Predicting Cuboids from Partial LIDAR Point Clouds

ESE-650 Team 77 Spring 2023 Project

Derek Cheng [derekch@seas], David Yan [yandavid@seas], Beiming Li [beimingl@seas],

***Abstract*— We aim to improve the prediction of cuboidal bounding boxes for cars given LiDAR data, by showing that applying a Deep Learning-based approach (PointRCNN) can outperform existing geometric approaches by requiring less accumulation pointclouds to generate accurate bounding box prediction. This project proposes a pipeline for labelling, generating data, and pre-processing data to convert the data into proper inputs for the PointRCNN model. Our training and evaluation results demonstrate the learning-based approach is able to predict bounding boxes for segmented car pointclouds with a single LiDAR scan with reasonable results. Further work to improve the model accuracy can be taken by re-training with a larger dataset to reduce overfitting and improved training resources, training with few-shot instead of single-shot LiDAR scans, with the goal of deploying the model on a live quadrotor for real-time bounding box prediction.**

Code: https://github.com/david-yan/semantic_car_mapping

I. Introduction

The goal of our project is to improve the Semantic Mapping pipeline of aerial drones in the Semantic Localization Odometry and Mapping (SLOAM) system, which is currently in development in Kumar's Laboratory. Specifically, we aim to improve the prediction of cuboidal bounding boxes for cars given LiDAR data. Since the current system uses a geometric approach, which must make assumptions about the car's shape, to predict these boxes, we instead wanted to leverage Deep Learning to see if we could do better. In order to achieve this, we needed to record and process the raw LiDAR data, and then train on a neural network architecture, which we chose to be PointRCNN.

A. Contributions

Our contributions for this project can be summarized as follows:

1) Developed ROS node to label ROS bags in RVIZ.
2) Labeled three ROS bags of LiDAR data containing a total of around 60 cars.
3) Developed data generation script that would combine the labeled cuboids with individual LiDAR scans.
4) Developed processing programs to correct data recorded from running FasterLIO, such as skew of the ground plane, as well as augment the data for training.
5) Developed visualization scripts to display and debug.
6) Performed the two-stage training process of PointR-CNN on the processed data.
7) Evaluated the model performance both on held out data, and visualization in RVIZ.

II. Background

Creating an environment model based on a set of semantic models is crucial for large-scale autonomous mapping tasks and long-range exploration. Semantic maps offer viewpoint invariance and can provide long-term localization constraints for robot teams, such as loop closure and intra-robot registration in the form of graph-based SLAM [1]. Sparse semantic representations are particularly useful in multi-robot settings with limited communication bandwidth, as they significantly reduce storage requirements. In the Kumar Lab at Penn, state-of-the-art research is conducted in Semantic Localization Odometry and Mapping (SLOAM) [1], with the goal of actively modeling bounding boxes for various classes of objects and obstacles during flight, to reduce dense pointclouds into semantic representations of relevant objects. Lots of flight data has been previously collected on our Falcon 4 platform [1], which features an on-board NUC 10 with i7-10710U processor and Ouster OS1-64 LiDAR. This LiDAR data has been labelled and trained on a RangeNet++ [2] based segmentation model to provide semantic labels for trees, cars, roads, and various other classes. This project will leverage this previously trained segmentation model to provide segmented point clouds for our bounding box predictions.

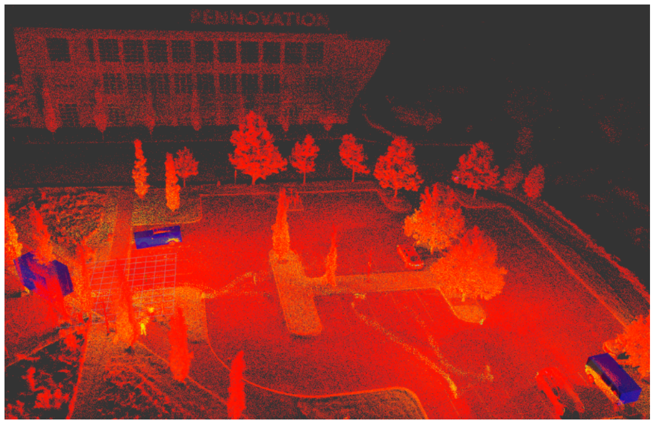

Fig. 1: Cuboid modeling from SLOAM [1] using geometric approaches. Predicted bounding boxes for cars shown in blue.

