E(n) Equivariant Graph Neural Network for 3D Object Detection in a Point Cloud

Xiujin Liu

Abstract— **In this paper, we proposed a innovative E(n) Equivariant Graph Neural Network to detect objects from a LiDAR point cloud. Unlike images which are represented in regular dense grids, 3D point clouds are irregular and unordered, hence extracting information on them can be difficult. In paper, we aim to project 3D point cloud onto a translation-invariant and permutation-invariant homogeneous space.[1] After we encode the point cloud efficiently in a fixed radius near neighbors graph, by using the translation-invariant and rotation-invariant attributes of 3D objects, We design a Equivariant Graph Neural Network to predict the category and location of the object by predicting which category that each vertex in the graph belongs to based on previous work Point-GNN.[2] the code is published at https://github.com/XiujinLiu/computational-symmetry-in-AI-Robotics**

I. INTRODUCTION

Understanding and analyzing 3D scene is a crucial cornerstone for different tasks in robotics and computer vision. Detecting 3D rigid objects from a 3D scene is one of the most important task in understanding 3D scene. Based on the translation-invariant and rotation-invariant attributes of 3D rigid objects. We could apply Equivariant Deep Learning methods[3], [4] to increase the computation efficiency and accuracy. However, since 3D environment is often represented irregular and unordered 3D point clouds obtained by LiDAR. While point cloud could accurately and precisely present the scene, it's still hard to pass the information to Equivariant Convolutional Neural Networks and use translation and rotation symmetry to reduce the sample complexity when we extracting features and implementing detections. In this paper, we try to project 3D point cloud onto a translation-invariant and permutation-invariant homogeneous space.[1], and implementing a unique Equivariant Graph Neural Network to detect the objects. our Neural Network aims to take point clouds as input, transform it into homogeneous space, extract the edges and vertex of graphs from projected point clouds, then use two auto-registration layer and one EGNN layer to update the the features of vertices and edges, and the output is class type of the objects and bounding box.

*This work was not supported by any organization

[1]Albert Author is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands albert.author@papercept.net

[2]Bernard D. Researcheris with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA b.d.researcher@ieee.org

II. RELATED WORK

A. Equivariant Neural Networks

[4] introduces Group equivariant Convolutional Neural Networks (G-CNNs), a natural generalization of convolutional neural networks that reduces sample complexity by exploiting symmetries. PointConv[1] presents a novel convolution operation for 3D point clouds that utilizes learned weight and density functions, enabling effective deep learning on irregular data. It achieves state-of-the-art results in semantic segmentation and matches the performance of 2D convolutional networks on point cloud representations of 2D datasets. EGNN[5] introduces a new model to learn graph neural networks equivariant to rotations, translations, reflections and permutations called E(n)-Equivariant Graph Neural Networks (EGNNs).

B. GNN on Point Cloud

GCNs[6] presents a scalable approach for semi-supervised learning on graph-structured data that is based on an efficient variant of convolutional neural networks which operate directly on graphs. GraphSAGE[7] introduces a general, inductive framework that leverages node feature information (e.g., text attributes) to efficiently generate node embeddings. Instead of training individual embeddings for each node, GraphSAGE learns a function that generates embeddings by sampling and aggregating features from a node's local neighborhood. GATs[8] presents a novel neural network architectures that operate on graph-structured data, leveraging masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions or their approximations. Point-GNN[2] proposes a graph neural network to detect objects from a LiDAR point cloud, whhich present an auto-registration mechanism to reduce translation variance, and also design a box merging and scoring operation to combine detections from multiple vertices accurately.

C. Gaps

However, gaps still remain. First of all, even though existing methods for processing LiDAR point cloud could guarantee translation invariance, they still cannot conduct translation invariance, which is computational efficiency, and avoiding computationally expensive higher-order representations in intermediate layers while it still achieves competitive or better performance.

Our method try to introduce a new model to learn graph neural networks equivariant to rotations, translations, reflections and permutations for object detection and localization on point cloud

