*APSIPA Transactions on Signal and Information Processing, 2023, 12, e103*


**Original Paper**

# PointFlowHop: Green and Interpretable Scene Flow Estimation from Consecutive Point Clouds

Pranav Kadam[1*], Jiahao Gu[1], Shan Liu[2] and C.-C. Jay Kuo[1]

[1] *University of Southern California, Los Angeles, CA, USA*
[2] *Tencent Media Lab, Tencent America, Palo Alto, CA, USA*

ABSTRACT

An efficient 3D scene flow estimation method called PointFlowHop is proposed in this work. PointFlowHop takes two consecutive point clouds and determines the 3D flow vectors for every point in the first point cloud. PointFlowHop decomposes the scene flow estimation task into a set of subtasks, including ego-motion compensation, object association and object-wise motion estimation. It follows the green learning (GL) pipeline and adopts the feedforward data processing path. As a result, its underlying mechanism is more transparent than deep-learning (DL) solutions based on end-to-end optimization of network parameters. We conduct experiments on the stereoKITTI and the Argoverse LiDAR point cloud datasets and demonstrate that PointFlowHop outperforms deep-learning methods with a small model size and less training time. Furthermore, we compare the Floating Point Operations (FLOPs) required by PointFlowHop and other learning-based methods in inference, and show its big savings in computational complexity.

*Keywords:* 3D scene flow estimation, green learning, unsupervised learning, PointHop.

*Corresponding author: Pranav Kadam, pranavka@usc.edu.
GitHub: https://github.com/pranavkdm/PointFlowHop

Received 27 February 2023; Revised 20 June 2023
ISSN 2048-7703; DOI 10.1561/116.00000006

## 1 Introduction

Dynamic 3D scene understanding based on captured 3D point cloud data is a critical enabling technology in the 3D vision systems. 3D scene flow aims at finding the point-wise 3D displacement between consecutive point cloud scans. With the increase in the availability of point cloud data, especially those acquired via the LiDAR scanner, 3D scene flow estimation directly from point clouds is an active research topic nowadays. 3D scene flow estimation finds rich applications in 3D perception tasks such as semantic segmentation, action recognition, and inter-prediction in compressing sequences of LiDAR scans.

Today's solutions to 3D scene flow estimation mostly rely on supervised or self-supervised deep neural networks (DNNs) that learn to predict the point-wise motion field from a pair of input point clouds via end-to-end optimization. One of the important components of these methods is to learn flow embedding by analyzing spatio-temporal correlations among regions of the two point clouds. After the successful demonstration of such an approach in FlowNet3D [25], there has been an increased number of papers on this topic by exploiting and combining other ideas such as point convolutions and attention mechanism.

These DNN-based methods work well in an environment that meets the local scene rigidity assumption. They usually outperform classical point-correspondence-based methods. On the other hand, they have a large number of parameters and rely on large training datasets. For the 3D scene flow estimation problem, it is non-trivial to obtain dense point-level flow annotations. Thus, it is challenging to adopt the heavily supervised learning paradigm with the real world data. Instead, methods are typically trained on synthetic datasets with ground truth flow information first. They are later fine-tuned for real world datasets. This makes the training process very complicated.

In this paper, we develop a green and interpretable 3D scene flow estimation method for the autonomous driving scenario and name it "PointFlowHop". We decompose our solution into vehicle ego-motion and object motion modules. Scene points are classified as static and moving. Moving points are grouped into moving objects and a rigid flow model is established for each object. Furthermore, the flow in local regions is refined assuming local scene rigidity. PointFlowHop method adopts the green learning (GL) paradigm [20]. It is built upon related recent work, GreenPCO [15], and preceding foundation works such as R-PointHop [16], PointHop [48], and PointHop++ [47].

The task-agnostic nature of the feature learning process in prior art enables scene flow estimation through seamless modification and extension. Furthermore, a large number of operations in PointFlowHop are not performed during training. The ego-motion and object-level motion is optimized in inference only. Similarly, the moving points are grouped into objects only during inference. This makes the training process much faster and the model size very small.

The decomposition of 3D scene flow into object-wise rigid motion and/or ego-motion components is not entirely novel. However, our focus remains in developing a GL-based solution with improved overall performance, including accuracy, model sizes and computational complexity.

The novelty of our work lies in two aspects. First, it expands the scope of existing GL-based point cloud data processing techniques. GL-based point cloud processing has so far been developed for object-level understanding [14, 17, 18, 46–48] and indoor scene understanding [16, 45]. This work addresses the more challenging problem of outdoor scene understanding at the point level. This work also expands the application scenario of R-PointHop, where all points are transformed using one single rigid transformation. For 3D scene flow estimation, each point has its own unique flow vector. Furthermore, we show that a single model can learn features for ego-motion estimation as well as object-motion estimation, which are two different but related tasks. This allows model sharing and opens doors to related tasks such as joint scene flow estimation and semantic segmentation. Second, our work highlights the over-paramertized nature of DL-based solutions which demand larger model sizes and higher computational complexity in both training and testing. The overall performance of PointFlowHop suggests a new point cloud processing pipeline that is extremely lightweight and mathematically transparent.

To summarize, there are three major contributions of this work.

- We develop a lightweight 3D scene classifier that identifies moving points and further clusters and associates them into moving object pairs.
- We optimize the vehicle ego-motion and object-wise motion based on point features learned using a single task-agnostic feedforward PointHop++ model.
- PointFlowHop outperforms representative benchmark methods in the scene flow estimation task on two real world LiDAR datasets with fewer model parameters and lower computational complexity measured by FLOPs (floating-point operations).

The rest of the paper is organized as follows. Related work is reviewed in Section 2. The PointFlowHop method is proposed in Section 3. Experimental results are presented in Section 4. Finally, concluding remarks and possible future extensions are given in Section 5.

