SELF-SUPERVISED 3D SCENE FLOW ESTIMATION FROM LIDAR POINT CLOUDS

A Thesis Presented

by

Y. S. S. V. SASI KIRAN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

May 2020

College of Information and Computer Science

SELF-SUPERVISED 3D SCENE FLOW ESTIMATION FROM LIDAR POINT CLOUDS

A Thesis Presented

by

Y. S. S. V. SASI KIRAN

Approved as to style and content by:

Evangelos Kalogerakis, Co-chair

Erik Learned-Miller, Co-chair

ACKNOWLEDGMENTS

I would like to thank my parents who believe in me all the time. Next, I would like to thank Professor Evangelos for guiding me throughout the course of this masters project. This has been one of the best research experiences of my career. Next, I would also like to thank Prof Erik Learned-Miller for being my second reader for my thesis. Finally, I would like to thank CICS for making my stay as a Master's student memorable.

ABSTRACT

SELF-SUPERVISED 3D SCENE FLOW ESTIMATION FROM LIDAR POINT CLOUDS

MAY 2020

Y. S. S. V. SASI KIRAN

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

With the rise in popularity of autonomous driving, perception on top of LIDAR point clouds is becoming increasingly important. One such task is the point-wise scene flow estimation in real-time which helps in improving the performance of localization and trajectory planning tasks in the autonomous stack. To tackle this task, we present a simple unsupervised clustering and ICP based method which performs as well as the current learning based self-supervised methods and comparable to the fully supervised methods trained on lots of simulated data. We show that this method can be used to generate good flow labels for the large amounts of unlabelled LIDAR scenes easily available in self-driving datasets which can be used to pre-train or fine-tune the existing models. Experiments on fine-tuning the popular flownet3D model using such labels for the point clouds in KITTI odometry dataset show a 25% improvement over the existing performance.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The recent advancements in the fields of artificial intelligence and computer vision has given rise to an increased interest in tackling the autonomous driving problem. One of the critical components of autonomous driving is to have an increasingly accurate perception of the world around the vehicle at all times. One option is to consider alternative sensors for perception other than camera which can provide additional information about the world in real-time. LIDARs are one such sensor which provide the 3D information about the world within a certain range of the ego-vehicle as point clouds.

These point clouds provide depth information for free in addition to the information of the world in the blind-spots of the cameras. As safety is a critical requirement of autonomous driving, perception on top of these LIDAR point clouds helps to augment the perception from cameras and hence helps in better decision making. Additionally, several 3D perception tasks like 3D objection detection [24, 8, 14], 3D semantic segmentation [12, 13, 18], etc. are made easier with the help of these point clouds.

One such task in 3D perception is estimating the 3D scene flow for each point in the point cloud. This involves estimating the point-wise velocities in x, y and z directions given the point clouds from consecutive scenes. This information is quite important as it can help in improving the localization accuracy by identifying points that move due to other vehicles in the scene and separating them. It can also help in making better planning decisions as future trajectories of other vehicles can be better

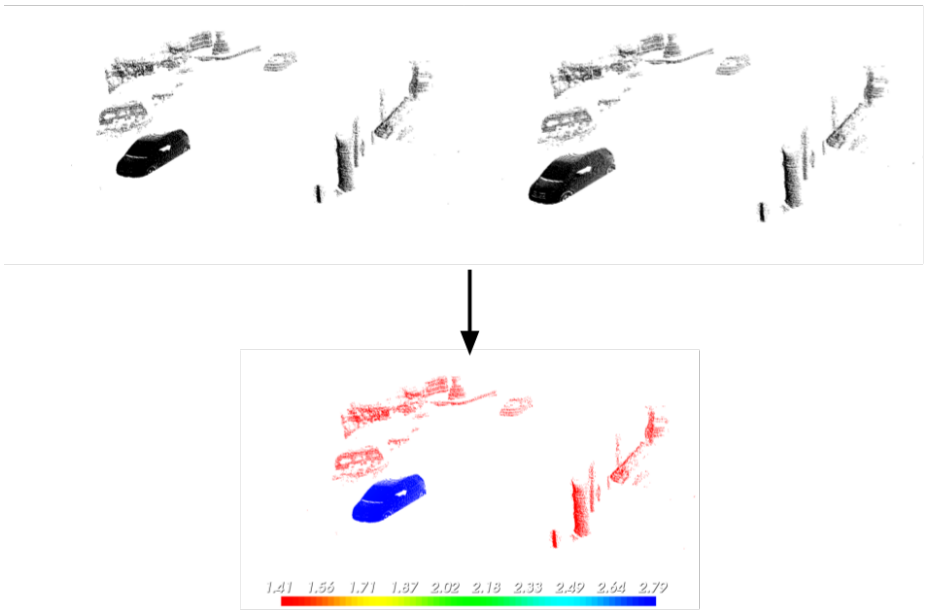

Figure 1.1. An example demonstrating the 3D scene flow estimation task given LIDAR scans from consecutive scenes. The images on the top visualize the point clouds from two consecutive scenes with the same projective camera. Notice how the car in the front moves forward from left to right. Given these two point clouds, our task boils down to computing the 3D scene flow for every point in the first point cloud to make it correspond to second point cloud. For this example, the l2 norm of the target 3D scene flow for the given point clouds is shown in the figure at the bottom. The scale of the scene flow norm is shown using color scale with red being at $1.41m$ and blue being at $2.72m$.

estimated from this 3D scene flow information. An example demonstrating this 3D scene flow estimation task is shown in Figure 1.1

Recently, several large autonomous driving datasets [2, 4, 20] with massive amounts of LIDAR scenes are being made available publicly. Despite this availability, obtaining point-wise labels for LIDAR scenes is a very tedious task that requires large amounts of money and effort. On top of this, obtaining point-wise scene flow labels is extremely difficult compared to other labels like semantic labels and hence is not feasible for these large datasets.