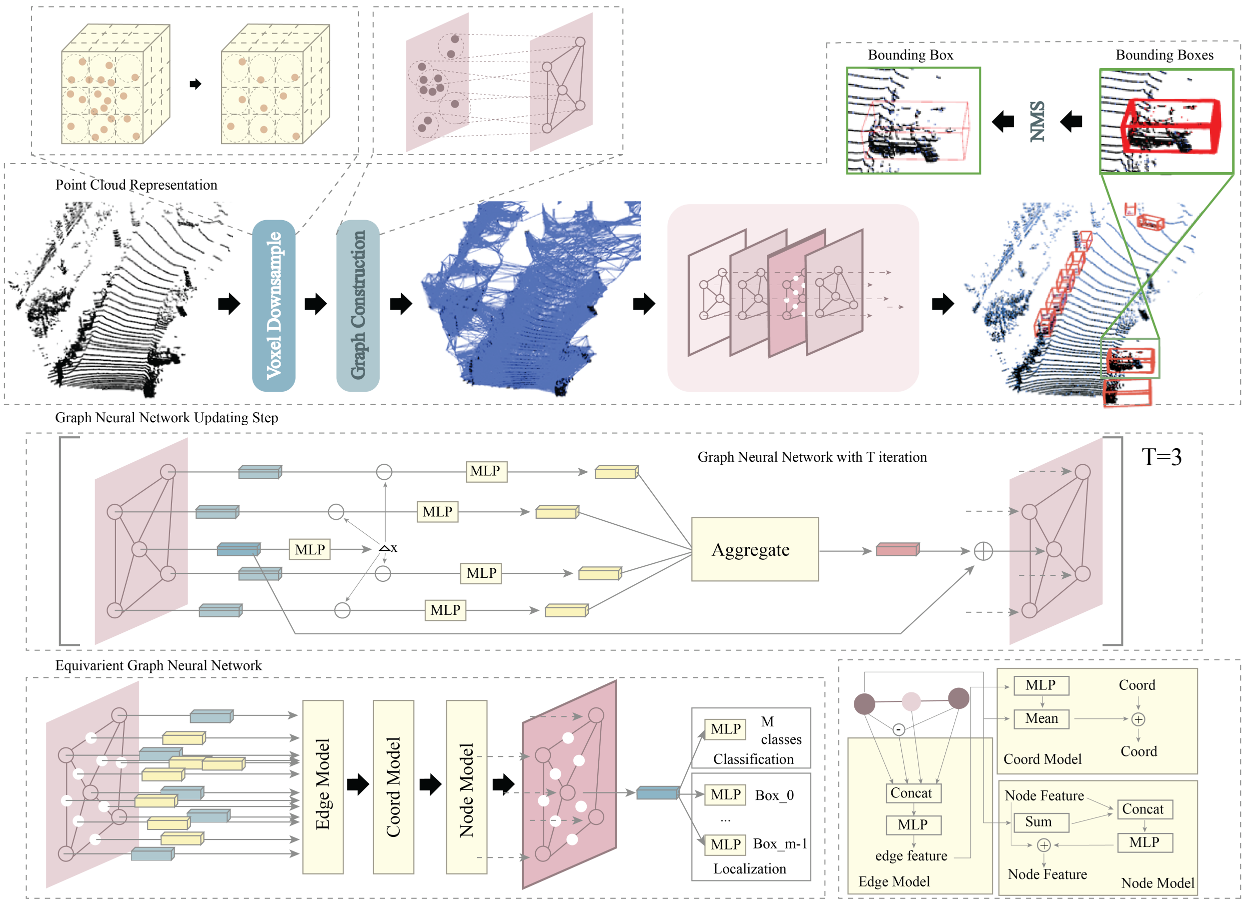

Fig. 1. Overall Pipeline for E(n) Equivariant Graph Neural Network for 3D Object Detection in a Point Cloud

III. METHODOLOGY

A. Overall Pipeline

Fig.1 depicts the structure. we use an encoder-decoder structure combined with graph neural network to extract feature and predict. Before Encoder, we use the closest 256 neighbors to form a graph for each point. Encoder as a neck consists of MLP layers to extract features, two auto-registration layer and one EGNN layer to update feature for both edges and vertices. The decoder layer outputs the final bounding box.

B. Data Processing

First of all, a point cloud commonly comprises tens of thousands of points. Constructing a graph with all the points as vertices imposes a substantial computational burden. Therefore, Shown as Fig.2, we use a voxel downsampled point cloud for the graph construction. For graph construction, as shown in Fig.5, we define a point cloud of N points as a set $P = p1, ..., pN$, where $pi = (xi, si)$ is a point with both 3D coordinates $xi \in R^3$ and the state value $si \in R^k$ is a klength vector that represents the point property. In our project, we project the point cloud onto RGB image frame, to get the color properties of each points. Given a point, we construct a graph $G = (P, E)$ by using P as the vertices and connecting a point to its neighbors within a fixed radius r, in our project $r = 256$.

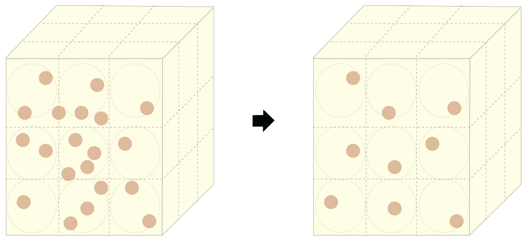

Fig. 2. Voxel Downsample

C. Auto-Registration layer and E(n) Equivariant Graph convolutional layer

we use sandwich structure to refines the vertex and edge features. The sandwich structure consists of two Auto-Registration layer layers at the head and tail, with a E(n) Equivariant Graph convolutional layer in the middle.