## 2 Related Work

### *2.1 Scene Flow Estimation*

Early work on 3D scene flow estimation uses 2D optical flow estimation followed by triangulation such as that given in [37]. The Iterative Closest Point (ICP)

[5] and the non-rigid registration work, NICP [1], can operate on point clouds directly. A series of image- and point-based seminal methods for scene flow estimation relying on similar ideas were proposed in the last two decades. The optical flow is combined with dense stereo matching for flow estimation in [13]. A variational framework that predicts the scene flow and depth is proposed in [2]. A piecewise rigid scene flow estimation method is investigated in [39]. Similarly, the motion of rigidly moving 3D objects is examined in [27]. Scene flow based on Lucas-Kanade tracking [26] is studied in [34]. An exhaustive survey on 2D optical flow and 3D scene flow estimation methods has been done by Zhai *et al.* [43]. We adopt the object-level rigid motion analysis as presented in [27] and several related follow-up works. However, their problem formulation and optimization is different from ours and they do use training data to learn features.

Deep-learning-based (DL-based) methods have been popular in the field of computer vision in the last decade. For DL-based 3D scene flow estimation, FlowNet3D [25] adopts the feature learning operations from PointNet++ [33]. HPLFlowNet [11] uses bilateral convolution layers and projects point clouds to an ordered permutohedral lattice. PointPWC-Net [41] takes a self-supervised learning approach that works in a coarse-to-fine manner. FLOT [32] adopts a correspondence-based approach based on optimal transport. HALFlow [40] uses a hierarchical network structure with an attention mechanism. The Just-Go-With-the-Flow method [30] uses self-supervised learning with the nearest neighbor loss and the cycle consistency loss. These DL-based methods do not decompose the scene flow into ego-motion and object-level rigid motion like ours.

DL-based methods that attempt to simplify the flow estimation problem using ego-motion and/or object-level motion have also been investigated. For example, Rigid3DSceneFlow [10] reasons the scene flow at the object level (rather than the point level). Accordingly, the flow of scene background is analyzed via ego-motion and that of a foreground object is described by a rigid model. RigidFlow [22] enforces the rigidity constraint in local regions and performs rigid alignment in each region to produce rigid pseudo flow. SLIM [3] uses a self-supervised loss function to separate moving and stationary points. However, these methods still require end-to-end training, unlike ours where the feature is learned in a feedforward manner.

### *2.2 Green Point Cloud Learning*

Green Learning (GL) [20] has started to gain attention as an alternative to Deep Learning (DL) in recent years. Typically, GL consists of three modules: 1) unsupervised representation learning, 2) supervised feature learning, and 3) supervised decision learning. The unsupervised representation learning in the first module is rooted in the derivation of data-driven transforms

such as the Saak [19] and the Saab [21] transforms. Both the training and inference processes in GL adopt a feedforward data processing path without backpropagation. The optimization is statistics-based, and it is carried out at each module independently. The learning process is lightweight, making it data and computation resource friendly. GL-based methods have been developed for a wide variety of image processing and computer vision tasks [20].

Green Point Cloud learning [24] was first introduced in PointHop [48]. The unsupervised representation learning process involves constructing a local point descriptor via octant space partitioning followed by dimensionality reduction via the Saab transform. These operations together are called one PointHop unit. It is the fundamental building block in a series of follow-up works along with other task-specific modules. PointHop++ [47] replaces the Saab transform with its efficient counterpart called the Channel-wise Saab transform [8]. We use PointHop++ for learning point-wise features in the ego-motion and object motion estimation steps. PointHop and PointHop++ adopt a multi-hop learning system for point cloud classification, whereby the learned point representations are aggregated into a global feature vector and fed to a classifier. The multi-hop learning architecture is analogous to the hierarchical deep learning architecture. The multi-hop architecture helps capture the information from short-, mid-, and long-range point cloud neighborhoods.

More recently, R-PointHop [16], GSIP [45] and GreenPCO [15] demonstrate green learning capabilities on more challenging large-scale point clouds for indoor scene registration, indoor segmentation, and odometry tasks, respectively. R-PointHop finds corresponding points between the source and target point clouds using the learned representations and then estimates the 3D rotation and translation to align the source with the target. We use this procedure in the object motion estimation step in PointFlowHop, where the 3D rotation and translation gives the object rigid motion model. In GreenPCO, a similar process is adopted to incrementally estimate the object's trajectory. Additional ideas presented in GreenPCO include a geometry-aware point cloud sampling scheme that is suitable for LiDAR data. We use GreenPCO in the ego-motion compensation step. Other noteworthy green point cloud learning works include SPA [17], UFF [46], PCRP [18], and S3I-PointHop [14]. While these works mainly focus on object-level or indoor-scene analysis, PointFlowHop is an application of green learning to outdoor scene analysis.

## 3 Proposed PointFlowHop Method

The system diagram of the proposed PointFlowHop method is shown in Figure 1. It takes two consecutive point clouds $X_t \in \mathbb{R}^{n_t \times 3}$ and $X_t \in \mathbb{R}^{n_{t+1} \times 3}$ as the input and calculates the point-wise flow $\bar{f}_t \in \mathbb{R}^{n_1 \times 3}$ for the points in $X_t$.

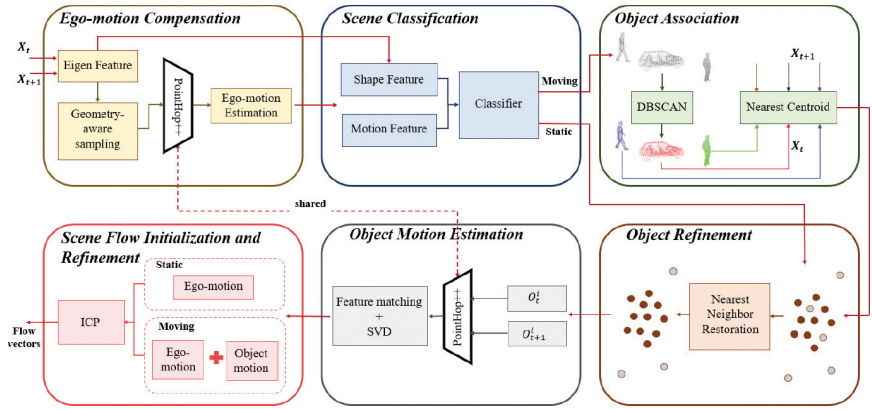


Figure 1: An overview of the PointFlowHop method, which consists of six modules: 1) ego-motion compensation, 2) scene classification, 3) object association, 4) object refinement, 5) object motion estimation, and 6) scene flow initialization and refinement.