III. Related Work

The existing method for Active Metric-Semantic Mapping [1] uses a naive geometric approach for estimating cuboidal bounding boxes for cars as seen above in Fig 1. This method does not work well in cases where only part of the car is observed, as the bounding box is determined using the 2D convex hull of the projected points. Additionally it makes a few naive assumptions, such as that the front of the car is lower than the back when determining the direction of the car. These drawbacks will often result in an incorrect semantic map.

We aim to improve on the generated bounding boxes by leveraging deep learning techniques. Specifically, we will attempt to use PointRCNN [3], a network that takes a two staged approach to predicting bounding boxes from irregular 3D point clouds. The paper provides both code that we can start with, and a pre-trained model that we can perform fine-tuning on. To make this more tractable on the limited hardware of an aerial robot, we will experiment with reducing the layers of the proposed architecture in [3]. We evaluate our model on labeled data (ROS bags recorded at Pennovation) from the Active Metric-Semantic Mapping project[1], and use the results of that paper to benchmark our results.

IV. Approach

A. Data Collection

We acquired the ROS bags from members of Kumar's lab who had previously flown the drone over the parking lot at Pennovation. The raw LiDAR data was transformed into `(x, y, z, intensity)` points using odometry data, and the FasterLIO tracking algorithm.

1) Data Labeling: Our first task was to label the processed data with the cuboids. To aid in this, we developed two ROS nodes. One was an interactive cuboid that we could shift its pose for. We would manually shift the cuboids until the car was properly bounded, and then record the result (of the form `(x, y, z, qx, qy, qz, qw)` in the world frame). The second node was one that would visualize all of our recorded cuboids to ensure we did not miss any. A visualization of one of our labeled bags can be seen in Figure 2.

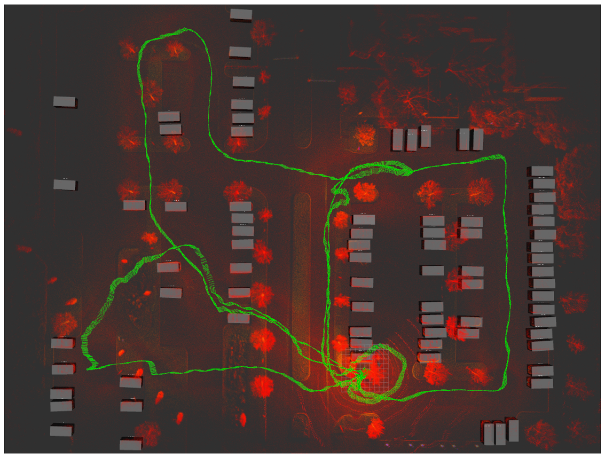

Fig. 2: Labeled cuboids.

2) Data Generation: We then created a script to generate the data we would feed into the neural network. To do this, we needed to convert the cuboids, which were recorded in the world frame, to the LiDAR frame, and for containment of the points. Transforming the cuboids to the LiDAR frame was simple, as we were given the transformation matrices from FasterLIO. To check for containment of points, we would transform the point cloud into each cuboid frame, using `(x, y, z)` as the origin, and `(qx, qy, qz, qw)` as the orientation, and then count the points around the origin within the bounds of the box. To ensure that there were an adequate number of points to predict a cuboid, we used a threshold of 70 points to qualify a cuboid as a label. A visualization of a single labeled LiDAR point cloud can be seen in Figure 3.

