | | | | | | | | ✔ | ✔ | ✔ | ✔ | | |
| M6 | Report writing & oral presentation | | | | | | | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

TableB-4: Comparison Between Specifications of the Student's Computer and the Research Server.

| | Student's computer | UTAR research server |
|---|---|---|
| Operating system | Windows | Linux |
| RAM (GB) | 12 | 64 |
| GPU | NVIDIA GeForce GTX 1050 | NVIDIA GP104GL |
| On-screen (FPS) | 60.0 | 13.0 |

TableB-5: Numerical Result for the Testing Sample 000001.

| **Object class** | Car (1) | Car (2) |
|---|---|---|
| **Truncation** | -1 | -1 |
| **Occlusion** | -1 | -1 |
| **Alpha** | 0 | 0 |
| **Clip_Xmin** | 141.20734621538205 | 36.41744711056066 |
| **Clip_Ymin** | 176.2229789093736 | 174.03262110345074 |
| **Clip_Xmax** | 351.68137221248566 | 232.33337647692284 |
| **Clip_Ymax** | 243.90982579877945 | 289.8086621292917 |
| **Height (m)** | 1.5215358 | 1.5745287 |
| **Width (m)** | 1.6863366 | 1.6601882 |
| **Length (m)** | 4.283223 | 4.113779 |
| **X3D** | -8.645287 | -7.8946705 |
| **Y3D** | 1.6074694 | 1.5971005 |
| **Z3D** | 17.363804 | 11.835676 |
| **Yaw** | 0.09412821 | 0.72221255 |
| **Score** | 82.2164862876902 | 151.07463956706272 |

TableB-6: Combined Numerical Result for the Testing Sample 000001.

| **Object class** | Car (1) | Car (2) | Car (3) |
|---|---|---|---|
| **Truncation** | -1 | -1 | -1 |
| **Occlusion** | -1 | -1 | -1 |
| **Alpha** | 0 | 0 | 0 |
| **Clip_Xmin** | 141.2073462 | 640.9306218 | 36.4174471 |
| **Clip_Ymin** | 176.2229789 | 118.7164306 | 174.0326211 |
| **Clip_Xmax** | 351.6813722 | 711.8680651 | 232.3333765 |
| **Clip_Ymax** | 243.9098258 | 166.1110190 | 289.8086621 |
| **Height** | 1.5215358 | 1.5051302 | 1.5745287 |
| **Width** | 1.6863366 | 1.6225216 | 1.6601882 |
| **Length** | 4.283223 | 3.9107132 | 4.113779 |
| **X3D** | -8.645287 | -2.2776604 | -7.8946705 |
| **Y3D** | 1.6074694 | 1.7630962 | 1.5971005 |
| **Z3D** | 17.363804 | -25.551062 | 11.835676 |
| **Yaw** | 0.09412821 | 1.4244215 | 0.72221255 |
| **Score** | 82.2164863 | 53.6548599 | 151.0746396 |

TableB-7: AP Predicted for Point-GNN using the 3D Detection Benchmark with Different Iteration Numbers.

| Number of Iteration | Average Precision (3D Detection Benchmark) | | |
|---|---|---|---|
| | Easy (%) | Moderate (%) | Hard (%) |
| 0 | 73.896561 | 64.421539 | 59.909767 |
| 1 | 88.003227 | 77.887131 | 76.144211 |
| 2 | 88.336563 | 78.510040 | 77.671188 |
| 3 | 87.890259 | 78.342728 | 77.377190 |

TableB-8: AP Predicted for Point-GNN using the BEV Detection Benchmark with Different Iteration Numbers.

| Number of Iteration | Average Precision (BEV Detection Benchmark) | | |
|---|---|---|---|
| | Easy (%) | Moderate (%) | Hard (%) |
| 0 | 87.241478 | 77.389450 | 75.839111 |
| 1 | 89.833923 | 87.672905 | 86.303192 |
| 2 | 89.995453 | 88.374153 | 87.218575 |
| 3 | 89.823723 | 88.308357 | 87.155525 |

TableB-9: Comparative 3D Object Localization Results of Various Approaches on the KITTI Test 3D Detection Benchmark, with an IoU Threshold of 0.7 for 3D Bounding Box (Guo, et al., 2020).

| Approaches | Modality | Average Precision for Cars | | |
|---|---|---|---|---|
| | | Easy (%) | Moderate (%) | Hard (%) |
| F-PointNet | LiDAR, Image | 82.19 | 69.79 | 60.59 |
| F-ConvNet | LiDAR, Image | 87.36 | 76.39 | 66.69 |
| MV3D | LiDAR, Image | 74.97 | 63.63 | 54.00 |
| ContFuse | LiDAR, Image | 83.68 | 68.78 | 61.67 |
| PointRCNN | LiDAR | 86.96 | 75.64 | 70.70 |
| PointRGCN | LiDAR | 85.97 | 75.73 | 70.60 |
| PIXOR | LiDAR | - | - | - |
| BirdNet | LiDAR | 13.53 | 9.47 | 8.49 |
| Vote3Deep | LiDAR | - | - | - |
| VoxelNet | LiDAR | 77.47 | 65.11 | 57.73 |
| 3DSSD | LiDAR | 88.36 | 79.57 | 74.55 |
| Point-GNN | LiDAR | 88.33 | 79.47 | 72.29 |