

Diploma Thesis



Czech  
Technical  
University  
in Prague

**F3**

Faculty of Electrical Engineering  
Department of Cybernetics

## Motion Feature Self-Supervised Learning in 3D Point Cloud Data

**Bc. Šimon Pokorný**

Supervisor: Ing. Patrik Vacek  
May 2023



## I. Personal and study details

Student's name: **Pokorný Šimon**

Personal ID number: **487004**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

## II. Master's thesis details

Master's thesis title in English:

**Motion Feature Self-Supervised Learning in 3D Point Cloud Data**

Master's thesis title in Czech:

**Uení bez u itele pro odhad pohybu v mra nu 3D bod**

Guidelines:

Perception models in the 3D point cloud domain require a lot of human supervision to achieve good results. The unsupervised machine learning methods, which help to mitigate the aforementioned cost, are still in development. To help the neural networks, the unsupervised methods utilizes physical constraints such as the rigidity of moving objects [1], feature matching with tracking [2], or their structure affiliation in the scene [3]. The goal of the thesis is to explore the spatial-temporal consistency of the dynamic objects in point cloud sequences and design a self-supervised method to improve motion feature learning (motion segmentation and/or motion flow). The recommended steps can be summarized as follows:

1. Use a point cloud dataset that contains dynamic objects and is ordered in sequence.
2. Perform the registration method to synchronize point cloud sequences or use the attached dataset odometry to establish the spatial-temporal synchronization between the frames.
3. Explore existing methods of motion flow and dynamic object segmentation to distinguish between movable classes and static classes and their per-point velocity estimation.
4. Design a self-supervised method to learn the motion features and apply it to the point cloud dataset.
5. Compare your results with the previous methods by auto-generating labels for the dynamic vs. static objects and training neural networks on dynamic point segmentation.

Bibliography / sources:

- [1] Z. Song and B. Yang, "OGC: Unsupervised 3d object segmentation from rigid dynamics of point clouds," in Advances in Neural Information Processing Systems, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: <https://openreview.net/forum?id=ecNbE00tqBU>
- [2] H. Jiang, K. Lan, L. Hui, G. Li, J. Xie, and J. Yang, "Point cloud registration-driven robust feature matching for 3d siamese object tracking," 2022. [Online]. Available: <https://arxiv.org/abs/2209.06395>
- [3] H. Lim, S. Hwang, and H. Myung, "Erasor: Egocentric ratio of pseudo occupancy-based dynamic object removal for static 3d point cloud map building," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 2272–2279,

Name and workplace of master's thesis supervisor:

**Ing. Patrik Vacek Vision for Robotics and Autonomous Systems FEE**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **03.02.2023**

Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Patrik Vacek  
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.  
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

### III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature

## Acknowledgements

I would like to thank my thesis advisor Ing. Patrik Vacek for his valuable guidance, patience, motivation, continuous support and also to prof. Ing. Tomáš Svoboda, Ph.D., both of them provided me with excellent expertise and consultations.

## Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 26, 2023

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 26. května 2023

## Abstract

We conducted research in the field of self-supervised motion flow in 3D point clouds. Throughout our exploration, we observed that no existing models had attempted to leverage the sequential characteristics of automotive datasets. Taking this into account, we introduced two self-supervised losses to regularize the learning process of the model. The first loss, called Velocity loss, aims to smooth the derivative of the flow of points across multiple time frames. By doing so, we encourage a more smooth flow across multiple time frames. Additionally, we proposed the Dynamic loss, which separates the motion patterns of dynamic objects from the motion of the ego and utilizes this knowledge to generate pseudo labels for improving motion segmentation. Furthermore, we discovered that the model we were using faced challenges in segmentation. To address this issue, we upgraded the Artificial Label loss to incorporate object homogeneity, taking into account the rigid consistency of objects within the scene. Lastly, we proposed a similar refinement module, designed specifically for evaluation purposes. Experimental results demonstrated a significant improvement in the model's performance, with an enhancement of nearly 3 cm on the  $AAE_{50-50}$  metric when the Dynamic Loss and Refinement Module was added. This outcome highlights the potential utilization of sequential data for self-supervised learning of motion flow.

**Keywords:** self-supervised learning, motion flow, motion segmentation, 3D point cloud data

**Supervisor:** Ing. Patrik Vacek  
ČVUT v Praze  
Fakult elektrotechnická  
Katedra kybernetiky  
Karlovo náměstí 13  
121 35 Praha 2

## Abstrakt

Provedli jsme průzkum v oblasti učení pohybů z 3D mračen bodů bez učitele. Během našeho průzkumu jsme zjistili, že žádný ze stávajících přístupů se nesnažil využít sekvenční povahy automobilových datasetů. S tímto zohledněním jsme představili dvě ztrátové funkce, které slouží k regularizaci učícího procesu modelu bez učitele. První ztrátová funkce nazvaná ztráta rychlosti se zaměřuje na vyhlazení derivace rychlosti bodů skrz určitý interval. Tato ztrátová funkce by měla zajistit plynulejší a tedy stabilnější predikci. Dále jsme navrhli dynamickou ztrátovou funkci, která se snaží oddělovat pohybové vzorce dynamických objektů od pohybu snímáčího vozidla. Toto rozdělení se následně využívá ke generování pseudo-značek pro zlepšení segmentace pohybu. Dále jsme zjistili, že použitý model má problém s pohybovou segmentací. Pro vyřešení tohoto problému jsme vylepšili ztrátovou funkci, která se snaží učit segmentaci z modelem vytvořených pseudo-značek. Toto vytváření nyní zohledňuje kompaktnost objektů ve scéně. Nakonec jsme navrhli modul na podobném principu, který je specificky navržen pro účely evaluace bez nutnosti učení. Experimentální výsledky ukázaly významné zlepšení predikce modelu a to zmenšení chyby o téměř 3 cm na metrice  $AAE_{50-50}$  při přidání Dynamické ztrátové funkce a modulu. Tento výsledek poukazuje na potenciální využití sekvenčních dat pro učení pohybů bez učitele.

**Klíčová slova:** učení bez učitele, pohyb scény, segmentace pohybu, mračna 3D bodů

**Překlad názvu:** Učení bez učitele pro odhad pohybu v mračnu 3D bodů

# Contents

<b>1 Introduction</b>	<b>1</b>	3.5.1 Self-supervised Refinement	
<b>2 Related Work</b>	<b>3</b>	Module . . . . .	37
2.1 Classical Pointcloud Alignment . . .	3	3.5.2 Object Aware Artificial Label	
2.1.1 Kabsch Algorithm . . . . .	4	Loss . . . . .	37
2.1.2 Iterative Closest Point . . . . .	4	3.5.3 Experiments . . . . .	38
2.2 Losses . . . . .	5	3.6 Combination . . . . .	40
2.2.1 Nearest Neighbor		<b>4 Conclusion</b>	<b>41</b>
Loss/Chamfer Loss . . . . .	5	<b>A Bibliography</b>	<b>43</b>
2.2.2 Smoothness Loss . . . . .	6		
2.2.3 Laplacian Regularization . . . . .	7		
2.2.4 Cycle Consistency Loss . . . . .	7		
2.3 Models . . . . .	8		
2.3.1 FlowNet3D (Jun 2019) . . . . .	8		
2.3.2 Just Go With the Flow (Dec			
2019) . . . . .	9		
2.3.3 PointPWC-Net (Nov 2019) . . . . .	9		
2.3.4 RAFT (March 2020) . . . . .	11		
2.3.5 Non-Rigid Residual Flow and			
Ego-Motion (Sep 2020) . . . . .	13		
2.3.6 FlowStep3D (Nov 2020) . . . . .	14		
2.3.7 PV-RAFT (Dec 2020) . . . . .	14		
2.3.8 Weakly Supervised Learning of			
Rigid 3D Scene Flow (Feb 2021) . . . . .	15		
2.3.9 SLIM (Oct 2021) . . . . .	16		
2.3.10 RigidFlow (Jun 2022) . . . . .	17		
2.3.11 OGC (Oct 2022) . . . . .	17		
<b>3 Methods</b>	<b>21</b>		
3.1 Datasets . . . . .	21		
3.1.1 Motion Flow label creation . . . . .	21		
3.1.2 Waymo Open Dataset . . . . .	22		
3.1.3 Kitty Raw Dataset . . . . .	23		
3.1.4 NuScenes . . . . .	23		
3.2 Metrics . . . . .	23		
3.3 Model . . . . .	24		
3.3.1 Architecture . . . . .	25		
3.3.2 Losses . . . . .	27		
3.3.3 Baseline . . . . .	28		
3.3.4 Static Point Loss . . . . .	31		
3.4 Leveraging the Temporal Structure			
of the Data . . . . .	32		
3.4.1 Velocity Loss (VC) . . . . .	32		
3.4.2 Dynamic Consistency Loss			
(DC) . . . . .	34		
3.4.3 Experiments . . . . .	34		
3.5 Object Classification Consistency	36		