PointFlowHop decomposes the scene flow estimation problem into two subproblems: 1) determining vehicle's ego-motion ($T_{ego}$) and 2) estimating the motion of each individual object (denoted by ($T_i$) for object $i$). It first proceeds by determining and compensating the ego-motion and classifying scene points as being moving or static in modules 1 and 2, respectively. Next, moving points are clustered and associated as moving objects in modules 3 and 4, and the motion of each object is estimated in module 5. Finally, the flow vectors of static and moving points are jointly refined. These steps are detailed below.

### *3.1 Module 1: Ego-motion Compensation*

The $i^{th}$ point in $X_t$ has coordinates $(x_t^i, y_t^i, z_t^i)$. Suppose this point is observed at $(x_{t+1}^i, y_{t+1}^i, z_{t+1}^i)$ in $X_{t+1}$. These point coordinates are expressed in the respective LiDAR coordinate systems centered at the vehicle position at time $t$ and $t+1$. Since the two coordinate systems may not overlap due to vehicle's motion, the scene flow vector, $\bar{f}_t^{\,i}$, of the $i^{th}$ point cannot be simply calculated using vector difference. Hence, we begin by aligning the two coordinates systems or, in other words, we compensate for the vehicle motion (or called ego-motion).

The ego-motion compensation module in PointFlowHop is built upon a recently proposed point cloud odometry estimation method, called GreenPCO [15]. It is briefly reviewed below for self-containedness. GreenPCO determines the vehicle trajectory incrementally by analyzing consecutive point cloud scans. It is conducted with the following four steps. Usually, the point clouds have

a large number of points and not all points are required, nor useful in the ego-motion compensation step. Uniformly downsampling the point cloud using iterative farthest point sampling is not useful since it selects some featureless points. Hence, first, the two point clouds are sampled using the geometry-aware sampling method instead, which selects points spatially spread out with salient local surfaces. Geometry-aware sampling considers two criteria jointly in selecting 2048 discriminant points for ego-motion estimation – point saliency based on the local geometric eigen feature [12] and spatial distance between discriminant points. Second, the sampled points from the two point clouds are divided into four views - front, left, right and rear based on the azimuthal angles. Third, point features are extracted using PointHop++ [47]. The features are used to find matching points between the two point clouds in each view. Last, the pairs of matched points are used to estimate the vehicle trajectory. These steps are repeated as the vehicle advances in the environment. The diagram of the GreenPCO method is depicted in Figure 2.

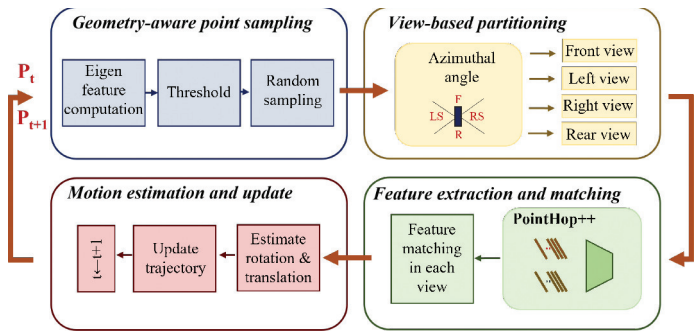


Figure 2: An overview of the GreenPCO method [15].

Ego-motion estimation in PointFlowHop involves a single iteration of GreenPCO whereby the vehicle's motion from time $t$ to $t+1$ is estimated. Then, the ego-motion can be represented by the 3D transformation, $T_{ego}$, which consists of a 3D rotation and 3D translation. Afterward, we use $T_{ego}$ to warp $X_t$ to $\tilde{X}_t$, making it in the same coordinate system as that of $X_{t+1}$. Then, the flow vector can be computed by

$$\bar{f}_t^{\,i} = (x_{t+1}^i - \tilde{x}_t^i, y_{t+1}^i - \tilde{y}_t^i, z_{t+1}^i - \tilde{z}_t^i), \tag{1}$$

where $(\tilde{x}_t^i, \tilde{y}_t^i, \tilde{z}_t^i)$ is the warped coordinate of the $i^{th}$ point.

### *3.2 Module 2: Scene Classification*

After compensating for ego-motion, the resulting $\tilde{X}_t$ and $X_{t+1}$ are in the same coordinate system (i.e., that of $X_{t+1}$). Next, we coarsely classify scene

points in $\tilde{X}_t$ and $X_{t+1}$ into moving and static two classes. Generally speaking, the moving points may belong to objects such as cars, pedestrians, mopeds, etc., while the static points correspond to objects like buildings, poles, etc. The scene flow of moving points can be analyzed later while static points can be assigned a zero flow (or equal to the ego-motion depending on the convention of the coordinate systems used). This means that the later stages of PointFlowHop would process fewer points.

For the scene classifier, we define a set of shape and motion features that are useful in distinguishing static and moving points. These features are explained below.

- Shape features

  We reuse the eigen features [12] calculated in the ego-motion estimation step. They summarize the distribution of neighborhood points using covariance analysis. The analysis provides a 4-dimensional feature vector comprising of linearity, planarity, eigen sum and eigen entropy.