In light of this difficulty of obtaining flow labels, researchers have looked at training models on large simulated datasets like FlyingThings3D [10]. Training on these simulated datasets will always require the bridging of the domain gap between point clouds from simulated and real scenes. The other alternative is to design a self-supervised learning approach [21, 11] to take advantage of large amounts of unlabelled LIDAR scenes available.

In this work, we present a simple unsupervised method for estimating 3D scene flow based on mean-shift clustering and Iterative closest point (ICP). We show that this simple approach performs as well as the current learning based self-supervised methods [21, 11] and comparable in performance to the fully supervised methods [9, 6].

Simultaneously, we show that this method can be used to generate good unsupervised flow labels for large amounts of point clouds in real-world LIDAR scans using the KITTI odometry dataset [5] as an example. We further show that these generated labels can then be used to fine-tune existing scene flow models to improve their performance. We show this improvement using the popular FlowNet3D model as the backbone network and observe better performance than just being trained on FlyingThings3D dataset [10].

CHAPTER 2

RELATED WORK

We first list the work related to general point cloud based learning methods. Then, we list all the related work corresponding to the 3D scene flow estimation from point clouds.

2.1 Point Cloud based Learning

Learning-based point cloud processing methods can be broadly divided into three categories. The most intuitive way is to divide the point cloud into a 3D grid of voxels and extract a feature per voxel which can then be processed using 3D convolutions [24, 8]. The other alternative is to project the point cloud along different viewpoints onto images and then apply standard learning-based methods on these images [17].

The main problem with these two approaches is the loss of information due to voxelization and projection along specific viewpoints respectively. The third category is to process the point clouds as sets of points and then apply learning methods that are invariant to ordering of points in the sets and this avoids the drawbacks of the previous approaches.

Qi *et al.* [12] were the first to propose set-based learning architecture with shared convolutions and pooling operations called PointNet. A later improvement over this called PointNet++ which uses hierarchical network to capture local neighborhood features was then proposed [13]. Su *et al.* [16] look at projecting the point set into high dimensional lattice and then use bilateral convolutions as the main module of

the network. Other alternative approaches [22, 18, 15] involve using various types of sparse convolutions to handle the inherent sparsity in the point clouds.

2.2 3D Scene Flow Learning

Liu *et al.* [9] were the first to design a deep network for scene flow estimation from point sets. They propose a deep network named FlowNet3D that estimates the scene flow directly from point cloud. PointNet++ [13] is used for extracting features from individual point clouds which are then combined using flow embedding layers. These features are then upsampled to predict flow for point clouds from consecutive scenes.

Gu *et al.* [6] also propose a network for scene flow estimation which uses modules based on Bilateral Convolutional Layers [16] to combine features from consecutive point clouds after projecting them onto high dimensional lattice. Behl *et al.* [1] propose to jointly learn 3D bounding box and rigid body motion of objects in the scene and use voxelization to extract features from point clouds. All these methods are trained on a synthetic dataset and are then evaluated on real LIDAR scans.

2.3 Self-Supervised Flow Learning

There are several works [19, 23] that jointly estimate depth, object segmentation, ego-motion and rigid body motion from videos in a self-supervised manner. However, very little work has been done in scene flow estimation from point clouds in a self-supervised manner. It is a promising area of research as flow labels are hard to obtain for real point clouds to train in a supervised manner.

Parallel to our work, Mittal *et al.* [11] design a self-supervised training approach with nearest neighbor loss and a cyclic consistency loss on top of a FlowNet3D backbone network. Wu *et al.* [21] design a network that can be trained in a self-supervised manner using a similar nearest neighbor loss as [11] and smoothness and regularization constraints. Unlike these approaches which design self-supervised loss functions, we

look at obtaining self-supervised labels for the large amounts of unlabeled data which can then be used as supervised signals to train the scene flow networks. Additionally, we also observe that our simple self-supervised baseline method with clustering and ICP obtains results that are as well as these other works.

CHAPTER 3

METHODOLOGY

3.1 Problem Definition

We now formally define the 3D scene flow estimation problem. Computing the point-wise scene flow requires the LIDAR scans of consecutive scenes as input. Define $\mathbf{P} \in R^{n_1 \times 3}$ and $\mathbf{Q} \in R^{n_2 \times 3}$ as the input point clouds corresponding to the consecutive scenes. They contain n_1 and n_2 points respectively with the three columns being the (x, y, z) locations of each point. We note that we do not use any additional information like reflectance, normals, etc for each point in the LIDAR scans.

Our 3D scene flow estimation task then boils down to computing the 3D displacement required for each point in $\mathbf{P}$ to make it correspond to the point cloud of next scene $\mathbf{Q}$. Formally, let $x \in \mathbf{P}$ be a point in the first point cloud which changes to a new position x' in the second scene. Then, the flow for this point is just the 3D displacement $f = x' - x$ which when computed for every point in $\mathbf{P}$ gives a vector $\mathbf{F} \in R^{n_1 \times 3}$. We note that this point x' may not be explicitly present in the second point cloud $\mathbf{Q}$ due to the difference in LIDAR sampling angles of the scenes.

3.2 Clustering-based ICP

We now describe the self-supervised method which uses clustering and ICP to compute scene flow from point clouds which is as well as learning based methods. The motivation for designing this method is the fact that ICP by itself computes good scene flow for the points which correspond to the background in the scene. For other rigid bodies like cars, trucks whose motion is different from the ego-motion,

ICP can still be used separately to compute good scene flow results as long as we find the correct correspondences for these objects across the scenes.