## Figures

2.1 Visualization for the nearest neighbor loss, where on the left is shown the degenerative solution and on the right, the red errors are optimized. . . . .	6
2.2 Overview for FlowNet3D. . . . .	9
2.3 The cost volume layer from PointPWC-Net [25] . . . . .	10
2.4 Architecture overview of the RAFT optical flow model [22]. . . . .	11
2.5 Building a correlation volumes. Here we depict 2D slices of a full 4D volume. [22]. . . . .	12
2.6 Update block of the RAFT architecture . . . . .	13
2.7 FlowStep3D architecture. . . . .	14
2.8 Pipeline of the PV-RAFT. . . . .	15
2.9 Overview of the pipeline of Weakly Supervised Learning of Rigid 3D Scene Flow . . . . .	16
2.10 Overall view on pipeline incorporated in OGC [20] . . . . .	18
3.1 Visualization of the Waymo dataset with and without the ground . . . . .	22
3.2 Overview of the SLIM architecture	25
3.3 SLIM update block . . . . .	26
3.4 Analysis of endpoint error: Official setup vs. our baseline on Waymo Open Dataset . . . . .	30
3.5 Overall visualization for Time Consistency Loss . . . . .	33
3.6 Behavior of the dynamic object in synchronized pointclouds on the pillars. . . . .	35
3.7 Motion segmentation, where the front portion of the car is inaccurately classified as static (static aggregated flow is used for these points) and the segmentation with Refinement module . . . . .	37
3.8 Cluster pointcloud with DBSCAN [4]; the noise is light green and other colors represent the clusters . . . . .	38

## Tables

3.1 Comparing our implementation with the official on Nusncenes dataset. . . . .	29
3.2 Comparing the models with default and added losses, where we add smoothnes and static point loss; both are trained and tested on the waymo dataset . . . . .	30
3.3 Comparing models trained and tested on Waymo: xyz-only versus xyz with elongation and intensity. .	31
3.4 Results of the models trained on the raw kitti train dataset and tested on Waymo . . . . .	31
3.5 Standard metrics for consistency methods; baseline is trained with 75,000 iterations, while the models with consistency losses are initialized with baseline trained for 25,000 iterations and then trained for 50,000 iterations. . . . .	35
3.6 Separate metrics for dynamic and static parts for consistency methods.	36
3.7 Classical metrics for object consistency methods on Waymo dataset . . . . .	39
3.8 Separate evaluation for dynamic and static points for object consistency methods on Waymo dataset . . . . .	39
3.9 Classical metrics on Waymo dataset for combining loss functions	40
3.10 Distinct metrics on Waymo dataset for dynamic and static parts for models integrating both approaches. . . . .	40





# Chapter 1

## Introduction

The recent advancement in sensor technologies, such as RGB-D cameras and LiDARs, has opened up new fields of research. These advanced sensors enabled the acquisition of three-dimensional data, thereby offering researchers opportunities across diverse disciplines. In the context of this work, our focus is specifically on the areas of motion flow and motion segmentation on 3D pointcloud data. Prior to this field, the research of motion features centered around the analysis of images utilizing optical flow techniques. Optical flow refers to the analysis of object motion and intensity patterns within images or image sequences. Its primary goal is to estimate the displacement vectors of pixels or small image regions between consecutive frames. The estimation of optical flow commonly relies on the fundamental assumption of brightness constancy, which posits that the intensity of a pixel or region remains constant as it moves through consecutive frames. The classical approaches depend on local Taylor series approximations of the image signal or are formulated as hand-crafted optimization problems. However, the rise of deep learning has introduced new possibilities for approaching the optical flow task. A significant breakthrough came with the introduction of the RAFT (Recurrent All Pairs Field Transforms) neural network [22]. This innovative neural network was inspired by conventional optimization-based approaches and offers a clever and efficient method for computing correlations between pixels in two images. This capability enables the network to capture fast-moving objects, occlusions, motion blur, and textureless surfaces [22, 5].

Scene flow (interchangeably used with term *Motion flow*) is a task closely related to optical flow as it pertains to the motion of objects within a three-dimensional scene. The primary goal of scene flow is to determine the displacement vector between points in three-dimensional space. However, it is important to note that scene flow exhibits several drawbacks in comparison to optical flow. One notable challenge arises from the fact that each point cloud derived from a real-world scene contains a varying number of points. Consequently, a bijective projection for scene flow does not exist. However, when the point cloud is projected onto an image plane, the task can be transformed into an optical flow problem [15].

Since the introduction of the first LiDAR automotive Raw Kitti dataset [6] in 2013, the use of LiDAR in research has gained in size, leading to

the development of new approaches for scene flow estimation and motion segmentation on LiDAR data. However, existing automotive datasets lack scene flow labels, posing a significant challenge for supervised learning. To overcome this issue, self-supervised learning approaches have emerged as promising solutions, leveraging the spatial and time consistency in unlabeled data to learn meaningful representations. By designing suitable loss functions that exploit the available information within the data itself, these methods can effectively learn to extract discriminative motion features without requiring explicit supervision.

Although automotive datasets generally lack scene flow labels, it is possible to provide supervision to the models by using artificially generated datasets with labels such as FlyingThing3D [16]. To evaluate the performance of self-supervised models trained on automotive datasets, labels can be created if the dataset contains bounding box annotations and odometry information within two consecutive frames. These soft ground truth labels can be used as motion flow labels, enabling the evaluation of the learned representations in a classical supervised manner.

While the focus of previous research was primarily on the optical flow analysis using images or RGB-D data, classical alignment algorithms such as Kabsch [10], ICP [2], or their improved versions were employed for point cloud alignment, thus for computing the motion flow on the point cloud data. One of the first attempts to directly predict motion flow on point cloud data by deep learning model was made by FlowNet3D [11], which initiated research in this field. The RAFT [22] model had a significant impact on the development of neural networks in this area. Following its release, many models were inspired by the RAFT approach and either built their models directly on it or adopted the idea of incorporating a correlation layer in 3D, such as voxels [24] or directly on points [11]. These models have proven effective in learning to extract informative motion features without the need for explicit supervision, when they are used with appropriate loss functions. Our objective was not to propose a novel neural network, but rather to leverage existing approaches and explore different strategies for addressing the self-supervised scene flow and motion segmentation tasks.

## Chapter 2

### Related Work

The domain of scene flow and the motion segmentation is considered to be in its early stages of development. This chapter aims to outline the chronological order in which various models were introduced. Since the introduction of Flownet3D [15] in 2019, there have been relatively few new models proposed. This self-contained overview assists in comprehending why certain models outperform the others, while also helps us identifying areas where improvements can be made. Throughout this thesis, a consistent notation is utilized for easy and efficient navigation within the equations, images, text and related elements:

- $\mathcal{P}$  denotes a point cloud, accompanied by a lower index representing the time frame of its capture. Additionally, a consistent color scheme is employed:  $\mathcal{P}_{t-2}$  is yellow,  $\mathcal{P}_{t-1}$  is green,  $\mathcal{P}_t$  is blue and lastly  $\mathcal{P}_{t+1}$  is red. We inherit this notation from previous works such as [17].
- $\mathbf{p}$  and  $\mathbf{f}$  represent vectorized form of point coordinates and flow.
- $NN_{\mathcal{P}_{t+1}}(\mathbf{p})$  represent the nearest neighbor function, which find the nearest neighbors of point  $\mathbf{p}$  in pointcloud  $\mathcal{P}_{t+1}$

Other functions or variable are not so common in the thesis, thus their specific definitions will be provided subsequently.

### 2.1 Classical Pointcloud Alignment

Classical point cloud registration methods, such as ICP [2] and Kabsch [10], have been widely used for aligning point clouds by estimating the rigid transformation between them through minimizing the distance between corresponding points. While these methods have a long-standing history and have shown to be effective in aligning point clouds, they do have limitations. For example, they are not suitable for autonomous datasets where the movement in the scene can not be described by a single transformation matrix due to the presence of multiple moving objects and the data are noisy and non-bijective.