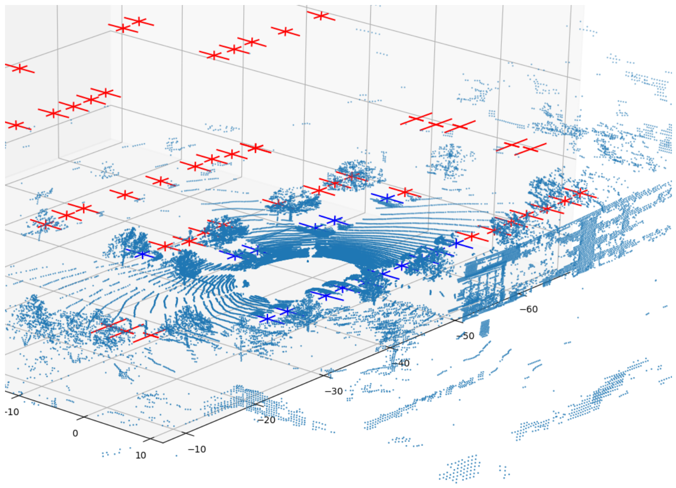

Fig. 3: LiDAR point cloud with intersecting cuboids (blue) and non-intersecting cuboids (red).

B. Data Pre-Processing

Before feeding the data into the network, we needed to perform a bit of processing to help us achieve better results. Specifically, we noticed from our visualizations that the ground planes were skewed in all of our scans (Figure 4). This skew was inconsistent between scans, thus we needed to correct for it if we wanted to achieve consistent results.

1) Ground Plane Segmentation: To do this, we needed to first segment the ground plane. We ran RANSAC on each scan to find the best fitting plane ($ax + by + cz + d = 0$).

2) Correcting Point Cloud: We then calculated the rotation to correct the points by calculating the angle between the normal vector to the plane, $w = (a, b, c)$, and the desired normal vector to the ground plane, $w_{des} = (0, 0, 1)$. The equations we used to calculate to axis-angle representation and the convert to quaternion were as follows:

$$\begin{aligned} a &= w \times w_{des} \\ \theta &= \arccos(w \cdot w_{des}) \\ q &= \begin{bmatrix} \sin(\frac{\theta}{2}) * a & \cos(\frac{\theta}{2}) \end{bmatrix} \end{aligned} \tag{1}$$

This correction was then applied to the point cloud, and the cuboids. The results can be seen in Figure 4.

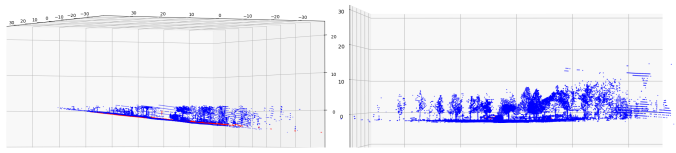
(a) Before correction (b) After correction

Fig. 4: Ground plane skew correction.

C. *Dataloader for Drone Data*

We developed a customized data loader to handle the preprocessed drone dataset. Upon initialization, the loader imports all preprocessed lidar scans, ground truth bounding boxes, and drone odometry in world coordinates. The ground truth bounding boxes are in the form $(x, y, z, qx, qy, qz, qw)$, which we transform into $(x, y, z, h, w, l, \theta)$ as expected by the neural network. Here, h is the height of the bounding box, w is the width, l is the length, and θ is the yaw angle of the bounding box. We obtain θ by converting the quaternion to Euler angles and discarding the pitch and roll angles. This operation would make sense only with the ground plane alignment in our preprocessing pipeline. In this setting, all objects of interest (i.e. cars) lie on the $z = 0$ plane with zero pitch and roll angles. As we lack labeled bounding box sizes, we manually assign the size of each ground truth bounding box as the average size of a car in reality.