- Motion feature

  We first voxelize $\tilde{X}_t$ and $X_{t+1}$ with a voxel size of 2 meters. Then, the 1-dimensional motion feature for each point in $\tilde{X}_t$ is the distance to the nearest voxel center in $X_{t+1}$, and vice versa, for each point in $X_{t+1}$.

The 5-dimensional (shape and motion) feature vector is fed to a binary XGBoost classifier. For training, we use the point-wise class labels provided by the SemanticKITTI [4] dataset. We observe that the 5D shape/motion feature vector are sufficient for decent classification. The classification accuracy on the SemanticKITTI dataset is 98.82%. Furthermore, some of the misclassified moving points are reclassified in the subsequent object refinement step.

### *3.3 Module 3: Object Association*

We simplify the problem of motion analysis on moving points by grouping moving points into moving objects. To discover objects from moving points, we use the Density-based Spatial Clustering for Applications with Noise (DBSCAN) [9] algorithm. Simply speaking, DBSCAN iteratively clusters points based on the minimum distance ($eps$) and the minimum points ($minPts$) parameters. Parameter $eps$ gives the minimum Euclidean distance between points considered as neighbors. Parameter $minPts$ determines the minimum number of points to form a cluster. Some examples of the objects discovered using PointFlowHop are colored in Figure 3.

Points belonging to distinct objects may get clustered together. We put the points marked as "outliers" by DBSCAN in the set of static points. The DBSCAN algorithm is run on $\tilde{X}_t$ and $X_{t+1}$ separately. Later, we use cluster

Figure 3: Objects clustered using the DBSCAN algorithm are shown in different colors.

centroids to associate objects between $\tilde{X}_t$ and $X_{t+1}$. That is, for each centroid in $\tilde{X}_t$, we locate its nearest centroid in $X_{t+1}$.

### *3.4 Module 4: Object Refinement*

Next, we perform an additional refinement step to recover some of the misclassified points during shape classification and potential inlier points during object association. This is done using the nearest neighbor rule within a defined radius neighborhood. For each point classified as a moving point, we re-classify static points lying within the neighborhood as moving points. The object refinement operation is conducted on $\tilde{X}_t$ and $X_{t+1}$.

The refinement step is essential for two reasons. First, an imbalance class distribution between static and moving points usually leads to the XGBoost classifier to favor the dominant class (which is the static points). Then, the precision and recall for moving points are still low in spite of high classification accuracy. Second, in the clustering step, it is difficult to select good values for *eps* and *minPts* that are robust in all scenarios for the sparse LiDAR point clouds. This may lead to some points being marked as outliers by DBSCAN. Overall, the performance gains of our method reported in Section 4 are a result of the combination of all steps and not due to a single step in particular.

### *3.5 Module 5: Object Motion Estimation*

We determine the motion between each pair of associated objects in this step. For that, we follow a similar approach as taken by a point cloud rigid registration method, R-PointHop [16]. The objective of R-PointHop is to register the source point cloud with the target point cloud. The block diagram of R-PointHop is illustrated in Figure 4. It includes the following two major steps. First, the source and target point clouds are fed to a sequence of R-PointHop units for hierarchical feature learning (or multiple hops) in the feature learning step. Point clouds are downsampled between two hops by iteratively selecting farther points. The R-PointHop unit comprises of constructing a

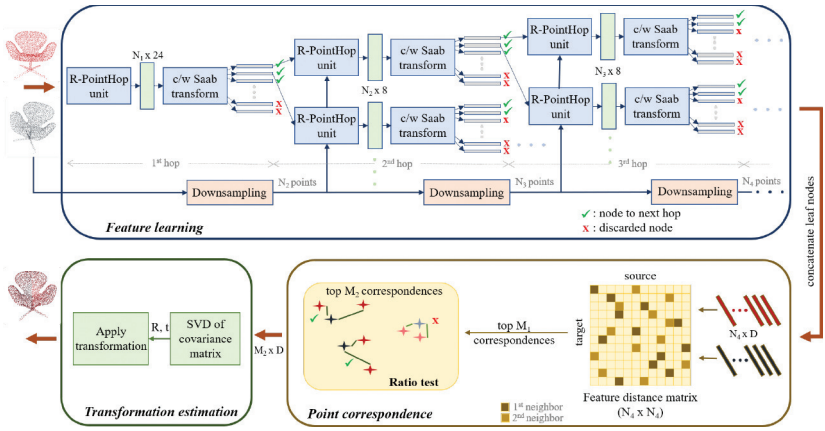


Figure 4: An overview of the R-PointHop method [16].

local point descriptor followed by the channel-wise Saab transform [8]. Second, the point features are used to find pairs of corresponding points. The optimal rigid transformation that aligns the two point clouds is then solved as an energy minimization problem [35].

For object motion estimation in PointFlowHop, the features of refined moving points from $\tilde{X}_t$ and $X_{t+1}$ are extracted using the trained PointHop++ model. We reuse the same model from the ego-motion estimation step here. While four hops with intermediate downsampling is used in R-PointHop, the PointHop++ model in PointFlowHop only involves two hops without downsampling to suit the LiDAR data. We use $O_t^i$ and $O_{t+1}^i$ to indicate sets of points belonging to object $i$. We find corresponding points between these two point clouds using the nearest neighbor search in the feature space. The correspondence set is further refined by selecting top correspondences based on: 1) the minimum feature distance criterion and 2) the ratio test (the minimum ratio of the distance between the first and second best corresponding points). The refined correspondence set is then used to estimate the object motion as follows.