We will shortly describe Kabsch and ICP algorithm, where their goal is to minimize the following objective function:

$$\arg \min_{R,t} \frac{1}{|\mathcal{P}_t|} \sum_i \|R\mathbf{p}_i + t - \mathbf{q}_i\|_2^2, \quad (2.1)$$

where  $\mathbf{p}_i$  is  $i$ -th point from point cloud  $\mathcal{P}_t$ ,  $R$  is a rotation matrix and  $t$  is a translation vector. The  $\mathbf{q}_i$  represents the corresponding point from point cloud  $\mathcal{P}_{t+1}$  to point  $\mathbf{p}_i$  in the context of the Kabsch algorithm. However, for ICP, it denotes the nearest neighbor point from point cloud  $\mathcal{P}_{t+1}$  for point  $\mathbf{p}_i$ .

### 2.1.1 Kabsch Algorithm

The older algorithm of the two aforementioned, the Kabsch algorithm is similar to ICP in its goal of aligning two sets of points in Euclidean space. Kabsch requires correspondences between the two point clouds, which can be a significant disadvantage in various applications. Unlike ICP, which can align point clouds without prior correspondences, the Kabsch algorithm relies on a known set of corresponding points in both point clouds, which may not always be available or accurate. The algorithm leverages the singular value decomposition to calculate the optimal rotation and translation, benefiting from the correspondence between the points. The algorithm is described here with the following pseudo-code:

---

#### Algorithm 1 Kabsch Algorithm

---

- 1: **procedure** KABSCH( $\mathcal{P}_t, \mathcal{P}_{t+1}$ )
  - 2:   compute center mass of both pointclouds  $\mu_t, \mu_{t+1}$
  - 3:   shift pointclouds as  $\mathcal{P}_t -= \mu_t$  and  $\mathcal{P}_{t+1} -= \mu_{t+1}$
  - 4:   compute covariance matrix  $\mathcal{H}$  as  $\mathcal{P}_t^T \cdot \mathcal{P}_{t+1}$
  - 5:   apply  $SVD(\mathcal{H}) = U\Sigma V^T$
  - 6:    $R = VU^T$
  - 7:    $t = \mu_t - R\mu_{t+1}$
  - 8:   **return**  $T$
  - 9: **end procedure**
- 

where  $T$  is transformation matrix composed from  $R$  and  $t$ . [10]

### 2.1.2 Iterative Closest Point

ICP is an algorithm commonly used in computer vision and robotics to align two sets of 3D point clouds without correspondences. The vanilla version of the ICP can be described in pseudo-code as:

**Algorithm 2** Iterative Closest Point (ICP) Algorithm

---

```

1: procedure ICP( $\mathcal{P}_t, \mathcal{P}_{t+1}$ )
2:   Initialization  $T$ 
3:   repeat
4:     for all  $p_i \in \mathcal{P}_t$  do
5:       find the closest point  $q_i$  in  $\mathcal{P}_{t+1}$ 
6:     end for
7:     compute center mass for the correspondencies as  $\mu_t$ 
8:     shift the pointcloud as  $\mathcal{P}_t \leftarrow \mu_t$ 
9:     compute covariance matrix  $\mathcal{H}$  as  $\mathcal{P}_t^T \cdot \mathcal{P}_{t+1}$ 
10:    apply  $SVD(\mathcal{H}) = U\Sigma V^T$ 
11:     $R = VU^T$ 
12:     $t = \mu_t - R\mu_{t+1}$ 
13:    apply the transformation to  $\mathcal{P}_t$ 
14:  until converged or maximum number of iterations reached
15:  return  $T$ 
16: end procedure

```

---

Although the ICP algorithm is known for its efficiency in registering point clouds, its performance heavily relies on the quality of the initialization of the rigid transformation and point matching. In recent years, self-supervised models for motion flow have outperformed ICP in terms of scene flow prediction. [2]

## 2.2 Losses

In the following section, we will focus on self-supervised models designed to predict motion flow or motion segmentation. To gain an understanding, why these models are able to work under self supervised manner, we will define the self-supervised losses utilized within these models.

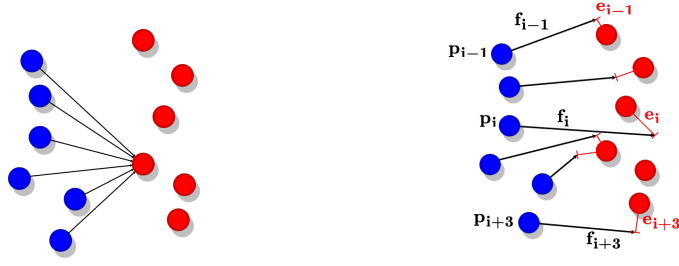
### 2.2.1 Nearest Neighbor Loss/Chamfer Loss

Nearest Neighbor loss is a form of self-supervision signal for flow estimation process. This approach has been previously used in other studies focusing on self-supervised scene flow estimation [23, 1, 17].

In a loss implementation, we assume that pointwise flow  $\mathbf{f}_i$ , represents the flow from  $\mathcal{P}_t$  to  $\mathcal{P}_{t+1}$ . We aim to regularize the predicted flow endpoints with the corresponding ground truth flow endpoints. However, the ground truth labels are not available. To overcome this limitation, a nearest neighbor function is defined as follows

$$NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i) = \operatorname{argmin}_{\mathbf{p}_j \in \mathcal{P}_{t+1}} |\mathbf{p}_j - \mathbf{p}_i| \quad (2.2)$$

where  $\mathbf{p}_i$  represents the point in current pointcloud and  $\mathbf{p}_j$  represents a point from the pointcloud at time  $t + 1$ . This equation provides us the



**Figure 2.1:** Visualization for the nearest neighbor loss, where on the left is shown the degenerative solution and on the right, the red errors are optimized.

nearest point in the subsequent point cloud, which we use as a pseudo label to regularize the flow. To implement the loss function, we need to define an error function as

$$e_i(\mathbf{p}_i, \mathbf{f}_i) = |NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i + \mathbf{f}_i) - (\mathbf{p}_i + \mathbf{f}_i)| \quad (2.3)$$

where the  $e_i$  is the distance between a point with added flow and its nearest neighbor using the nearest neighbor function as described in equation 2.2 or depicted in fig 2.1. The final loss is computed as the mean of all the errors as in equation

$$L_{nn} = \frac{1}{|\mathcal{P}_t|} \sum_i e_i(\mathbf{p}_i, \mathbf{f}_i) \quad (2.4)$$

where  $|\mathcal{P}_t|$  is a number of points in current pointcloud. [1]

As a result of noisy and non-bijective point clouds, errors in flow estimation may be more significant for distant points. Additionally, the loss function used for flow estimation can have multiple degenerate minima, as illustrated in fig 2.1. In extreme cases, all estimated flows may converge towards the same point, resulting in a loss of zero but an incorrect solution. Therefore, additional guidance is required beyond this loss to overcome these limitations in the model.

### 2.2.2 Smoothness Loss

To further enhance the accuracy of the motion flow estimation, a smoothness constraint in space can be introduced. This can be applied not only to predicted flow field in optical flow [26], but also to motion flow on pointclouds [1]. It is achieved by introducing a smoothness loss term that enforces the predicted flow to be locally smooth in space. This loss aims to regularize the flow to be locally homogeneous in space, given the assumption that only rigid objects are present in the scene. Specifically, the smoothness loss in motion flow is formulated as the mean of the L2 losses on differences between flow from point  $p_i$  with  $k$  nearest points and their flows. Loss is defined as

$$L_{smoothness} = \frac{1}{|\mathcal{P}_t|} \sum_i \frac{1}{k} \sum_j^k \|\mathbf{f}_i - \mathbf{f}_j\|_2^2 \quad (2.5)$$

where  $k$  is a constant that determines the number of nearest neighbors, and  $\mathbf{f}_j$  represents the flows corresponding to the closest  $k$  points in the current point cloud.

### 2.2.3 Laplacian Regularization

According to [21], the Laplacian coordinate vector can be utilized to provide an estimation of the local shape properties of the surface. Specifically, the computation of the Laplacian coordinate vector can be used as the approximation of the Laplacian function. Laplacian vector can be carried out as follows:

$$\delta(\mathbf{p}_i) = \frac{1}{k} \sum_j^k (\mathbf{p}_j - \mathbf{p}_i) \quad (2.6)$$

where  $\mathbf{p}_j$  is a point from k-NN neighborhood. The laplacian loss is then computed as

$$L_{laplacian} = \frac{1}{|\mathcal{P}_t|} \sum_i \frac{1}{k} \sum_j^k \|\delta(\mathbf{p}_i + \mathbf{f}_i) - \delta(\mathbf{p}_j + \mathbf{f}_i)\|_1 \quad (2.7)$$

where  $\mathbf{f}_i$  is predicted flow on point  $\mathbf{p}_i$  and  $\mathbf{p}_j$  is point from k-NN neighborhood of point  $\mathbf{p}_i$ . Using this approach, the optical flow is enforced to preserve its Laplacian when the source point cloud is warped based on the predicted flow:

$$\begin{aligned} \mathcal{L}(S + F_k) &\simeq \mathcal{L}(S) \\ &\Downarrow \\ \mathcal{L}(F_k) &\longrightarrow \mathbf{0} \end{aligned} \quad (2.8)$$

where  $\mathcal{L}$  represent Laplacian operator [11, 25].