Then, we implemented a filter to ensure the uniqueness of each data sample in the preprocessed drone dataset. Specifically, we calculate the norm of the difference between the translation vectors of two consecutive frames and remove the second frame if the norm is smaller than a predefined threshold. This filter eliminates duplicate scenes during training, resulting in better generalization. When the norm of the difference between translation vectors is small, it indicates that the drone has not moved significantly between those two frames, resulting in similar lidar data.

Additionally, we apply data augmentation techniques to enrich the dataset and improve generalization. Common techniques for LiDAR-based 3D object detection include random flips, global scaling, etc [4]. Among these techniques, we select global rotation as the primary data augmentation technique. For each query to the data loader, we rotate all points around the upright yaw axis by an angle α drawn from a uniform distribution $U(-\pi, \pi)$. We also rotate each annotated bounding box such that the new bounding box has the form $(x', y', z', h, w, l, (\theta+\alpha) \bmod 2\pi)$. Here, x', y', and z' represent the new bounding box center after rotation. Other data augmentation approaches are left as future work.

D. *PointRCNN for Point Cloud 3D Detection*

PointRCNN is a state-of-the-art two-stage object detection model proposed in [3]. It is designed to detect and predict the 3D bounding boxes of objects of interest based on raw point cloud data. The original network structure was intended for use on autonomous ground vehicles and was evaluated on large autonomous driving datasets such as KITTI [5]. However, the differences between drone and ground vehicle data, such as large pitch and roll angles, make it challenging to utilize the network structure for drone-based applications. To address this, the data processing pipeline, as mentioned in previous sections, is employed to detect the ground plane and correct for skewed orientation by aligning it to the $z = 0$ plane. This pipeline bridges the gap between the two types of data and allows for training the 3D detection model using the modified network structure, which will be discussed below.

1) Bottom-up 3D proposal generation via point cloud segmentation: In the first stage of the PointRCNN network, a large number of initial 3D bounding box proposals are generated from the raw point cloud data. The process begins with the point cloud passing through the backbone network, which uses PointNet++ [6] to learn a discriminative point-wise representation that captures the feature of each point in the point cloud.

The point-wise features are then passed through the segmentation head, which assigns each point a semantic label of object or background. To handle the class imbalance issue, where the number of background points is typically much larger than the number of foreground points, the classic focal loss function is used.

Next, the point-wise features are passed to the box regression head, which predicts rudimentary bounding boxes from foreground points with the bin-based regression technique they proposed. To estimate the center location of an object, the surrounding area of each foreground point is divided into discrete bins along the X and Y axes. The search range S for each axis is set, and each 1D search range is split into uniform-length bins to represent different object centers (x, y). The loss function for estimating the object's position along the X and Y axis includes two components: one for bin classification along each axis and another for residual regression within the classified bin, which is used for further location refinement within the assigned bin. To estimate the orientation of an object, the full possible orientation range 2π is cut into n bins, and the bin classification loss and residual regression loss are calculated in a similar way as the object center terms.

Finally, non-maximum suppression (NMS) is applied to remove redundant proposals, and the remaining proposals are passed to the second stage of the network for further refinement. This bottom-up approach efficiently generates a large number of object proposals from the point cloud data, allowing the network to focus on refining the most relevant proposals in the subsequent stage.

2) Canonical 3D bounding box refinement: The second stage of the PointRCNN refines the 3D box predictions based on the box proposals produced by the first stage. The initial step in the box refinement stage is to transform the points associated with each box proposal into the canonical coordinate system. This involves setting the origin at the center of the box proposal and aligning the X-axis with the head direction of the proposal. By doing so, all box proposals containing a car would share a common feature in the canonical coordinate system and the variability caused by different orientations is minimized. This transformation