First, the mean coordinates of the corresponding points in $\tilde{O}_t^i$ and $O_{t+1}^i$ are found by:

$$\bar{o}_t^i = \frac{1}{N_i}\sum_{j=1}^{N_i} \tilde{o}_t^{ij}, \quad \bar{o}_{t+1}^i = \frac{1}{N_i}\sum_{j=1}^{N_i} o_{t+1}^{ij}. \tag{2}$$

Then, the $3 \times 3$ covariance matrix is computed using the pairs of corresponding points as

$$K(\tilde{O}_t^i, O_{t+1}^i) = \sum_{j=1}^{N_i} (\tilde{o}_t^{ij} - \bar{o}_t^i)(o_{t+1}^{ij} - \bar{o}_{t+1}^i)^T. \tag{3}$$

The Singular Value Decomposition of $K$ gives matrices $U$ and $U$, which are formed by the left and right singular vectors, respectively. Mathematically, we have

$$K(\tilde{O}_t^i, O_{t+1}^i) = USV^T. \tag{4}$$

Following the orthogonal procrustes formulation [35], the optimal motion of $\tilde{O}_t^i$ can be expressed in form of a rotation matrix $R^i$ and a translational vector $t^i$. They can be computed as

$$R^i = VU^T, \quad t^i = -R^i \bar{o}_t^i + \bar{o}_{t+1}^i. \tag{5}$$

Since $(R^i, t^i)$ form the object motion model for object $i$, it is denoted as $T_i$.

Actually, once we find the corresponding point $o_{t+1}^{ij}$ of $\tilde{o}_t^{ij}$, the flow vector may be set to

$$f_t^{ij} = o_{t+1}^{ij} - \tilde{o}_t^{ij}.$$

However, this point-wise flow vector can be too noisy, and it is desired to use a flow model for the object rather than each point. The object flow model found using SVD in the step after finding correspondences is optimal in the mean square sense over all corresponding points and, hence, is more robust. It makes a reasonable assumption of existence of a rigid transformation between the two objects.

### *3.6 Module 6: Flow Initialization and Refinement*

In the last module, we apply the object motion model $T_i$ to $\tilde{O}_t^i$ and align it with $O_{t+1}^i$. Since the static points do not have any motion, they are not further transformed. We denote the new transformed point cloud as $\tilde{X}_t'$. At this point, we have obtained an initial estimate of the scene flow for each point in $X_t$. For static points, the flow is given by the ego-motion transformation $T_{ego}$. For the moving points, it is a composition of ego-motion and corresponding object's motion $T_{ego} \cdot T_i$.

In this module, we refine the flow for all points in $\tilde{X}_t'$ using the Iterative Closest Point (ICP) [5] algorithm in small non-overlapping regions. In each region, the points in $\tilde{X}_t'$ falling within it are aligned with corresponding points in $X_{t+1}$. The flow refinement step ensures a tighter alignment and is a common post processing operation in several related tasks. Finally, the flow vectors for $X_t$ are calculated as the difference between the transformed and initial coordinates. Exemplar pairs of input and scene flow compensated point clouds using PointFlowHop are shown in Figure 5.

It is worth noting that naive point-to-point ICP can be replaced with its variants such as point-to-plane ICP [7], the Generalized ICP [36], or similar local registration methods. However, global registration methods like Fast Global Registration (FGR) [49] or TEASER [42] may not be necessary for the refinement since the initial flow is already close to optimal.

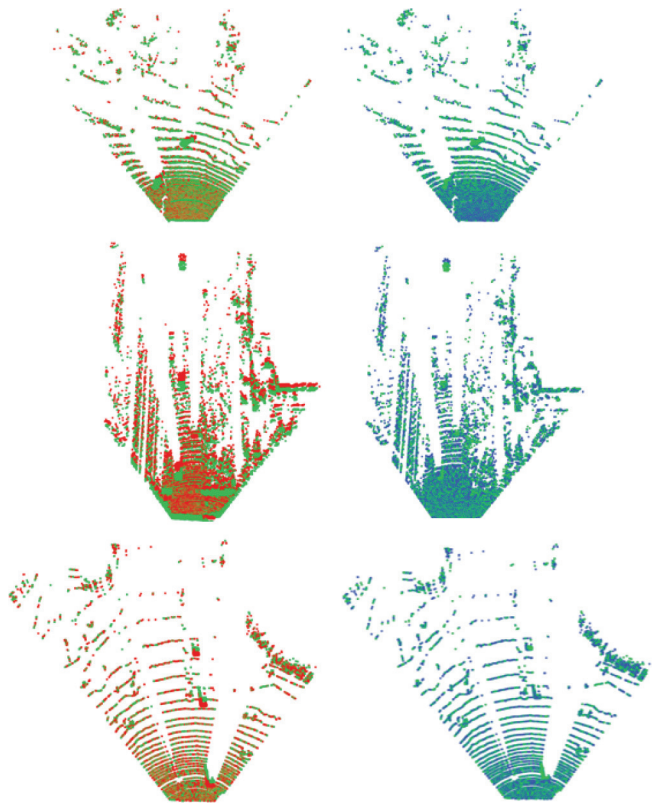

Figure 5: Flow estimation results using PointFlowHop: input point clouds (left) and warped output using flow vectors (right).

## 4 Experiments

In this section, we report experimental results on real world LiDAR point cloud datasets. We choose the stereoKITTI [28, 29] and the Argoverse [6] two datasets since they represent challenging scenes in autonomous driving environments. StereoKITTI has 142 pairs of point clouds. The ground truth flow of each pair is derived from the 2D disparity maps and the optical flow information. There are 212 test samples for Argoverse whose flow annotations were given in [31]. We use per-point labels from the SemanticKITTI dataset [4] to train our scene classifier.

Following a series of prior art, we measure the performance in the following metrics:

- *3D end point error (EPE3D).* It is the mean Euclidean distance between the estimated and the ground truth flow.

- *Strict accuracy (Acc3DS).* It is the percentage of points for which EPE3D is less than 0.05m or the relative error is less than 0.05.
- *Relaxed accuracy (Acc3DR).* It gives the ratio of points for which EPE3D is less than 0.1m or the relative error is less than 0.1.
- *Percentage of Outliers.* It is the ratio of points for which EPE3D is greater than 0.3m or the relative error is greater than 0.1. This is reported for the StereoKITTI dataset only.
- *Mean angle error (MAE).* It is the mean of the angle errors between the estimated and the ground truth flow of all points expressed in the unit of radians. This is reported for the Argoverse dataset only.