### 2.2.4 Cycle Consistency Loss

To address the degenerative issues in nearest neighbor loss, the cycle consistency loss was introduced as an additional self-supervised loss in [17]. Another variant was presented with smooth L1 loss (Huber loss) in [15]. The forward flow  $\mathbf{f}_i$  is estimated to the next pointcloud frame. Thus, we define the corresponding (anchored) point in the consentitive frame equally as in equation 2.2

$$\mathbf{p}_j = NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i + \mathbf{f}_i) \quad (2.9)$$

where  $\mathbf{p}_j$  would be the nearest neighbor in the next frame after applying the predicted flow. To predict the backward flow to the current frame, an

anchored point is used as a reference. This flow is marked as  $\hat{\mathbf{f}}_i$ . If both flow are accurate, the flows should be similar with opposite direction. The cycle consistency loss is defined as the mean square error between forward flow and backward flow from anchored point  $\mathbf{p}_j$ . The equation for loss is as defined as

$$L_{cycle} = \frac{1}{|\mathcal{P}_t|} \sum_i \|\mathbf{f}_i + \hat{\mathbf{f}}_i\|_2^2 \quad (2.10)$$

where the loss function have also degenerative solutions wherein the predicted flows are zero, resulting in a zero loss. To mitigate this issue, it is necessary to incorporate the nearest neighbor loss in conjunction with the cycle consistency loss. This is because the nearest neighbor loss is high during such moments and can guide the network away from respective local minima.

## 2.3 Models

### 2.3.1 FlowNet3D (Jun 2019)

This model represents one of the first attempts to directly predict flow on lidar pointclouds, while most previous methods have focused on using stereo and RGB-D images as inputs. The proposed model is capable of learning flow in an end-to-end fashion, which was a significant contribution to the field.

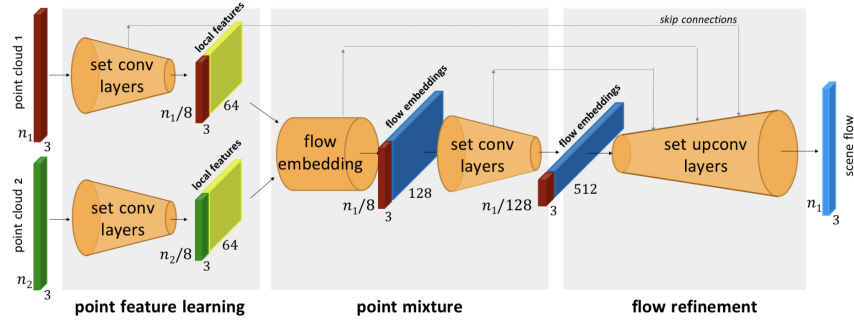
The proposed model consists of three main components: point feature learning, point mixture, and flow refinement. For feature learning, the authors employ a recently proposed model at that time, PointNet++ [19], which is invariant to rigid transformations. This is applied to both pointclouds, and then they are passed to flow embeddings (point mixture) where a novel flow embedding layer is proposed. The architecture of the model is depicted in fig 2.2a

To illustrate the design inspiration of the embedding layer pictured in fig 2.2b, consider a point at time  $t$  and a corresponding point in time  $t+1$ , then its scene flow is simply the relative displacement between them. However, in real lidar data, correspondences between point clouds in two frames don't exist. Despite this, it is still possible to estimate scene flow by finding multiple softly corresponding points in frame  $t+1$  and making a "weighted" decision. This embedding is then passed to the subsequent PointNet++ layers. Finally, the flow embedding is upsampled with up convolution layer to the same size as the input pointcloud. [15]

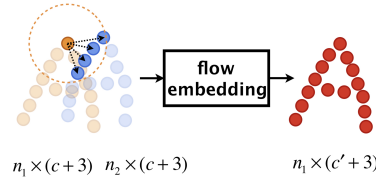
Although this study is one of the first attempts to predict flow directly on pointclouds without any transformation to 2D, they only proposed supervised learning mode. They employ a smooth L1 loss (Huber loss) with ground truth labels, along with the cycle consistency loss for regularization as total loss

$$L_{total} = \frac{1}{|\mathcal{P}_t|} \sum_i \left\{ \|\mathbf{f}_i - \mathbf{f}_i^*\| + \lambda \|\mathbf{f}_i + \hat{\mathbf{f}}_i\| \right\} \quad (2.11)$$





(a) : FlowNet3D architecture [15].



(b) : FlowNet3D flow embeddings layer [15]

Figure 2.2: Overview for FlowNet3D.

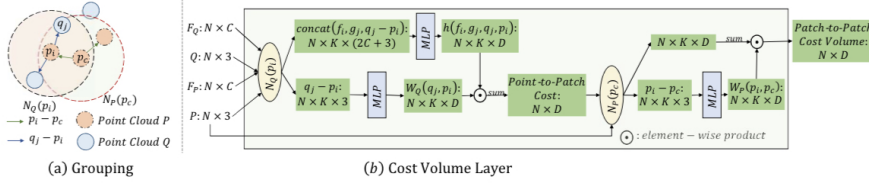
where  $\mathbf{f}_i^*$  is ground thruth flow,  $\mathbf{f}_i$  is predicted flow and  $\hat{\mathbf{f}}_i$  is reversed flow from points  $\mathbf{p}_i + \mathbf{f}_i$ . [15]

### 2.3.2 Just Go With the Flow (Dec 2019)

This paper presents a self-supervised method for training a scene flow network using the FlowNet3D architecture. The authors utilize two losses: cycle consistency and nearest neighbor loss, where the nearest point to the predicted translated points is used as a pseudo-label. They demonstrate that this combination of losses can be used to train a scene flow network over large-scale unannotated datasets. While the model can operate in a self-supervised manner, the authors use pre-trained weights on the synthetic dataset FlyingThings3D [16] to initialize the FlowNet3D model. They then fine-tune it on large-scale automotive datasets such as NuScenes [3] and Raw Kitt [6]. The authors found that the scanned ground (road) in the pointclouds degrades the model’s performance when trained in a self-supervised manner. Lastly, they show that the cycle consistency loss with anchoring works much better than simple cycle consistency loss [17]

### 2.3.3 PointPWC-Net (Nov 2019)

It is the first self-supervised model to predict scene flow on the pointclouds. The authors of this new architecture suggested that the FlowNet3D style of encoding the motion of two consecutive point clouds through their flow embedding layer requires encoding and capturing a large neighborhood in order to record significant motion. The flow embedding layer is calculated in



**Figure 2.3:** The cost volume layer from PointPWC-Net [25]

a single layer and is responsible for capturing the correlation between points, which is then propagated through the network to estimate flow. The paper proposes a new approach using a learnable point-based cost volume, without creating a dense 4D tensor. Additionally, the authors proposed a warping and upsampling layer to estimate flow. [25]

**The cost volume layer.** The process of optical flow estimation often relies on the cost volume technique, however, when applied to point cloud data, the challenge of correlating points becomes an open problem. To address this challenge, the authors propose a novel learnable cost volume layer. This layer takes in two consecutive point clouds,  $\mathcal{P}_t \in \mathbb{R}^{n \times d}$  and  $\mathcal{P}_{t+1} \in \mathbb{R}^{m \times d}$ , along with their corresponding points,  $\mathbf{p}_i \in \mathcal{P}_t$  and  $\mathbf{q}_j \in \mathcal{P}_{t+1}$ . The cost layer for two points is defined as

$$c(\mathbf{p}_i, \mathbf{q}_j) = MLP(\text{concat}(\mathbf{p}_i, \mathbf{q}_j, \mathbf{q}_j - \mathbf{p}_i)) \quad (2.12)$$

where  $p_i$  and  $q_j$  can either represent raw input points or latent space variables from some previous layer. The authors hypothesize that a multi-layer perceptron (MLP) can learn the non-linear relationship between these points. Furthermore, they extend this idea to neighborhood correlation ( $NC$ ), as this approach can be sensitive to outliers or sparse regions in the data. For a point  $\mathbf{p}_i \in \mathcal{P}_t$  they create  $k$ -NN neighborhood area as  $NN(\mathbf{p}_i) \in \mathbb{R}^{k \times c}$ . For each point from  $NN(\mathbf{p}_i)$  denoted as  $\mathbf{p}_j$  they find a  $k$ -NN neighborhood in  $\mathcal{P}_{t+1}$  and same process is repeated for the points from second pointcloud. The neighborhood correlation is defined as

$$NC(p_i) = \sum_{\mathbf{p}_j \in NN(\mathbf{p}_i)} MLP(\mathbf{p}_j - \mathbf{p}_i) \sum_{\mathbf{q}_i \in NN(\mathbf{p}_j)} MLP(\mathbf{q}_i - \mathbf{p}_j) c(\mathbf{q}_i, \mathbf{p}_j) \quad (2.13)$$

where the MLPs are weighted w.r.t. the direction of the vectors that that are used to aggregate the costs from the neighborhood in the two pointcloud [25].