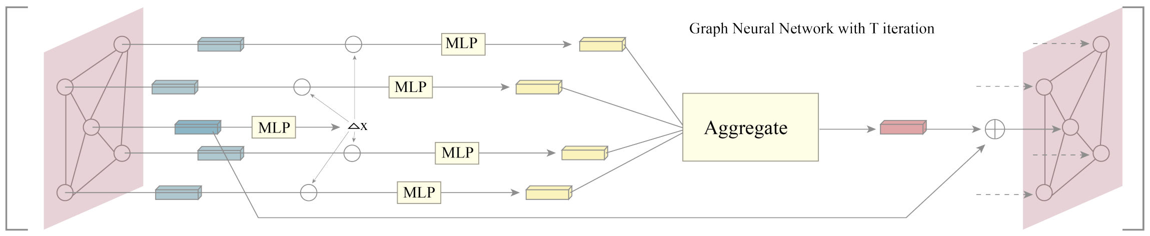

Fig. 3. Auto Registration Layer

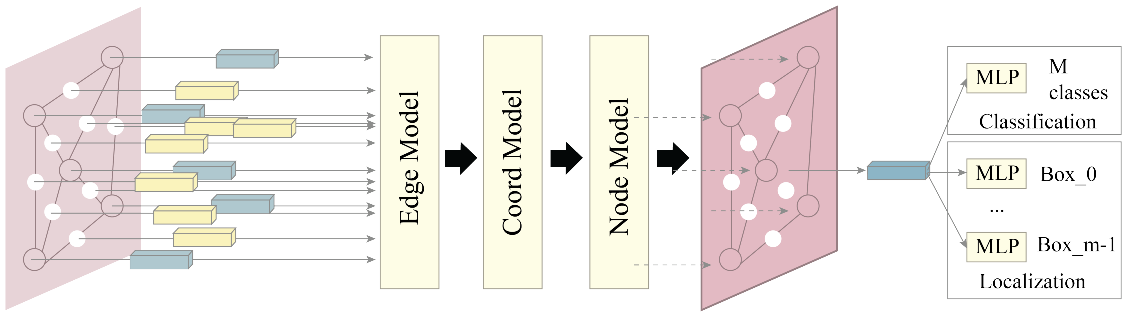

Fig. 4. E(n) Equivariant Graph convolutional layer

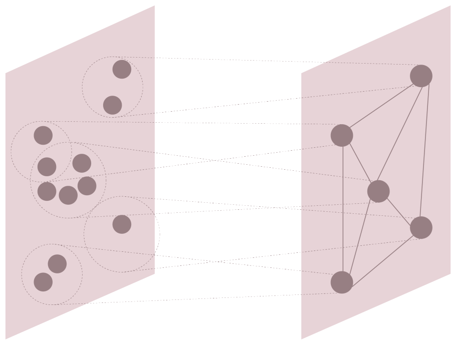

Fig. 5. Graph Construction

1) Auto-Registration layer: The first Auto-Registration layer is used to extract edge features, and the last Auto-Registration layer is used to update vertex features using edge features and vertex features produced by E(n) Equivariant Graph convolutional layer. As an auto-registration mechanism, we can also use it to predict an alignment offset, because the center vertex already contains some structural features from the previous iteration.

As shown in Fig. 3, In the (t+1)th iteration, it updates each vertex feature in the form 1:

$$\begin{aligned} & s_i^{t+1} = g^t(\rho(f^t(x_j - x_i, s_j^t)|(i,j) \in E), s_i^t) \\ & \Delta x_i^t = h^t(s_i^t) \\ & s^t + 1_i = g^t(\rho(f(x_j - x_i + \Delta x_i^t, s_j^t), s_i^t) \end{aligned} \tag{1}$$

where $f^t(.)$ computes the edge feature between two vertices. $\rho(.)$ is a set function which aggregates the edge features for each vertex. $g^t(.)$ takes the aggregated edge features to update the vertex features.Δx_i^t is the coordination offset for the vertices to register their coordinates. $h^t(.)$ calculates the offset using the center vertex state value from the previous iteration. By setting $h^t(.)$ to output zero, the GNN can disable the offset if necessary.

2) E(n) Equivariant Graph convolutional layer: Fig. **??** shows the workflow of E(n) Equivariant Graph convolutional layer. EGCL layer is a new model to learn graph neural networks equivariant to rotations, translations, reflections and permutations.