### *4.1 Performance Benchmarking*

The scene flow estimation results on stereoKITTI and Argoverse are reported in Table 1 and Table 2, respectively. For comparison, we show the performance of several representative methods proposed in the past few years. Overall, the EPE3D, Acc3DS and Acc3DR values are significantly better for stereoKITTI as compared to the Argoverse dataset. This is because Argoverse is a more challenging dataset. Furthermore, PointFlowHop outperforms all benchmarking methods in almost all evaluation metrics on both datasets.

### *4.2 Ablation Study*

In this section, we assess the role played by each individual module of PointFlowHop using the stereo KITTI dataset as an example.

**Ego-motion compensation.** First, we may replace GreenPCO [15] with ICP [5] for ego-motion compensation. The results are presented in Table 3. We see a sharp decline in performance with ICP. The substitution makes the new method much worse than all benchmarking methods. While the naive

Table 1: Comparison of scene flow estimation results on the Stereo KITTI dataset, where the best performance number is shown in boldface.

| Method | EPE3D (m)↓ | Acc3DS ↑ | Acc3DR ↑ | Outliers ↓ |
|---|---|---|---|---|
| FlowNet3D [25] | 0.177 | 0.374 | 0.668 | 0.527 |
| HPLFlowNet [11] | 0.117 | 0.478 | 0.778 | 0.410 |
| PointPWC-Net [41] | 0.069 | 0.728 | 0.888 | 0.265 |
| FLOT [32] | 0.056 | 0.755 | 0.908 | 0.242 |
| HALFlow [40] | 0.062 | 0.765 | 0.903 | 0.249 |
| Rigid3DSceneFlow [10] | 0.042 | 0.849 | 0.959 | 0.208 |
| PointFlowHop (Ours) | **0.037** | **0.938** | **0.974** | **0.189** |

Table 2: Comparison of scene flow estimation results on the Argoverse dataset, where the best performance number is shown in boldface.

| Method | EPE3D (m) ↓ | Acc3DS ↑ | Acc3DR ↑ | MAE (rad) ↓ |
|---|---|---|---|---|
| FlowNet3D [25] | 0.455 | 0.01 | 0.06 | 0.736 |
| PointPWC-Net [41] | 0.405 | 0.08 | 0.25 | 0.674 |
| Just Go with the Flow [30] | 0.542 | 0.08 | 0.20 | 0.715 |
| NICP [1] | 0.461 | 0.04 | 0.14 | 0.741 |
| Graph Laplacian [31] | 0.257 | 0.25 | 0.48 | 0.467 |
| Neural Prior [23] | 0.159 | 0.38 | 0.63 | **0.374** |
| PointFlowHop (Ours) | **0.134** | **0.39** | **0.71** | 0.398 |

Table 3: Ego-motion compensation – ICP vs. GreenPCO.

| Ego-motion Method | EPE3D ↓ | Acc3DS ↑ | Acc3DR ↑ | Outliers ↓ |
|---|---|---|---|---|
| ICP [5] | 0.574 | 0.415 | 0.481 | 0.684 |
| GreenPCO [15] | **0.037** | **0.938** | **0.974** | **0.189** |

ICP could be replaced with other advanced model-free methods, it is preferred to use GreenPCO since the trained PointHop++ model is still needed later.

**Performance Gain Due to Object Refinement.** Next, we compare PointFlowHop with and without the object refinement step. The results are shown in Table 4. We see consistent performance improvement in all evaluation metrics with the object refinement step. On the other hand, the performance of PointFlowHop is still better than that of benchmarking methods except for Rigid3DSceneFlow [10] (see Table 1) even without object refinement.

**Performance Gain Due to Flow Refinement.** Finally, we compare PointFlowHop with and without the flow refinement step in Table 5. It is not surprising that flow refinement is crucial in PointFlowHop. However, one may argue the refinement step may be included in any of the discussed methods as a post processing operation. While this argument is valid, we see that even without flow refinement, PointFlowHop still is better than almost all methods (see Table 1). Between object refinement and flow refinement, flow refinement seems slightly more important if we consider all four evaluation metrics jointly.

Table 4: Performance gain due to object refinement.

| Object Refinement | EPE3D ↓ | Acc3DS ↑ | Acc3DR ↑ | Outliers ↓ |
|---|---|---|---|---|
| ✗ | 0.062 | 0.918 | 0.947 | 0.208 |
| ✓ | **0.037** | **0.938** | **0.974** | **0.189** |

Table 5: Performance gain due to flow refinement.

| Flow Refinement | EPE3D ↓ | Acc3DS ↑ | Acc3DR ↑ | Outliers ↓ |
|---|---|---|---|---|
| ✗ | 0.054 | 0.862 | 0.936 | 0.230 |
| ✓ | **0.037** | **0.938** | **0.974** | **0.189** |

### *4.3 Complexity Analysis*

The complexity of a machine learning method can be examined from multiple angles, including training time, the number of model parameters (i.e., the model size) and the number of floating point operations (FLOPs) during inference. These metrics are valuable besides performance measures such as prediction accuracy/error. Furthermore, since some model-free methods (e.g., LOAM [44]) and the recently proposed KISS-ICP [38] can offer state-of-the-art results for related tasks such as Odometry and Simultaneous Localization and Mapping (SLAM), the complexity of learning-based methods deserves additional attention.

To this end, PointFlowHop offers impressive benefits as compared to representative DL-based solutions. Training in PointFlowHop only involves the ego-motion compensation and shape classification steps. For object motion estimation, PointHop++ obtained from the ego-motion compensation step is reused while the rest of the operations in PointFlowHop are parameter-free and performed only in inference.