**Losses.** The authors presents the first self-supervised model designed to learn scene flow from unannotated data. The model employs three types of loss functions: Chamfer distance known as Nearest Neighbor loss, Smoothness loss, and Laplacian Regularization. The authors conducted an analysis to investigate the interactions between these three loss functions. Results showed

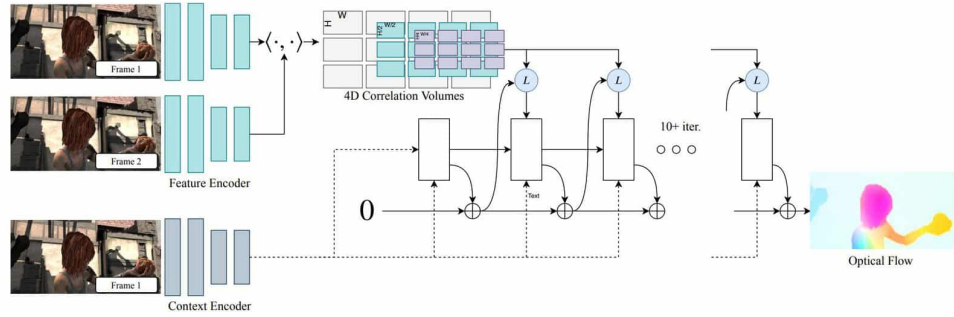


Figure 2.4: Architecture overview of the RAFT optical flow model [22].

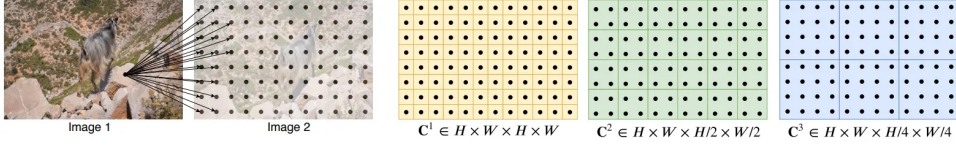
that while the model is capable of learning solely with the Chamfer loss, its performance in estimating reliable flow is inadequate. By incorporating the Smoothness loss, the model’s performance improved by 38.2%. Furthermore, the addition of the Laplacian regularization slightly enhanced the model’s results. [25]

### 2.3.4 RAFT (March 2020)

The Recurrent All-Pairs Field Transformations (RAFT) is a state-of-the-art deep network architecture for solving the optical flow problem in images. In traditional approaches, optical flow was treated as an optimization problem over dense displacement fields between a pair of images using hand-crafted methods, which achieved good results but faced challenges in further improvement. With the rise of deep learning, such methods are being replaced by end-to-end trainable neural networks that reformulate the optimization problem. RAFT consists of three main components: a feature encoder, a correlation layer, and a GRU update block. The architecture can be seen at fig 2.4. We are mentioning this network because the principles are then used in the following point clouds models.

**Feature encoder.** has shared weights for both images. It consists of 6 residual blocks that extract per-pixel features from the images. The output of the encoder is at 1/8th of the resolution of the original image. In addition to the Feature Encoder, there is a Context Encoder, which is only applied to the first image. The architecture of the Context Encoder is similar to that of the Feature Encoder. The features extracted from the Context Encoder are directly injected into the update block. The authors suggest that this helps improve the aggregation of spatial information within motion boundaries.

**Correlation Layer.** Computation the visual similarity as a correlation volume between all pairs. The correlation field represents the relationship between two pixels from two images. The correlation volume is done as dot product between all pairs from the two features image, this results in tensor in shape  $H \times W \times H \times W$  defined by



**Figure 2.5:** Building a correlation volumes. Here we depict 2D slices of a full 4D volume. [22]

$$C_{ijkl} = \sum_h g_\theta(I_1)_{ijh} \cdot g_\theta(I_2)_{klh} \quad (2.14)$$

where  $I_1, I_2 \in \mathbb{R}^{H \times W \times D}$  are the first and second image and  $g_\theta(-)$  is a function of the feature encoder. The principle is depicted in fig 2.5.

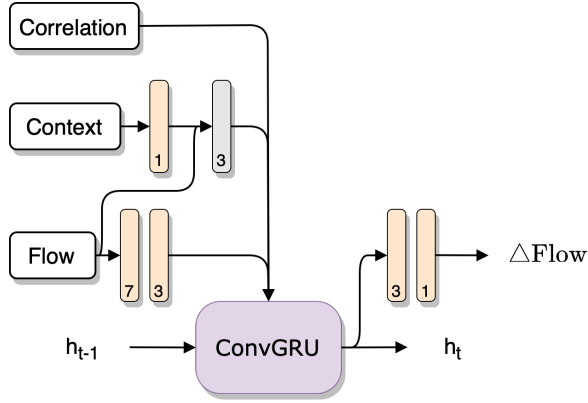
The approach involves constructing a pyramid of correlation volumes  $\{C^1, C^2, C^3, C^4\}$  with 4 layers by pooling the last two dimensions of the correlation volume with kernel sizes 1,2,4, and 8 and equivalent stride, where  $C^2$  represents the correlation between 2x2 patches of the images. By using this pyramid, the method is able to capture both large and small displacements, which is important for accurately estimating optical flow for fast moving objects.

For usability the correlation pyramid in update block, they proposed the correlation lookup operator to generate a feature map by indexing from a multi-level correlation pyramid. The operator maps each pixel in the first image to its estimated correspondence in the second image, and defines a local grid around it to index from the correlation volume using bilinear sampling. The lookup is performed on all levels of the pyramid with a  $\frac{1}{2^k}$  of the first width, where  $k$  is index of correlation volume and the resulting values are concatenated into a single feature map. [22]

**Update Block.** is a computational unit that estimates a sequence of flow, denoted as  $\{\mathbf{f}_1, \mathbf{f}_2 \dots \mathbf{f}_N\}$ , based on a predetermined number of iterations, given an initial flow of zero. As depicted in fig 2.6, the block takes in context features, correlation features obtained from correlation lookup, and the previous flow. The inputs are then processed through convolutional layers and concatenated before being passed into a gated recurrent unit (GRU) cell. The GRU produces the next hidden state and  $\Delta \mathbf{f}_i$ , which is added to the previous flow. This process is repeated for a total of six iterations before the final flow is upsampled to match the original image dimensions [22].

**Supervision.** They proposed the architecture as supervised model for optical flow, where the use  $L_1$  distance between the predicted and ground truth flow over the full sequence of predictions from update block with exponentially increasing weights. The final loss is defined as

$$L_{total} = \sum_i \gamma^{N-i} \|\mathbf{f}_{gt} - \mathbf{f}_i\|_1 \quad (2.15)$$



**Figure 2.6:** Update block of the RAFT architecture

where they set  $\gamma = 0.8$  to penalize more the final prediction than the sub-results. Although the model in question was not originally designed for motion flow and was only supervised, its outstanding performance has served as a source of inspiration for the development of a subsequent self-supervised model aimed at motion flow prediction. [22]

### 2.3.5 Non-Rigid Residual Flow and Ego-Motion (Sep 2020)

This paper presents a novel approach for estimating motion flow on pointclouds. The proposed method decomposes the flow into two components: ego motion flow and residual flow, which captures the motion of dynamic rigid objects in the scene. This approach differs from previous works that aimed to learn a single flow for static points and dynamic objects.