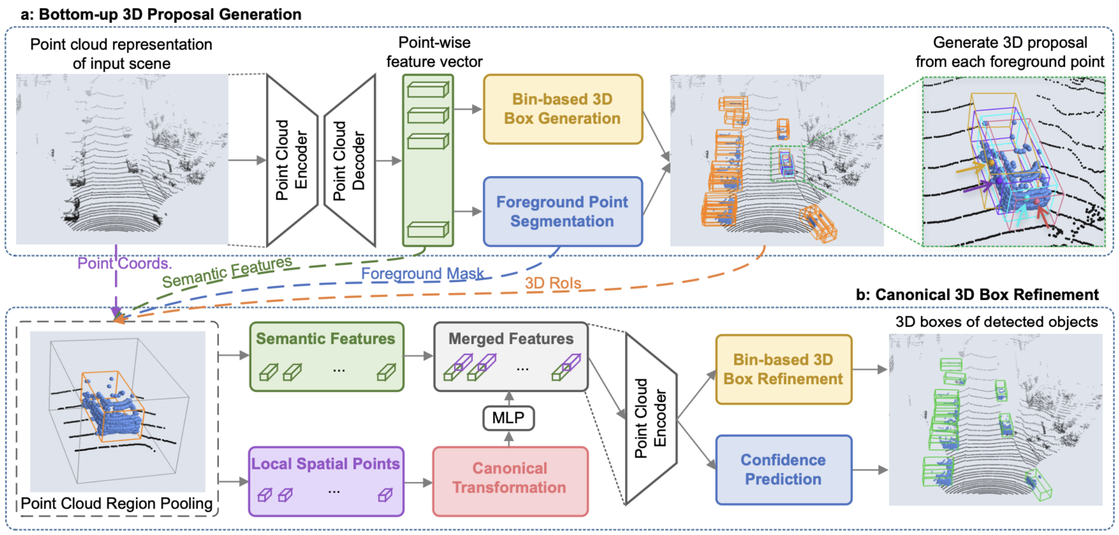
a: Bottom-up 3D Proposal Generation
Point cloud representation of input scene
Point-wise feature vector
Point Cloud Encoder
Point Cloud Decoder
Bin-based 3D Box Generation
Foreground Point Segmentation
Generate 3D proposal from each foreground point
Point Coords.
Semantic Features
Foreground Mask
3D RoIs
b: Canonical 3D Box Refinement
Point Cloud Region Pooling
Semantic Features
Merged Features
MLP
Local Spatial Points
Canonical Transformation
Point Cloud Encoder
Bin-based 3D Box Refinement
Confidence Prediction
3D boxes of detected objects

Fig. 5: The PointRCNN architecture for 3D object detection from point cloud [3]

helps the network to learn improved local spatial features for each proposal.

After transforming the points within each proposal to the canonical coordinate system, they are encoded again by the backbone network to get a local feature vector. This vector is then concatenated with the global feature vector obtained in the first stage and is used as the input for the following box refinement head. The same box regression technique as discussed in the previous section is used to predict the object center bin along each axis and the residual term for local refinement within the predicted bin. However, a smaller search range is used since the proposal from the first stage already provides a great starting point for refinement. Finally, NMS is applied again to remove the redundant proposals. Figure 5 from [3] provides a clear visualization of the PointRCNN network structure, the description above is a summary based on our understanding, and please refer to the original paper for more details.

V. RESULTS

For each frame, the model takes an input of ground truth cuboids and pointcloud, and outputs the predicted cuboids. Since we lack car dimensions in the labelled ground truth data, and there is only yaw (no roll or pitch) in the ground-truth data, we can evaluate our results by comparison of ground truth cuboids (x, y, z, θ) to the model outputs predicted cuboids (x', y', z', θ'). To evaluate model performance on a given sample, we use cosine similarity (Eqn 2) to determine the closest ground-truth cuboid for each predicted cuboid, then evaluates each predicted cuboid centroid compared to its ground-truth cuboid with a passing criteria of $SSE < 0.25$ and $\Delta\theta < 10°$ (Eqn 3).