Table 6 provides details about the number of parameters of PointFlowHop. It adopts the PointHop++ architecture with two hops. The first hop has 13 kernels of dimension 88 while the second hop has 104 kernels of dimension 8. For XGBoost, it has 100 decision tree estimators, each of which has a maximum depth of 3. We also report the training time of PointFlowHop in the same table, where the training is conducted on Intel(R) Xeon(R) CPU E5-2620 v3 at 2.40 GHz.

While we do not measure the training time of other methods ourselves, we use [31] as a reference to compare our training time with others. It took the authors of [31] about 18 hours to train and fine-tune the FlowNet3D [25]

Table 6: The number of trainable parameters and training time of the proposed PointFlowHop.

| Trainable module | Number of Parameters | Training time |
|---|---|---|
| Hop 1 | 1144 | 20 minutes |
| Hop 2 | 832 | |
| XGBoost | 2200 | 12 minutes |
| **Total** | **4176** | **32 minutes** |

Table 7: Comparison of model sizes (in terms of the number of parameters) and computational complexity of inference (in terms of FLOPs) of four benchmarking methods.

| Method | Number of Parameters | FLOPs |
|---|---|---|
| FlowNet3D [25] | 1.23 M (308X) | 11.67 G (61X) |
| PointPWC Net [41] | 7.72 M (1930X) | 17.46 G (92X) |
| FLOT [32] | 110 K (28X) | 54.65 G (288X) |
| PointFlowHop (Ours) | **4 K** (1X) | **190 M** (1X) |

method for the KITTI dataset and about 3 days for the Argoverse dataset. We expect comparable time for other methods. Thus, PointFlowHop is extremely efficient in this context. While the Graph Laplacian method [31] offers a variant where the scene flow is entirely optimized at runtime (non-learning based), its performance is inferior to ours as shown in Table 2.

Finally, we compare the model sizes and computational complexity of four benchmarking methods in Table 7. It is apparent that PointFlowHop demands significantly less parameters than other methods. Furthermore, we compute the number of floating-point operations (FLOPs) of PointFlowHop analytically during inference and report it in Table 7. While calculating the FLOPs, we consider input point clouds containing 8,192 points. Thus, the normalized FLOPs per point is 23.19K. We conclude from the above discussion that PointFlowHop offers a green and high-performance solution to 3D scene flow estimation.

## 5 Conclusion and Future Work

A green and interpretable 3D scene flow estimation method called PointFlowHop was proposed in this work. PointFlowHop takes two consecutive LiDAR point cloud scans and determines the flow vectors for all points in the first scan. It decomposes the flow into vehicle's ego-motion and the motion of an individual object in the scene. The superior performance of PointFlowHop over benchmarking DL-based methods was demonstrated on stereoKITTI and Argoverse datasets. Furthermore, PointFlowHop has advantages in fewer trainable parameters and fewer FLOPs during inference.

One future research direction is to extend PointFlowHop for the 3D object detection task. Along this line, we may detect moving objects using PointFlowHop and derive 3D bounding boxes around them. The clustered points obtained by PointFlowHop may act as an initialization in the object detection process. Another interesting problem to pursue is simultaneous flow estimation and semantic segmentation. The task-agnostic nature of our representation learning can be useful.

**Acknowledgments**

This work was supported by a research gift from Tencent Media Lab. The authors also acknowledge the Center for Advanced Research Computing (CARC) at the University of Southern California for providing computing resources that have contributed to the research results reported within this publication. URL: https://carc.usc.edu.

**References**

[1] B. Amberg, S. Romdhani, and T. Vetter, "Optimal step nonrigid ICP algorithms for surface registration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2007, 1–8.

[2] T. Basha, Y. Moses, and N. Kiryati, "Multi-view scene flow estimation: A view centered variational approach," *International journal of computer vision*, 101, 2013, 6–21.

[3] S. A. Baur, D. J. Emmerichs, F. Moosmann, P. Pinggera, B. Ommer, and A. Geiger, "SLIM: Self-supervised LiDAR scene flow and motion segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, 13126–36.

[4] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[5] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Sensor fusion IV: control paradigms and data structures*, Vol. 1611, International Society for Optics and Photonics, 1992, 586–606.

[6] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, *et al.*, "Argoverse: 3D tracking and forecasting with rich maps," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, 8748–57.

[7] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, 10(3), 1992, 145–55.

[8] Y. Chen, M. Rouhsedaghat, S. You, R. Rao, and C.-C. J. Kuo, "PixelHop++: A small successive-subspace-learning-based (ssl-based) model for image classification," in *IEEE International Conference on Image Processing (ICIP)*, IEEE, 2020, 3294–8.

[9] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, Vol. 96, No. 34, 1996, 226–31.

[10] Z. Gojcic, O. Litany, A. Wieser, L. J. Guibas, and T. Birdal, "Weakly supervised learning of rigid 3D scene flow," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, 5692–703.

[11] X. Gu, Y. Wang, C. Wu, Y. J. Lee, and P. Wang, "HPLFlowNet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, 3254–63.

[12] T. Hackel, J. D. Wegner, and K. Schindler, "Fast semantic segmentation of 3D point clouds with strongly varying density," *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*, 3, 2016, 177–84.

[13] F. Huguet and F. Devernay, "A variational method for scene flow estimation from stereo sequences," in *2007 IEEE 11th International Conference on Computer Vision*, IEEE, 2007, 1–7.

[14] P. Kadam, H. Prajapati, M. Zhang, J. Xue, S. Liu, and C.-C. J. Kuo, "S3I-PointHop: SO (3)-Invariant PointHop for 3D Point Cloud Classification," *arXiv preprint arXiv:2302.11506*, 2023.

[15] P. Kadam, M. Zhang, J. Gu, S. Liu, and C.-C. J. Kuo, "GreenPCO: An Unsupervised Lightweight Point Cloud Odometry Method," in *2022 IEEE 24th International Workshop on Multimedia Signal Processing (MMSP)*, IEEE, 2022, 1–6.