The proposed method employs a network that learns the rigid motion between two pointclouds and is subsequently followed by an iterative refinement process. The authors demonstrate that their model can be trained in both supervised and self-supervised manners. While their model outperformed state-of-the-art supervised models, their self-supervised approach did not yield comparable results. [23]

Once their model (relative pose regressor) produce a rough estimate of odometry, the authors synchronize the two pointclouds and begin to estimate the non-rigid flow of dynamic objects. Their rationale is that when the pointclouds are synchronized, the learning of non-rigid flow should be easier for the model. For this part, they employ HPLFlowNet [8], a state-of-the-art supervised total flow learning approach in that time. In the self-supervised model, the authors utilize two loss functions: the nearest neighbor loss and the cycle consistency loss. These two losses are combined to form the total loss:

$$L_{total} = L_{nn} + L_{cc} \quad (2.16)$$

where  $L_{nn}$  is nearest neighbor loss and the  $L_{cc}$  is cycle consistency loss. [23]

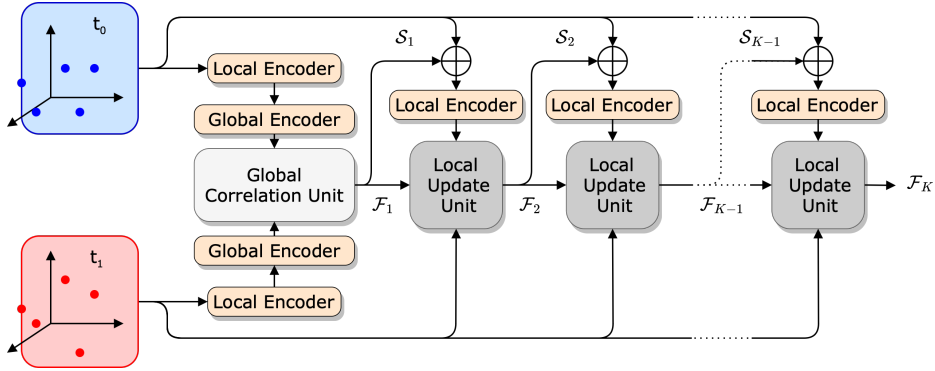


Figure 2.7: FlowStep3D architecture

### 2.3.6 FlowStep3D (Nov 2020)

FlowStep3D is a novel model that draws inspiration from the RAFT and extends this idea to scene flow. They changed the correlation layer from RAFT to operate directly on 3D points. The authors also propose to enrich the pipeline with a mechanism that computes new features of the warped point cloud at each iteration of the update block. This addition is considered crucial due to the inherent lack of invariance to rotation exhibited by existing point cloud convolution methods, which results in changing features as the cloud is rotated towards the target point cloud. They also compute the global correlation once to initialize the flow in the recurrent pipeline. [11]

In addition, the authors propose to train the aforementioned model in a self-supervised manner using two loss functions. The first loss function employed is the nearest neighbor loss. The second loss function utilized is the Laplacian loss, which serves as a regularization technique for the flow in the surrounding area of the corresponding point. The overall loss is the linear combination of this two losses:

$$L_{total} = \alpha L_{nn} + L_{laplacian} \quad (2.17)$$

where  $L_{nn}$  and  $L_{smooth}$  are nearest neighbor loss and laplacian loss. [11]

### 2.3.7 PV-RAFT (Dec 2020)

This model extends the concept of RAFT to Point-Voxel Correlation. Due to the unordered nature of point clouds, efficiently identifying neighboring points is challenging. Prior methods such as FlowNet3D and PointPWC [15, 25] only considered nearby neighborhoods, which proved inadequate for fast movement. To address this issue, the authors combined point neighborhood information obtained through k-nearest neighbor search with voxel neighborhood information obtained through a correlation pyramid on voxels, enabling the capture of larger portions of the scene [24].

Model employs the Pointnet++ [19] architecture as the feature extractor, while also the Pointnet++ was used for the context encoder of the first

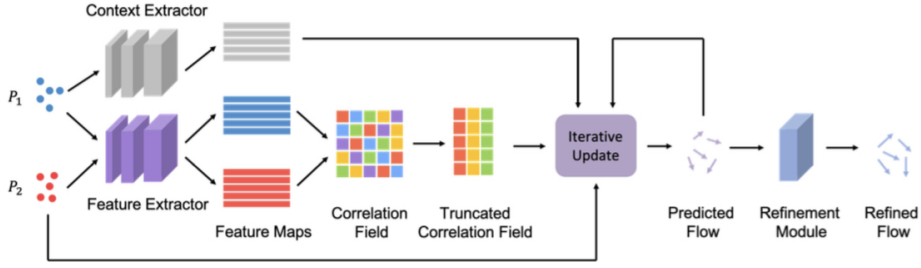


Figure 2.8: Pipeline of the PV-RAFT

point cloud. The correlation pyramid was constructed in a similar fashion as in RAFT, except that it was 3D-volumed due to the use of voxels. The update block was initialized with zero flow, and all key components of the architecture were retained. The architecture is depicted in fig 2.8. However, the correlation features were obtained by combining the voxel-based and point-based pyramids. The flow was refined to be smooth in 3D space using two convolutional layers and one fully connected layer. The training process was divided into two phases. First, the feature extractor and backbone were trained, followed by the training of the refinement module. Notably, the authors did not explore the extension of the model to self-supervised learning [24].

### 2.3.8 Weakly Supervised Learning of Rigid 3D Scene Flow (Feb 2021)

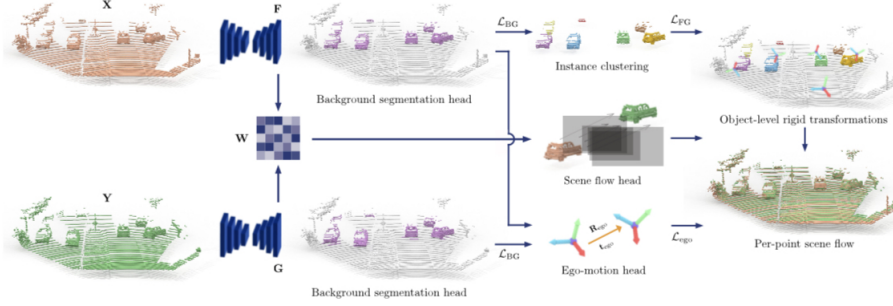
This weakly supervised model requires ground truth odometry and labels for foreground and background point segmentation. Although this model is only weakly supervised, it introduces an interesting idea: after dividing the pointclouds into foreground (all moving objects in the scene) and background, they cluster the foreground into  $N$  different homogeneous rigid objects and try to find rigid transformations for them independently, as shown in fig 2.9. The model uses Minkowski Net as the backbone followed by U-Net with skip connections with shared weights for both pointclouds. Then, three heads are incorporated: Background segmentation head, ego motion head, and scene flow head.

They introduced three losses to regularize the training of the model. For background segmentation, which is the first step in the pipeline, they use binary cross-entropy loss for both pointclouds. The background segmentation loss is defined as

$$L_{bg} = \frac{1}{|\mathcal{P}_t|} \sum_i \text{BCE}(l_{cls,i}, l_{gt,i}) \quad (2.18)$$

where  $l_{cls,i}$  is prediction for background label and  $l_{gt,i}$  is a ground truth class for following class. Segmentation loss is then defined as

$$L_{seg} = \frac{1}{2} (L_{bg}(\mathcal{P}_t) + L_{bg}(\mathcal{P}_{t+1})) \quad (2.19)$$



**Figure 2.9:** Overview of the pipeline of Weakly Supervised Learning of Rigid 3D Scene Flow

where  $L_{seg}$  is final loss. For ego motion learning they are using weighed Kabsch algorithm [10], which results in loss

$$L_{ego} = \frac{1}{|\mathcal{P}_t|} \sum_i \|(R\mathbf{p}_i + \mathbf{t}) - (R_{gt}\mathbf{p}_i + \mathbf{t}_{gt})\|_1 \quad (2.20)$$

where  $\mathcal{R}, \mathbf{t}$  is rotation matrix and translation vector from odometry.