Based on this motivation, we aim to separate the point clouds into foreground object clusters and a background cluster so that their scene flows can be computed separately. Background clusters from consecutive scenes can be matched directly using ICP to compute scene flow for background points. Simultaneously, we need to match the foreground clusters from both the consecutive scenes to find correct associations. If the objects clusters are correctly matched, then ICP can be used directly between these two matched clusters to find scene flow for points corresponding to that object cluster.

We now describe this approach in detail which is summarized in Figure 3.1. We start by using ICP on whole point clouds to compute an initial estimate of the ego-motion transformation $\tau \in SE(3)$ and a corresponding residual vector $\mathbf{R} \in R^{n_1 \times 3}$. We then transform the the first point cloud $\mathbf{P}$ using this estimate transformation to obtain an ego-shifted point cloud $\mathbf{P'} = \tau(\mathbf{P})$.

We then cluster the ego-shifted point cloud $\mathbf{P'}$ and the second point cloud $\mathbf{Q}$ using mean shift clustering with a bandwidth $\mathbf{B}$. We chose mean-shift clustering over other clustering methods because of its intrinsic ease of parameter specification. Other clustering methods like K-means, hierarchical require specifying the number of clusters or the distance threshold respectively while also not handling variable density clusters properly. Other density based methods like DBSCAN, OPTICS require proper parameter tuning while the mean-shift clustering only requires specifying the bandwidth of the kernel which is slightly easier to tune.

We next iterate over each cluster in $\mathbf{P'}$ and count the number of points in the cluster with l2-norm of the residual greater than a residual threshold (α). If the number of such points exceeds a certain percentage ($\beta\%$), then it contains lots of points that are not aligned with the estimated ego-motion transformation τ. We thus

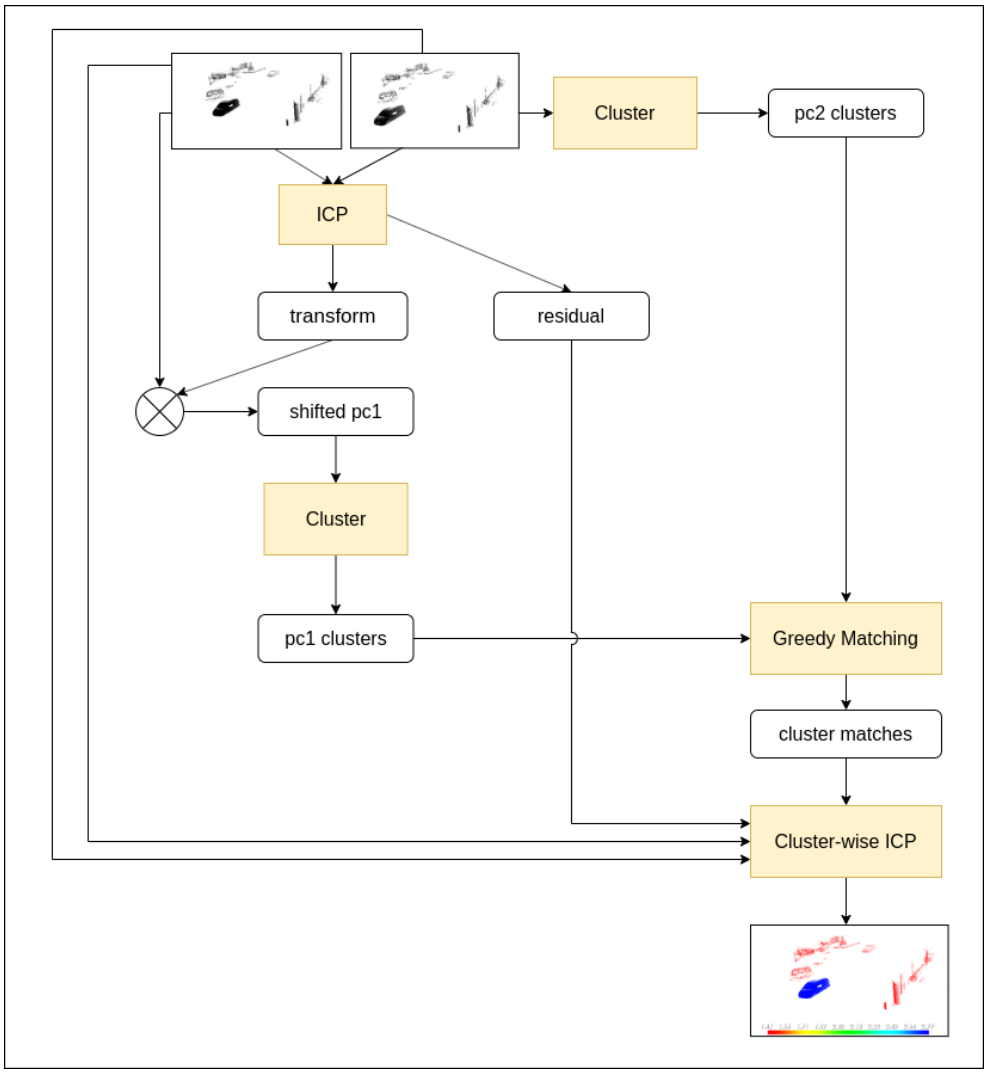

Figure 3.1. The above figure outlines the pipeline for the clustering-based ICP method. Note that the final "cluster-wise ICP" method uses residual to assign clusters as part of background or foreground. Then ICP is done for background clusters as a whole while it is done cluster-wise for each foreground cluster.

assign such a cluster as part of foreground object and hence match it with clusters in the other scene **Q**. Otherwise, we can assign this cluster to be a part of the background as it follows the background transformation τ. We later tune these hyperparameters α and β to optimize the scene flow performance.