In this layer, we consider a graph $G = (V, E)$ with nodes $v_i \in V$ and edges $e_ij \in E$. In addition to the feature node embeddings $h_i \in R^{nf}$, we now also consider a n-dimensional coordinate $xi \in R^n$ associated with each of the graph nodes. This layer will preserve equivariance to rotations and translations on these set of coordinates x_i and it will also preserve equivariance to permutations on the set of nodes V in the same fashion as GNNs.

Sepcifically, Equivariant Graph Convolutional Layer (EGCL) takes as input the set of node embeddings $h^l = h_0^l, ..., h_{M-1}^l$, which is extracted from node properties $s_i \in R^k$, coordinate embeddings $x^l = x_0^l, ..., x_{M-1}^l$, which is node coordinates $pi \in R^3$ and edge information $E = e_{ij}$, which is obtained from Auto-Registration layer. And outputs a transformation on h^{l+1} and x^{l+1}. Concisely: $h^{l+1}, x^{l+1} = EGCL[h^l, x^l, E]$. The equations that define this layer are in the form 3, and the implementation detail are shown in Fig. 6.

$$\begin{aligned}
m_{ij} &= \phi_e(h_i^l, h_j^l, ||x_i^l - x_j^l||^2, a_{ij}) \\
x_i^{l+1} &= x_i^l + C\sum_{j\neq i}(x_i^l - x_j^l)\phi_x(m_{ij}) \\
m_i &= \sum_{i\neq j} m_{ij} \\
h^l + 1_i &= \phi_h(h_i^l, m_i)
\end{aligned} \tag{2}$$

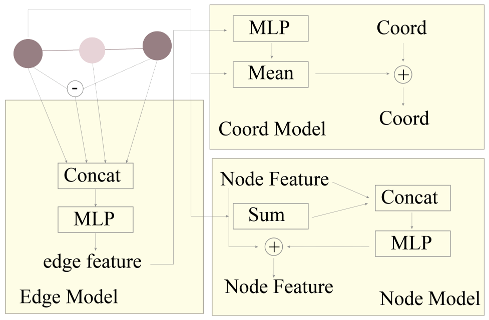

Fig. 6. Implementation Detail for EGCL

D. Loss

For the weights, we conduct three components.

For the object category, we use a multi-class probability distribution $p_{c1}, ..., p_{cM}$ for each vertex. M is the total number of object classes, including the Background class. If a vertex is within a bounding box of an object, we assign the object class to the vertex. If a vertex is outside any bounding boxes, we assign the background class to it. We use the average cross-entropy loss as the classification loss.

$$l_{cls} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1} My_{cj}^i log(p_{cj}^i) \tag{3}$$

For the object bounding box, we predict it in the 7 degree-of-freedom format $b = (x, y, z, l, h, w, \theta)$, where (x, y, z) represent the center position of the bounding box, (l, h, w) represent the box length, height and width respectively, and θ is the yaw angle. We encode the bounding box with the vertex coordinates (x_v, y_v, z_v) as follows:

$$\begin{aligned}
&\phi x = \frac{x - x_v}{l_m}, \phi y = \frac{y - y_v}{h_m}, \phi x = \frac{z - z_v}{w_m} \\
&\phi_l = log(\frac{l}{l_m}), \phi_h = log(\frac{h}{h_m}), \phi_w = log(\frac{w}{w_m}) \\
&\phi_{theta} = \frac{\theta - \theta_0}{\theta_m}
\end{aligned} \tag{4}$$

The last components predicts the encoded bounding box $\phi_b = (\phi_x, \phi_y, \phi_z, \phi_l, \phi_h, \phi_w, \phi_\theta)$ for each class. If a vertex is within a bounding box, we compute the Huber loss between the ground truth and our prediction. If a vertex is outside any bounding boxes or it belongs to a class that we do not need to localize, we set its localization loss as zero. We then average the localization loss of all the vertices:

$$l_{loc} = \frac{1}{N}\sum_{i=1}^{N}(v_i \in b_{interest}) \sum_{\phi\in\phi_{bi}} l_{huber}(\phi - \phi^{gt}) \tag{5}$$

Finally, the total loss is $total = l_{cls} + l_{loc} + l_{reg}$.

IV. CONCLUSIONS

In conclusion, in this project, we present a potential work about using Equivariant Graph Neural Network to solve 3D obejcts detection problem in point clouds by using translation and rotation symmetry.

REFERENCES

[1] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, 2019, pp. 9621–9630.
[2] W. Shi and R. Rajkumar, "Point-gnn: Graph neural network for 3d object detection in a point cloud," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1711–1719.
[3] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International conference on machine learning*. PMLR, 2016, pp. 2990–2999.
[4] T. Cohen *et al.*, "Equivariant convolutional networks," Ph.D. dissertation, Taco Cohen, 2021.
[5] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," in *International conference on machine learning*. PMLR, 2021, pp. 9323–9332.
[6] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[7] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.
[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.