The authors also proposed an instance-level rigidity error, which computes the per-instance rigid loss. To achieve this, they employed the DBSCAN clustering algorithm [4] to divide the points from foreground into instances. For each instance, the transformation matrix is computed using the Kabsch algorithm. This results in a loss per cluster that is defined as the difference between the transformed instance and its corresponding flow. The loss per instance is defined as

$$L_{instance} = \frac{1}{|\mathcal{C}_{t,i}|} \sum_k \|(R\mathbf{p}_k + \mathbf{t}) - (\mathbf{p}_k + \mathbf{f}_k)\| \quad (2.21)$$

where  $\mathbf{f}_k$  is predicted flow,  $\mathcal{R}, \mathbf{t}$  is a rigid transformation from Kabsch and  $\mathcal{C}_{t,i}$  represent the cluster  $i$  from time  $t$ . This weakly supervised approach has been demonstrated to outperform the state-of-the-art motion flow networks at that time. [7]

### ■ 2.3.9 SLIM (Oct 2021)

In this model, the authors were motivated by the potential benefits derived from the disparity between ego motion and raw flow predicted by neural network and also get inspiration from the RAFT [22], which is also part of the model. To address the disparity, they propose a novel model capable of predicting motion flow and motion segmentation. Notably, this represents the first instance of a self-supervised model with the ability to predict motion segmentation. [1]

The model is utilized in the experiments, thus it is described in detail in section 3.3.



### ■ 2.3.10 RigidFlow (Jun 2022)

The authors of RigidFlow proposed an alternative method to previously mentioned, which posits that a real-world scene can be effectively represented as an aggregation of rigidly moving regions. Drawing inspiration from this notion, they developed a technique for generating pseudo labels through piecewise estimation of rigid motion. This technique involves breaking down the current pointcloud into individual regions, with each region being treated as an independent rigid component. Their proposed pipeline involves the generation of pseudo labels for self-supervised learning, which are then used to train the model using traditional supervised losses. [14]

The authors claim that conventional self-supervised learning techniques rely on pointwise similarity measures, which are prone to inconsistency in capturing potential motion flow. To address this issue, the authors propose an oversegmentation approach, where the current pointcloud  $\mathcal{P}_t$  is divided into supervoxels, which are treated as rigid components. By analyzing the movement of each supervoxel, the authors attempt to determine the corresponding rigid transformation. From the rigid transformation they are able to find the flow for current supervoxel as

$$\mathbf{f}_i = (T_{t \rightarrow t+1} - I_4)\mathbf{p}_i \quad (2.22)$$

where  $T_{t \rightarrow t+1}$  is a found rigid transformation. By combining of flows from all supervoxels, they obtain the flow for all points in the current pointcloud  $\mathcal{P}_t$ . [14]

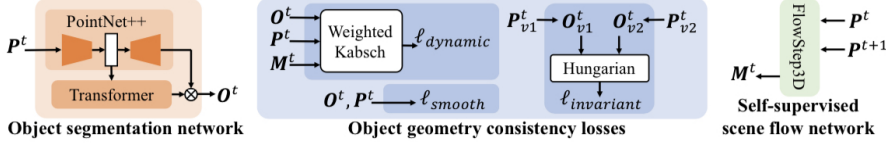
To determine the rigid transformation, the authors employ the iterative closest point (ICP) algorithm [2]. Nevertheless, the success of ICP relies on good initialization, which the authors tackle by leveraging a trained neural network. Subsequently, the authors obtain the final rigid transformation after a few iterations of the ICP algorithm. Once the pseudo labels are generated, the authors train a neural network using a loss function defined as follows

$$L_{sup} = \frac{1}{|\mathcal{P}_t|} \sum_i \|\mathbf{f}_i - \mathbf{f}_{i,ps}\|_1 \quad (2.23)$$

where the  $\mathbf{f}_{i,ps}$  is a pseudo label for the corresponding flow. For their experiments they chose a neural network FLOT [18] as they were able to achieve the state-of-the-art performance among self-supervised predictors of motion flow. [14]

### ■ 2.3.11 OGC (Oct 2022)

OGC represents a new approach to 3D instance segmentation that is distinct from prior models in that it is the first self-supervised method. Although it is not explicitly designed to improve motion flow, the authors utilize the dynamic motion of objects over sequential data to enhance object segmentation. However, the motion flow can also benefit from the object segmentation. The approach involves a pipeline composed of three main components: an



**Figure 2.10:** Overall view on pipeline incorporated in OGC [20]

object segmentation network that estimates multi-object masks from a single point cloud frame, a self-supervised motion flow model, and an object geometry consistency component, where the all optimization is done. By combining these components, OGC can achieve state-of-the-art performance in self-supervised 3D object segmentation. [20]

In the fig 2.10, can be seen, that the object segmentation module employs PointNet++ [19] for the extraction of per-point features. These features are subsequently utilized in conjunction with the Transformer decoder to generate an object mask. We denote the object mask for point cloud  $\mathcal{P}_t$  as  $\mathcal{O}_t$ . The pipeline involves the generation of flow between two consecutive point clouds  $\mathcal{P}_t, \mathcal{P}_{t+1}$ . The resulting flow, denoted as  $\mathcal{M}_t$  to followed the same notatin as in fig 2.10. The authors employ the FlowStep3D model [11] for flow prediction. To train the flow predictor and object segmentator, the autors proposed three losses to work together. [20]

**Geometry Consistency over Dynamic Object Transformations.** The initial predictions for mask  $\mathcal{M}_t$  have meaningless information and requires optimization. The proposed principle follow the loss presented in 2.3.8. Assuming the rigidity of all objects, the motion of each instance (based on the segmentation mask) can be characterized by a rigid transformation. To compute the transformation matrix for each object, the authors employ a differentiable weighted Kabsch algorithm. This algorithm determines the transformation between  $\mathcal{P}_t$  and  $\mathcal{P}_t + \mathcal{M}_t$ , considering the weights  $\mathcal{O}_t$ . The loss function is then defined as

$$L_{dynamic} = \frac{1}{|\mathcal{P}_t|} \sum_i \left\| \left( \sum_k o_{ik} (T_{ik} \cdot \mathbf{p}_i) \right) - (\mathbf{p}_i + \mathbf{f}_i) \right\|_2 \quad (2.24)$$

where  $o_{ik}$  and  $T_{ik}$  represents the object mask and rigid tranformation of the  $k^{th}$  cluster. This loss function must be employed in conjunction with another loss function, because one of the solutions may result in the oversegmentation of the pointclouds, which is an undesiarable local minima [20].

**Geometry Smoothness Regularization.** To address the issue of oversegmentation, the authors implemented a smoothing technique by employing KNN spherical queries instead of the classical KNN approach. By performing these queries, they were able to select all points within a given neighborhood. The goal of this loss function is not to smooth the flow, but rather to ensure the consistency of the mask  $\mathcal{O}_t$  in the space. [20]

**Geometry Invariance over Scene Transformation.** Lastly, they proposed a loss function aimed at enhancing the generalization of predictions. While the aforementioned two loss functions can effectively segment dynamic objects independently, they will have difficulties with similar objects exhibiting zero motion.

Two distinct rigid transformations are applied to the point clouds  $\mathcal{P}_t$  in order to generate two augmented point clouds  $\mathcal{P}_{t,v_1}$  and  $\mathcal{P}_{t,v_2}$ . Subsequently, these augmented point clouds serve as an input into an object segmentation network, resulting in masks  $\mathcal{O}_{t,v_1}$  and  $\mathcal{O}_{t,v_2}$ . The utilization of the Hungarian algorithm [12] creates a one-to-one matching of individual masks within  $\mathcal{O}_{t,v_1}$  and  $\mathcal{O}_{t,v_2}$ . As a consequence, the masks are reordered to  $\hat{\mathcal{O}}_{t,v_1}$  and  $\hat{\mathcal{O}}_{t,v_2}$  to have same points in the same indexes. These reordered masks are subsequently employed in the calculation of an invariant loss function as

$$L_{invariant} = \frac{1}{|\mathcal{P}_t|} \sum_i d(\hat{o}_{t,i,v_1}, \hat{o}_{t,i,v_2}) \quad (2.25)$$

where the functions  $d(-)$  represents L1, L2 or cross entropy loss. This loss motivates the estimation of an object mask that the prediction should be invariant with different views on the point cloud  $\mathcal{P}_t$  [20].



# Chapter 3

## Methods

### 3.1 Datasets

We mainly focused on predicting motion flow in real-world scenarios, specifically targeting the automotive domain. Therefore, we selected the Waymo Open Dataset [9], NuScenes [3], and Raw Kitty [6] as our primary datasets. Notably, we intentionally refrained from utilizing commonly employed artificial datasets such as FlyingThings3D [16], as our focus was on addressing challenges associated with automotive applications of motion flow prediction.

However, the automotive datasets are large scale established dataset, they do not have in majority a scene flow label, therefore the label needs to be created artificially. Additionally, most of the approaches to predict motion flow from real scenes suffers from scanned ground. We followed the commonly used trick to remove the ground from the scene by naively cropping the scene in  $z$  coordinates over threshold.