Finally, we match each foreground cluster in $\mathbf{P}'$ with a single closest cluster in **Q** based on the euclidean distance between the cluster centers. We then use ICP to compute the transformation τ_{bg} for all the background points and the individual transformations τ_{fg}^{i} for the i^{th} foreground cluster using matched points only. We combined these estimated transformations to get the 3D point-wise scene flow for points clouds **P** and **Q**

3.3 Finetuning FlowNet3D

The scene flow results generated from the clustering-based ICP method described in Chapter 3.2 can be used to finetune the existing learning-based scene flow methods. We show the finetuning using FlowNet3D network as an example which can be similarly extended to other networks.

We first give a short description of the flownet3D architecture followed by describing the process of finetuning on labels generated by the clustering-based ICP method.

3.3.1 FlowNet3D

Flownet3D [9] is the first proposed deep neural network that estimates scene flow from point clouds in an end-to-end manner. The model for estimating scene flow is divided into three modules which is summarized in the Figure 3.2.

The first module consists of point-wise feature learning network to learn features for each point cloud separately. Here, the authors use PointNet++ [13] to extract features from each point cloud individually.

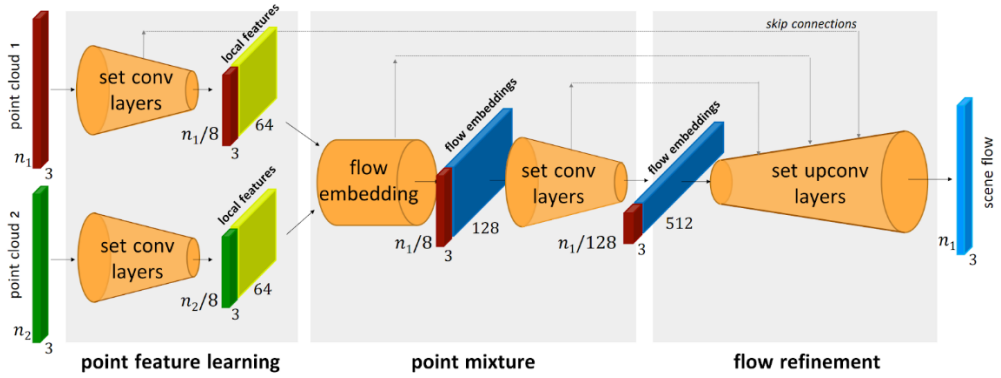

Figure 3.2. Architecture diagram of FlowNet3D model.

The flow embedding layer then combines the point cloud features from both frames. For each point in the first frame, we gather all the points in the second frame that are within a radius from the (x, y, z) position of that point. We then pass them through PointNet [12] style fully connected followed by maxpool layers to get a feature for every point in first frame. This can be visualized from the Figure 3.3.

Finally, the features from each of the new points are upsampled back to get a scene flow prediction for each point using the set upconv layers.

3.3.2 Finetuning

For finetuning the FlowNet3D model, we first load the pre-trained model weights obtained from training the model on FlyingThings3D [10] which is a synthetic dataset. We then finetune the network on unlabeled real-world LIDAR scans by generating labels using the unsupervised clustering-based ICP approach. We then use these labels as self-supervisory signal for finetuning FlowNet3D model.

We explore the following loss function for finetuning FlowNet3D.

L2 Loss: We use the labels generated by the clustering-based ICP method described in Chapter 3.2 as a supervisory signal. Given the point clouds **P** and **Q**, we

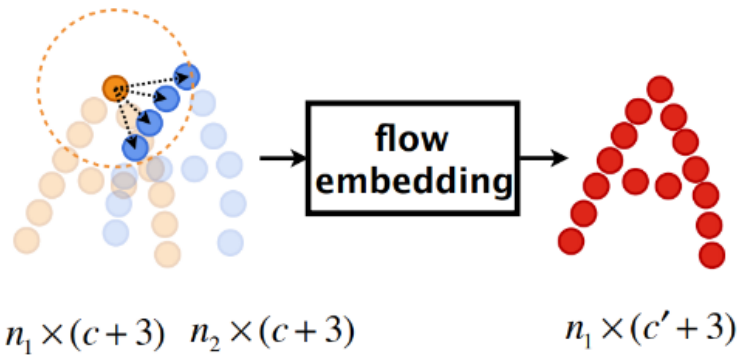

Figure 3.3. Pictorial representation of the flow embedding module. For each point in the first point cloud (displayed in orange), we group the nearest points from the second point cloud (denoted in blue) as shown. Then, we get a feature vector (denoted in red) for each point in the first point cloud by passing these grouped points through a PointNet style point aggregator network.

simply compute the average squared l2 distance between the estimated scene flow $\{\hat{sf}_i\}_{i=1}^N$ and the generated label $\{sf_i\}_{i=1}^N$ and use this as a supervised objective.

$$\mathcal{L}_{sup} = \frac{1}{N}\sum_{i=1}^{N} ||sf_i - \hat{sf}_i||_2^2 \tag{3.1}$$

CHAPTER 4

EXPERIMENTS

4.1 Datasets

We describe in short detail about the datasets used for the pre-trained model as well as fine-tuning and evaluating FlowNet3D model on scene flow estimation task.

4.1.1 FlyingThings3D:

This [10] is a synthetic dataset that originally contains stereo and depth images rendered from 3D objects from ShapeNet [3] moving in the scene. Liu *et al.* [9] randomly select $20,000$ examples for training and 2000 examples for evaluation. They also pre-process the data by converting disparity maps into 3D point clouds and optical flow to scene flow labels. This dataset is widely used for pre-training the scene flow estimation models like FlowNet3D [9].