[16] P. Kadam, M. Zhang, S. Liu, and C.-C. J. Kuo, "R-PointHop: A Green, Accurate, and Unsupervised Point Cloud Registration Method," *IEEE Transactions on Image Processing*, 2022.

[17] P. Kadam, M. Zhang, S. Liu, and C.-C. J. Kuo, "Unsupervised point cloud registration via salient points analysis (SPA)," in *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, IEEE, 2020, 5–8.

[18] P. Kadam, Q. Zhou, S. Liu, and C.-C. J. Kuo, "Pcrp: Unsupervised point cloud object retrieval and pose estimation," in *IEEE International Conference on Image Processing (ICIP)*, IEEE, 2022, 1596–600.

[19] C.-C. J. Kuo and Y. Chen, "On data-driven saak transform," *Journal of Visual Communication and Image Representation*, 50, 2018, 237–46.

[20] C.-C. J. Kuo and A. M. Madni, "Green learning: Introduction, examples and outlook," *Journal of Visual Communication and Image Representation*, 2022, 103685.

[21] C.-C. J. Kuo, M. Zhang, S. Li, J. Duan, and Y. Chen, "Interpretable convolutional neural networks via feedforward design," *Journal of Visual Communication and Image Representation*, 60, 2019, 346–59.

[22] R. Li, C. Zhang, G. Lin, Z. Wang, and C. Shen, "Rigidflow: Self-supervised scene flow learning on point clouds by local rigidity prior," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, 16959–68.

[23] X. Li, J. Kaesemodel Pontes, and S. Lucey, "Neural scene flow prior," *Advances in Neural Information Processing Systems*, 34, 2021, 7838–51.

[24] S. Liu, M. Zhang, P. Kadam, and C.-C. J. Kuo, *3D Point Cloud Analysis: Traditional, Deep Learning, and Explainable Machine Learning Methods*, Springer.

[25] X. Liu, C. R. Qi, and L. J. Guibas, "FlowNet3D: Learning scene flow in 3D point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, 529–37.

[26] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, Vol. 2, 1981, 674–9.

[27] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2015, 3061–70.

[28] M. Menze, C. Heipke, and A. Geiger, "Joint 3D estimation of vehicles and scene flow," *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences*, 2, 2015, 427.

[29] M. Menze, C. Heipke, and A. Geiger, "Object scene flow," *ISPRS Journal of Photogrammetry and Remote Sensing*, 140, 2018, 60–76.

[30] H. Mittal, B. Okorn, and D. Held, "Just go with the flow: Self-supervised scene flow estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, 11177–85.

[31] J. K. Pontes, J. Hays, and S. Lucey, "Scene flow from point clouds with or without learning," in *2020 international conference on 3D vision (3DV)*, IEEE, 2020, 261–70.

[32] G. Puy, A. Boulch, and R. Marlet, "Flot: Scene flow on point clouds guided by optimal transport," in *European conference on computer vision*, Springer, 2020, 527–44.

[33] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in Neural Information Processing Systems*, 30, 2017.

[34] J. Quiroga, F. Devernay, and J. Crowley, "Scene flow by tracking in intensity and depth data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshop*, 2012, 50–7.

[35] P. H. Schönemann, "A generalized solution of the orthogonal procrustes problem," *Psychometrika*, 31(1), 1966, 1–10.

[36] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," in *Robotics: science and systems*, Vol. 2, No. 4, 2009, 435.

[37] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade, "Three-dimensional scene flow," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Vol. 2, IEEE, 1999, 722–9.

[38] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "KISS-ICP: In Defense of Point-to-Point ICP Simple, Accurate, and Robust Registration If Done the Right Way," *IEEE Robotics and Automation Letters*, 2023.

[39] C. Vogel, K. Schindler, and S. Roth, "Piecewise rigid scene flow," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, 1377–84.

[40] G. Wang, X. Wu, Z. Liu, and H. Wang, "Hierarchical attention learning of scene flow in 3D point clouds," *IEEE Transactions on Image Processing*, 30, 2021, 5168–81.

[41] W. Wu, Z. Y. Wang, Z. Li, W. Liu, and L. Fuxin, "PointPWC-Net: Cost volume on point clouds for (self-) supervised scene flow estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2020, 88–107.

[42] H. Yang, J. Shi, and L. Carlone, "TEASER: Fast and Certifiable Point Cloud Registration," *IEEE Transactions on Robotics*, 2020.

[43] M. Zhai, X. Xiang, N. Lv, and X. Kong, "Optical flow and scene flow estimation: A survey," *Pattern Recognition*, 114, 2021, 107861.

[44] J. Zhang and S. Singh, "LOAM: Lidar Odometry and Mapping in Real-time," in *Robotics: Science and Systems*, Vol. 2, No. 9, 2014.

[45] M. Zhang, P. Kadam, S. Liu, and C.-C. J. Kuo, "GSIP: Green semantic segmentation of large-scale indoor point clouds," *Pattern Recognition Letters*, 164, 2022, 9–15.

[46] M. Zhang, P. Kadam, S. Liu, and C.-C. J. Kuo, "Unsupervised feedforward feature (UFF) learning for point cloud classification and segmentation," in *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, IEEE, 2020, 144–7.

[47] M. Zhang, Y. Wang, P. Kadam, S. Liu, and C.-C. J. Kuo, "PointHop++: A lightweight learning model on point sets for 3D classification," in *IEEE International Conference on Image Processing (ICIP)*, IEEE, 2020, 3319–23.

[48] M. Zhang, H. You, P. Kadam, S. Liu, and C.-C. J. Kuo, "PointHop: An Explainable Machine Learning method for Point Cloud Classification," *IEEE Transactions on Multimedia*, 22(7), 2020, 1744–55.

[49] Q.-Y. Zhou, J. Park, and V. Koltun, "Fast global registration," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2016, 766–82.