$$\max_{j=1}^{N_g} \frac{\vec{p_i} \cdot \vec{g_j}}{\|\vec{p_i}\| \cdot \|\vec{g_j}\|} \tag{2}$$

$$Pred = \begin{cases} 1 & \text{if } (\hat{y}_i - y_i)^2 < 0.25 \text{ and } |\theta - \theta_i| < 10° \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Cosine similarity measures the similarity between two vectors of an inner product space and determines whether two vectors are pointing in roughly the same direction. $\boldsymbol{p}_i$ and $\boldsymbol{g}_j$ are the predicted and ground truth cuboid centroids, respectively, and Eqn 2 will give us a similarity score between -1 and 1. The sum of squared errors given in Eqn 3 denotes $\hat{y}_i$ as the predicted centroid for the ith cuboid and y_i as the corresponding ground truth centroid.

The results of this evaluation are shown in Table I, as are the precision, recall, and F1-score metrics. Note that the dataloader randomly samples points from each frame for the model. 1207 pointcloud frames were uploaded to the model, but due to stochasticity in the data uploader there are a differing number of ground-truth cuboids for the augmented and non-augmented data.

Dataset	# G.T.	# Pred	Precision	Recall	F1
Original	4087	2466	0.597	0.476	0.530
Augmented	4306	2923	0.406	0.462	0.432

TABLE I: Results with and without data augmentation

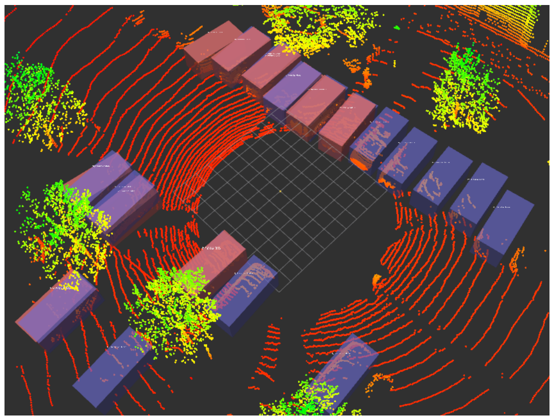

Fig. 6: Model output example from augmented dataset: ground-truth in blue (17 G.T. cuboids), predictions in red (8 true positive, 2 false positive)

The results indicate that the model is able to predict cuboids at a reasonable rate despite only having information from a single pointcloud scan. Given the difficult nature of this problem, the precision and recall metrics are quite promising in this limited sample result. Note that the significant decrease in precision between non-augmented and augmented data (0.406 vs 0.597) suggests the model is overfitting to the data in the non-augmented dataset, which shows the importance of our data augmentation pipeline in the dataloader as described in Section IV-C.

Figure 7 below shows an example of the effect of data augmentation. These pointclouds are sampled from the same data frame, but due to the random sampling of our dataloader, there is a significant difference between the augmented vs non-augmented datasets in predicted cuboids in precision (0.4 vs 1.0) and recall (0.33 vs 1.0).

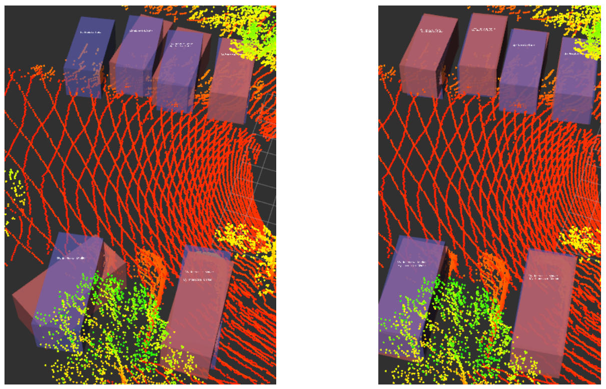

Fig. 7: Side-by-side comparison of the same frame (6 G.T. cuboids) with (2/5 predicted) and without augmentation (6/6 predicted).