4.1.2 KITTI Odometry dataset:

This dataset [5] consists of real world image and LIDAR data collected for 22 closed-loop tracks. Each track contains a variable number of continuous LIDAR point cloud scans depending on the length of the track. This dataset contains a total of $100k$ unlabeled LIDAR point clouds approximately. We use this large unlabelled dataset for generating scene flow labels using clustering-based ICP method which are further used for fine-tuning FlowNet3D.

4.1.3 KITTI Scene Flow 2015:

This dataset also contains camera and LIDAR scans from 200 real-world scenes. Gu *et al.* [6] process the dataset to obtain 3D point-wise flow labels for 142 scenes in training set. Following all the previous works [6, 21], this dataset is used to compare the performance of the scene flow results.

4.2 Evaluation Metrics

We use the standard evaluation metrics to evaluate the quality of the scene flow estimates compared to ground truth labels. The first metric is 3D End Point Error (EPE3D) which is the average l2 distance between the predicted and ground truth scene flow same as the supervised objective in Equation 3.1. The next metric is accuracy (Acc3D) which measures the percentage of points whose scene flow predictions are within a threshold of the ground truth labels. We measure accuracy at two thresholds - (Acc3D-5%) where EPE3D $< 0.05m$ or the relative error is less than 5% and (Acc3D-10%) where EPE3D $< 0.10m$ or the relative error is less than 10%.

4.3 Pre-processing

We use the standard pre-processing employed by [9, 6, 21, 11] for the LIDAR scans while finetuning FlowNet3D. We first remove the ground points from the point clouds by assigning all the points which are below $1.4m$ in the z-direction as ground points. This is because finding correspondences for ground points would not yield good results on account of it being a flat surface most of the time.

We then consider only the points which are within $35m$ of the ego vehicle. This is because most foreground moving objects are within this depth range. While finetuning FlowNet3D, we sample n points randomly from the first point cloud. We then select the corresponding nearest points in the second point cloud for each point in the first after shifting the first point cloud by the flow label. If the points are randomly

selected in the second point cloud also, there may not be correspondences for the points in the first cloud which makes the training harder. We use $n = 16384$ while finetuning FlowNet3D.

4.4 Results

4.4.1 Clustering-based ICP method

Quantitative results comparing the best performing clustering based ICP method with the other self-supervised methods is shown in Table 4.1

Method	EPE3D⇓	Acc3D-5%⇑	Acc3D-10%⇑
ICP	0.5181	6.69%	16.67%
PointPWC-Net [21]	0.2549	23.79%	49.57%
Clustering-based ICP (Ours)	**0.2490**	**58.5%**	**65.3%**

Table 4.1. The above table compares the clustering-based ICP method with other unsupervised and self-supervised methods.

We notice that this simple unsupervised clustering-based ICP method outperforms the current best self-supervised method PointPWC-Net [21] which involves training a deep network. Noticeably, we gain a remarkable **145%** improvement in Acc3D-5% while also improving the Acc3D-10% by **32%**. We again note that this clustering-based ICP method does not involve any learning.

Similarly, we compare this clustering-based ICP method with the existing fully-supervised methods and this is shown in Table 4.2. We note that in spite of our method lacking any training, we perform almost as well as fully supervised methods.

In particular, we note that we perform the second best in Acc3D-5% and the third best in Acc3D-10% while comparing with fully supervised methods. This shows the good performance of our simple method.

Method	EPE3D⇓	Acc3D-5%⇑	Acc3D-10%⇑
FlowNet3D [9]	0.1767	37.38%	66.77%
SPLATFlowNet [16]	0.1988	21.74%	53.91%
original BCL	0.1729	25.16%	60.11%
HPLFlowNet [6]	0.1169	47.83%	77.76%
PointPWC-Net [21]	**0.0694**	**72.81%**	**88.84%**
Clustering-based ICP (Ours)	0.2490	58.5%	65.3%

Table 4.2. The above table compares the clustering-based ICP, unsupervised method with other fully supervised methods.

4.4.1.1 Hyper-parameter tuning

We now describe the effect of the hyper-parameters on the performance of clustering based ICP method. We note that the parameters of this method are the bandwidth (B) of the mean-shift clustering, residual threshold (α) and point percentage threshold (β%). We present the results in Table 4.3.

Method	EPE3D⇓	Acc3D-5%⇑	Acc3D-10%⇑
$(B = 2, \alpha = 0.2, \beta = 10\%)$	**0.249**	**58.5%**	**65.3%**
$(B = 1, \alpha = 0.2, \beta = 10\%)$	0.342	48.1%	56.7%
$(B = 3, \alpha = 0.2, \beta = 10\%)$	0.276	48.9%	56.3%
$(B = 2, \alpha = 0.1, \beta = 10\%)$	0.258	52.3%	60.3%
$(B = 2, \alpha = 0.3, \beta = 10\%)$	0.291	44.9%	53.4%
$(B = 2, \alpha = 0.2, \beta = 15\%)$	0.286	48.3%	56.0%
$(B = 2, \alpha = 0.2, \beta = 5\%)$	0.264	53.1%	60.6%

Table 4.3. The effect of changing the hyper-parameters on the performance of the clustering-based ICP method.

4.4.1.2 Qualitative Results

We first observe success cases of this method as shown in Figure 4.1. As observed in both these cases, the moving vehicle has been assigned a correct flow due to foreground clustering and ICP on individual clusters.

We now turn to the cases where this method does not perform as well as shown by examples in Figure 4.2. The left point cloud has correct flow for cars but they

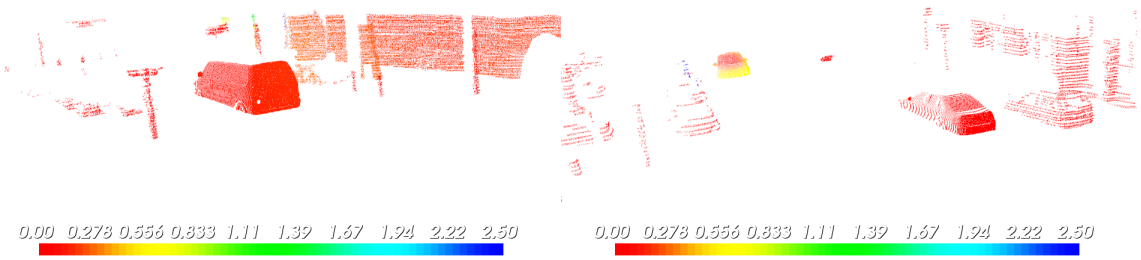

Figure 4.1. Example of point clouds where the clustering-based ICP method works very well. The color scale shown represents the error in flow compared to ground truth. Red being correct to blue being $2.5m$ error.

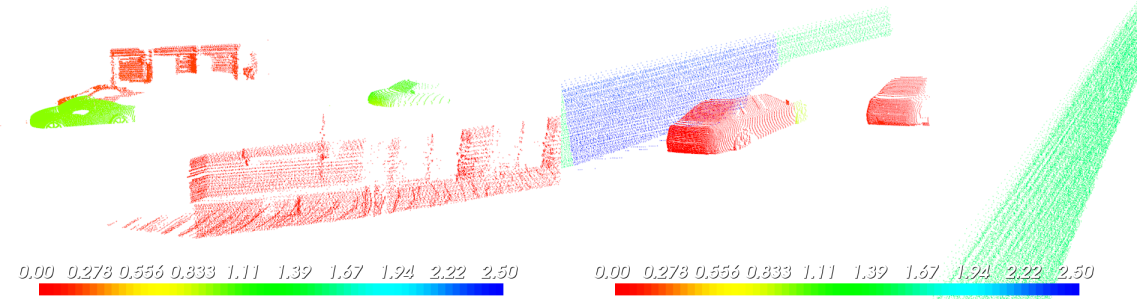

Figure 4.2. Example of point clouds where the clustering-based ICP method does not work well. The color scale shown represents the error in flow compared to ground truth. Red being correct to blue being $2.5m$ error.

	Method	EPE3D ⇓	Acc3D-5% ⇑	Acc3D-10% ⇑
Fully-Supervised	FlowNet3D [9]	0.1727	18.56%	55.61%
	SPLATFlowNet [16]	0.1988	21.74%	53.91%
	original BCL	0.1729	25.16%	60.11%
	HPLFlowNet [6]	0.1169	47.83%	77.76%
	PointPWC-Net [21]	**0.0694**	**72.81%**	**88.84%**
Self-Supervised	ICP	0.5181	6.69%	16.67%
	PointPWC-Net [21]	0.2549	23.79%	49.57%
	Clustering-based ICP (Ours)	0.2490	**58.5%**	65.3%
	Finetuned FlowNet3D (Ours)	**0.1286**	36.6%	**72.31**%

Table 4.4. The above table shows the performance of both self-supervised and fully supervised methods on 3D scene flow estimation task.

are both slightly off from ground truth. The right case is the one where this method does not perform well at all. This is because the walls of the background lack any texture and are just straight walls. This makes it hard for ICP to find the right correspondences.

4.4.2 Finetuning FlowNet3D

4.4.2.1 Finetuning Parameters

We initially load the pre-trained flownet3d model which is trained on FlyingThings3D [10] synthetic dataset. The inputs to the finetuning network have a sample size of $n = 16,384$ points for each point cloud. We use Adam optimizer [7] with a learning rate of 0.001. The learning rate follows staircase decay with a decay rate of 0.7 after every $2,000,000$ steps. We use a batch size of 16 and finetune the model for 65 epochs.

4.4.2.2 Quantitative Results

The quantitative comparison of all the methods, both supervised and self-supervised are presented in Table 4.4. We observe the performance of finetuning flownet3d using labels from clustering-based ICP method performs much better than the original supervised flownet3d. We observe an improvement of **25%** in EPE3D, **97%** in Acc3D-

Method	EPE3D⇓	Acc3D-5%⇑	Acc3D-10%⇑
Pre-trained	0.1727	18.56%	55.61%
Finetuning-5.5k	0.1693	21.2%	51.82%
Finetuning-full	0.1286	36.6%	72.31%

Table 4.5. The results of finetuning on only a part of the dataset i.e. 250 examples per each sequence with a total of 22 sequences.

5% and **30%** in Acc3D-10%. We also note that the performance is comparable to fully supervised methods.

One point to note is the fact that Acc3D-5% of the finetuned model did not exceed that of clustering-based ICP method even though it improved upon original flownet3d model. This could possibly be attributed as improving the overall performance while not reaching the fine-grained limits for scene flow estimation.

4.4.2.3 Few-shot Learning

We also look at the performance of finetuning when we do not use all the dataset and only use first few examples. The performance comparison when only using first 250 examples per each sequence (22 sequences in dataset) is compared against the full performance in Table 4.5.

We observe that the few-shot finetuning model has an improvement of 2% in EPE3D and 14% in Acc3D-5% with a slight decrease in Acc3D-10% compared to original FlowNet3D model. This concludes that the few-shot model focuses more on correcting fine-grained errors compared to coarse and big errors.