VI. Discussion

Our initial results are quite promising as we have demonstrated this Deep Learning-based approach is able to predict bounding boxes for segmented car pointclouds even with a little as a single LiDAR scan. The current SLOAM [1] pipeline requires segmented points to be accumulated over several scans before it's able to fit a bounding box using geometric approaches with naive assumptions. If given more time, data, and/or resources, there are several improvements and extensions to this work that we would like to accomplish:

A. Larger training dataset and training resources

A major drawback of the current results is there is a lack of publicly available data to adequately train our model without overfitting to the data. Notably, we require quadrotor flight data with labelled and segmented LiDAR scans of cars in different environments, quantities, perspectives (roll/pitch/yaw of LiDAR wrt to cars), and diverse shapes of cars. For this project, we used previously labelled data taken by our colleagues in the Kumar lab of cars in the Pennovation and IKEA parking lots. However, as their bounding box methods were geometric-based and not learning-based, they only needed enough data for car segmentation over accumulated scans, whereas our model needs much more data of single-shot LiDAR scans of cars. Thus, the next step would be to collect flight data in different diverse environments and to use our labelling and training pipelines to improve our model performance.

Addtionally, the model training time was significant. The first stage is trained for 200 epochs with batch size 16, which takes 4 hrs. The second stage is trained for 70 epochs with batch size 4, taking another 4 hrs. We trained locally on System76 Desktop with Nvidia A5000 GPU. We were unable to optimize the hyperparameters (learning rate, bin size, etc.) of the RCNN network as we were only able to tune hyperparameters and train the model 3 times before the project deadline. Access to stronger GPU (or AWS credits) would allow us to refine and optimize the model hyperparameters.

B. Few-shot instead of single-shot LiDAR

This project only investigated the model accuracy when considering a single LiDAR scan. However, another model hyperparameter that could be implemented would be N frames, where N is the number of accumulated frames the model would consider during training and evaluation. We would need to tune this hyperparameter so that it improves model accuracy, without causing the model to use too much compute (storing too many scans at once) or becoming slower than other methods that require accumulated scans. As noted above, improvements in GPU training resources are required to tuned these type of hyperparameters, but this is an interesting improvement that could be optimized.

C. Running RCNN model on live quadrotor for real-time bounding box prediction

After the model has been refined through the methods described above, we would like to set up flight experiments on the Falcon 4 quadrotor [1] platform for real-time bounding box prediction. First, we would need to ensure the software dependencies are properly installed on the Ubuntu 18.04 that runs on the on-board Intel NUC 10. Additionally, we would need to verify the on-board i7-10710U processor is adequate to support our data pre-processing for converting the pointcloud from the quadrotor frame to the frame required by the PointRCNN model (methods described in IV-B) and to run the PointRCNN model live. If our model is able to predict car bounding boxes with accuracy, precision, and speed (no accumulation of frames required), we believe we can demonstrate a novel and meaningful experimental result that can be published in a journal paper. Furthermore, we can extend this work to bounding box prediction for other interesting objects and obstacles for quadrotors, such as lightpoles, trees, and buildings.

References

[1] X. Liu, G. V. Nardari, F. C. Ojeda, Y. Tao, A. Zhou, T. Donnelly, C. Qu, S. W. Chen, R. A. F. Romero, C. J. Taylor, and V. R. Kumar, "Large-Scale Autonomous Flight With Real-Time Semantic SLAM Under Dense Forest Canopy," *IEEE Robotics and Automation Letters*, vol. 7, pp. 5512–5519, 2021.

[2] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, “Rangenet ++: Fast and accurate lidar semantic segmentation,” pp. 4213–4220, 11 2019.

[3] S. Shi, X. Wang, and H. Li, “PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud,” 2019.

[4] M. Hahner, D. Dai, A. Liniger, and L. V. Gool, “Quantifying Data Augmentation for LiDAR based 3D Object Detection,” 2022.

[5] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research (IJRR)*, 2013.

[6] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,” 2017.