4.4.2.4 Qualitative Results

We refer to Figure 4.3 to show some of the point clouds where finetuned model performs really well. We observe that error is mostly zero for almost all of the points which showcases the good performance of the model.

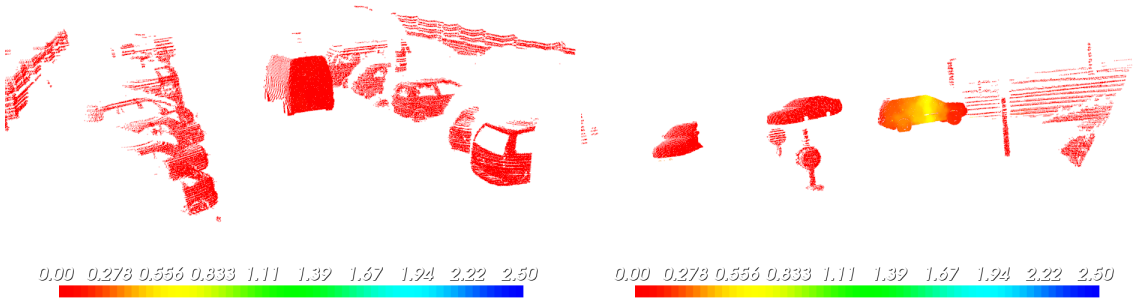

Figure 4.3. Example of well performing output obtained by finetuning the flownet3d model in self-supervised manner. The color scale shown represents the error in flow compared to ground truth. Red being correct to blue being $2.5m$ error.

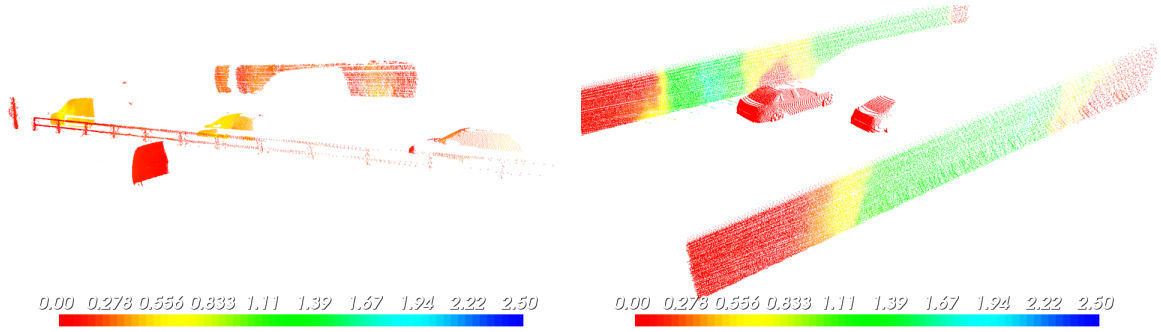

Figure 4.4. Example of not so well performing output obtained by finetuning the flownet3d model in self-supervised manner. The color scale shown represents the error in flow compared to ground truth. Red being correct to blue being $2.5m$ error.

Similarly, we examine the cases where the model performs not well in Figure 4.4. First observing the figure to the left we notice that although the flow is mostly identified in the correct direction, the magnitude of the flow is slightly off from ground truth (around $0.5m$ off) for most of the foreground moving cars. This pattern repeats for few cases which explains the lesser Acc3D-5% compared to clustering-based ICP method observed while having higher Acc3D-10%. Secondly, we again observe from the figure on the right that the model still has problems when there is a lack of variation in the background (it being a flat wall). In this case, we observe that the results are slightly better compared to the clustering-based ICP model.

CHAPTER 5

APPLICATIONS & FUTURE WORK

5.1 Applications

We now discuss some of the applications of predicting 3D scene flow from LIDAR point clouds.

5.1.1 Motion Segmentation

The estimated 3D scene flow can also be used for doing motion segmentation i.e. separating moving objects from stationary background. Motion Segmentation helps in better decision making in the planning phase of the autonomous pipeline as more informed actions can be taken. It also helps in improving the localization accuracy as ego-motion can be separated from motion of other vehicles through motion segmentation.

A simple implementation of motion segmentation is to cluster the points based on both location and flow values. An example of such an approach is shown in Figure 5.1.

5.1.2 Object Tracking

Tracking vehicles is one of the most important problems in autonomous vehicles. We can use the scene flow information to design a simple object tracker which performs pretty well. We start with 3D bounding boxes detected for a given scene. Then we can just use greedy matching to match objects with trajectories using the scene flow information.

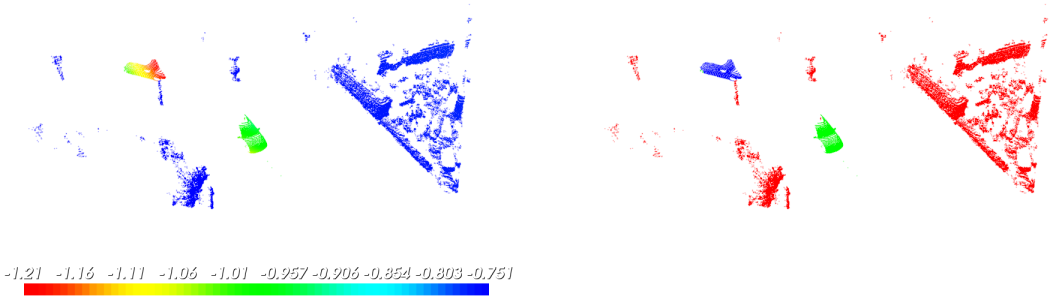

Figure 5.1. The figure shows an example of motion segmentation using estimated scene flow. The left image shows the estimated scene flow for the scene. The right image shows the clustering obtained by motion segmentation. Notice the clear separation of background and the two moving objects.

5.2 Future Work

There are several promising directions to explore in the future work. The first direction is extending the simple clustering-based ICP method into an iterative method where the subsequent iterations improve upon the flow from previous iterations. The next direction would be adding additional losses like cyclic consistency to make finetuning easier. Finally, we would try to similarly finetune other models in literature to get better performance using labels generated from clustering-based ICP method.

BIBLIOGRAPHY

[1] Behl, Aseem, Paschalidou, Despoina, Donné, Simon, and Geiger, Andreas. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *CVPR* (2019), Computer Vision Foundation / IEEE, pp. 7962–7971.

[2] Caesar, Holger, Bankiti, Varun, Lang, Alex H., Vora, Sourabh, Liong, Venice Erin, Xu, Qiang, Krishnan, Anush, Pan, Yu, Baldan, Giancarlo, and Beijbom, Oscar. nuscenes: A multimodal dataset for autonomous driving. *CoRR abs/1903.11027* (2019).

[3] Chang, Angel X., Funkhouser, Thomas, Guibas, Leonidas, Hanrahan, Pat, Huang, Qixing, Li, Zimo, Savarese, Silvio, Savva, Manolis, Song, Shuran, Su, Hao, Xiao, Jianxiong, Yi, Li, and Yu, Fisher. Shapenet: An information-rich 3d model repository, 2015. cite arxiv:1512.03012.

[4] Chang, Ming-Fang, Lambert, John, Sangkloy, Patsorn, Singh, Jagjeet, Bak, Slawomir, Hartnett, Andrew, Wang, De, Carr, Peter, Lucey, Simon, Ramanan, Deva, and Hays, James. Argoverse: 3d tracking and forecasting with rich maps. *CoRR abs/1911.02620* (2019).

[5] Geiger, Andreas, Lenz, Philip, and Urtasun, Raquel. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).

[6] Gu, Xiuye, Wang, Yijie, Wu, Chongruo, Lee, Yong Jae, and Wang, Panqu. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *CVPR* (2019), Computer Vision Foundation / IEEE, pp. 3254–3263.

[7] Kingma, Diederik, and Ba, Jimmy. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[8] Lang, Alex H., Vora, Sourabh, Caesar, Holger, Zhou, Lubing, Yang, Jiong, and Beijbom, Oscar. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR* (2019), Computer Vision Foundation / IEEE, pp. 12697–12705.

[9] Liu, Xingyu, Qi, Charles R., and Guibas, Leonidas J. Flownet3d: Learning scene flow in 3d point clouds. In *CVPR* (2019), Computer Vision Foundation / IEEE, pp. 529–537.

[10] Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., and Brox, T. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). arXiv:1512.02134.

[11] Mittal, Himangi, Okorn, Brian, and Held, David. Just go with the flow: Self-supervised scene flow estimation. *CoRR abs/1912.00497* (2019).

[12] Qi, Charles R., Su, Hao, Mo, Kaichun, and Guibas, Leonidas J. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016. cite arxiv:1612.00593.

[13] Qi, Charles R., Yi, Li, Su, Hao, and Guibas, Leonidas J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. cite arxiv:1706.02413.

[14] Qi, Charles Ruizhongtai, Liu, Wei, Wu, Chenxia, Su, Hao, and Guibas, Leonidas J. Frustum pointnets for 3d object detection from RGB-D data. *CoRR abs/1711.08488* (2017).

[15] Ren, Mengye, Pokrovsky, Andrei, Yang, Bin, and Urtasun, Raquel. Sbnet: Sparse blocks network for fast inference. In *CVPR* (2018), IEEE Computer Society, pp. 8711–8720.

[16] Su, Hang, Jampani, Varun, Sun, Deqing, Maji, Subhransu, Kalogerakis, Evangelos, Yang, Ming-Hsuan, and Kautz, Jan. Splatnet: Sparse lattice networks for point cloud processing. In *CVPR* (2018), IEEE Computer Society, pp. 2530–2539.

[17] Su, Hang, Maji, Subhransu, Kalogerakis, Evangelos, and Learned-Miller, Erik G. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV* (2015), IEEE Computer Society, pp. 945–953.

[18] Thomas, Hugues, Qi, Charles R., Deschaud, Jean-Emmanuel, Marcotegui, Beatriz, Goulette, François, and Guibas, Leonidas J. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV* (2019), IEEE, pp. 6410–6419.

[19] Vijayanarasimhan, Sudheendra, Ricco, Susanna, Schmid, Cordelia, Sukthankar, Rahul, and Fragkiadaki, Katerina. Sfm-net: Learning of structure and motion from video. *CoRR abs/1704.07804* (2017).

[20] Wang, Peng, Huang, Xinyu, Cheng, Xinjing, Zhou, Dingfu, Geng, Qichuan, and Yang, Ruigang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence* (2019).

[21] Wu, Wenxuan, Wang, Zhiyuan, Li, Zhuwen, Liu, Wei, and Li, Fuxin. Pointpwc-net: A coarse-to-fine network for supervised and self-supervised scene flow estimation on 3d point clouds. *CoRR abs/1911.12408* (2019).

[22] Xu, Yifan, Fan, Tianqi, Xu, Mingye, Zeng, Long, and Qiao, Yu. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV (8)* (2018), Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, Eds., vol. 11212 of *Lecture Notes in Computer Science*, Springer, pp. 90–105.

[23] Zhou, Tinghui, Brown, Matthew, Snavely, Noah, and Lowe, David G. Unsupervised learning of depth and ego-motion from video. In *CVPR* (2017), IEEE Computer Society, pp. 6612–6619.

[24] Zhou, Yin, and Tuzel, Oncel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR abs/1711.06396* (2017).