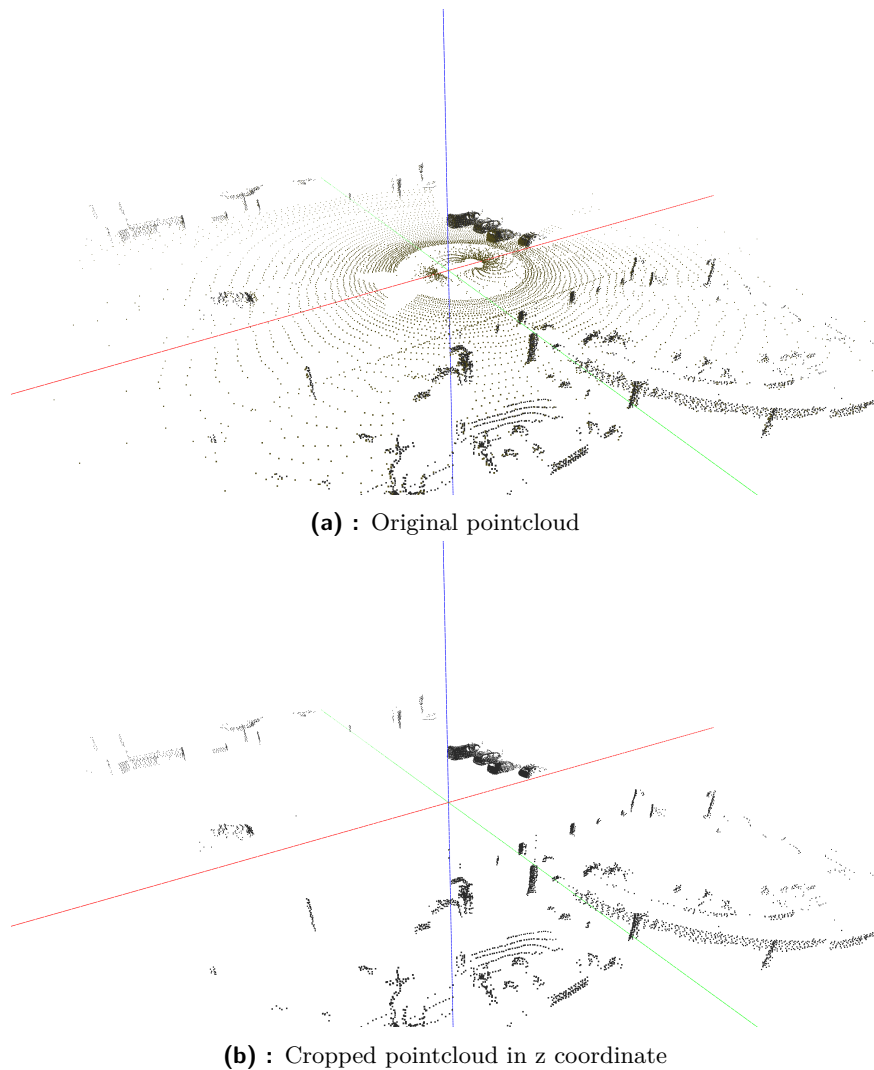
#### 3.1.1 Motion Flow label creation

The process of label creation involved utilizing 3D bounding boxes equipped with unique identifiers for every individual point within the point cloud. This approach relied on the presence of existing labeled and tracked objects within the LiDAR data sequences. Initially, the static flow, resulting from the ego's movement is computed as

$$\mathbf{f}_i = (T_{t \rightarrow t+1} - I_4)\mathbf{p}_i \quad (3.1)$$

for each point from  $\mathcal{P}_t$ .

By assuming that labeled objects are rigid, we can leverage the 3D bounding boxes to circumvent the pointwise correspondence problem between the two consecutive frames for the dynamic object in the scene. We synchronize the two consecutive pointclouds and the flow for the dynamic objects is computed as a displacement of the corresponding bounding box over the time duration. The approach have drawbacks as that we assumed that each object is rigid or it is captured in the previous frame. Also, some rare moving objects do not have bounding boxes and are belonging to the "background" class without movement. [9]



**Figure 3.1:** Visualization of the Waymo dataset with and without the ground

### ■ 3.1.2 Waymo Open Dataset

Waymo Open Dataset for motion flow is a large-scale dataset containing high-resolution LiDAR pointclouds captured in diverse urban and suburban environments. The dataset contains 130,272 train samples and 6368 validation frames, which were used as test samples.

The data were captured with five LiDAR sensors at 10 frames per second with 360-degree view. The scanning vehicle was equipped with one mid-range sensor on the top and four short range on the front, left side, right side, and rear of the vehicle. The original pointcloud and the cropped pointcloud used for train are pictured in fig 3.1.

### ■ 3.1.3 Kitty Raw Dataset

The KITTI raw dataset is a well-known dataset for autonomous driving research, which comprises a large number of synchronized sensor measurements, including high-resolution images, 3D point clouds, and GPS/IMU data. The dataset was collected in and around the city of Karlsruhe in Germany, where the sequences were captured at various times of the day and under different weather conditions, which provide a diverse set of driving scenarios. Nevertheless, the raw dataset used in our study lacks bounding box annotations. As a result, we restricted the usage of this dataset only for training phase. We are aiming to assess the domain transfer performance between two distinct datasets captured using different lidar sensors. This shows the model’s ability to generalize across varying sensor characteristics. [6]

### ■ 3.1.4 NuScenes

NuScenes dataset is a highly comprehensive dataset aimed at autonomous driving research. It provides a diverse range of real-world traffic scenarios recorded from various sensors, such as cameras, lidars, and radars. The authors collected a total of 1000 driving scenes in Boston and Singapore, which are well-known for their dense traffic and challenging driving situations.

The pointclouds were scanned using a 32 beam LiDAR sensor at 20 Hz. This leads to a domain shift in sparsity of the data and distribution shift with comparison with KITTI or Waymo Open Dataset.

Despite providing high-quality sensor data the original NuScenes dataset does not include point-level motion flow labels. Nevertheless, we followed the same principle as for Waymo Open dataset, where we used the approach described in section 3.1.1. [3]

## ■ 3.2 Metrics

We used the established evaluation metrics in order to compare performance. In the itemized list of the metrics, we used the  $ee_i$  as an endpoint error in cartesian space for corresponding flow  $\mathbf{f}_i$ .

- **AEE** - The endpoint error (EE) is calculated across all points. It is used as the primary metric and the calculation is

$$AEE = \frac{1}{N} \sum_{i=0}^N \|\mathbf{f}_i - \mathbf{f}_{gt_i}\|_2 = \frac{1}{N} \sum_{i=0}^N ee_i \quad (3.2)$$

- **AccS** - measures the ratio of points where the EE is less than 0.05 or the relative error is less than 0.05 m

$$AeeS = \frac{1}{N} \sum_{i=0}^N \left( [ee_i < 0.05] \vee \left[ \frac{ee_i}{\|\mathbf{f}_{gt_i}\|_2} < 0.05 \right] \right) \quad (3.3)$$

- **AccR** - measures the ratio of points where the EE is less than 0.1 or the relative error is less than 0.1.

$$AeeR = \frac{1}{N} \sum_{i=0}^N \left( [ee_i < 0.1] \vee \left[ \frac{ee_i}{\|\mathbf{f}_{gt_i}\|_2} < 0.1 \right] \right) \quad (3.4)$$

- **Outl** - measures the ratio of points where the EE is greater than 0.3 or the relative error is greater than 0.1

$$Outl = \frac{1}{N} \sum_{i=0}^N \left( [ee_i > 0.3] \vee \left[ \frac{ee_i}{\|\mathbf{f}_{gt_i}\|_2} > 0.1 \right] \right) \quad (3.5)$$

- **ROutl** - measures the the ratio of points where the EE is greater than 0.3 and the relative error is greater than 0.3

$$ROutl = \frac{1}{N} \sum_{i=0}^N \left( [ee_i > 0.3] \wedge \left[ \frac{ee_i}{\|\mathbf{f}_{gt_i}\|_2} > 0.3 \right] \right) \quad (3.6)$$

We also adopted a similar approach as presented in SLIM [1], wherein the performance evaluation was conducted separately for static and dynamic points. This division was motivated by the substantial imbalance observed between the static and dynamic components within each dataset. To mitigate this issue, they introduce a threshold, denoted as  $m_{thresh}$ , to divide the ground truth flow for static and dynamic. More precisely, ground truth flow is subtracted with flow from odometry as

$$\mathbf{f}_{stat/dyn} = \mathbf{f}_{gt} - (T_{t \rightarrow t+1} - I_4) \mathbf{p}_i \quad (3.7)$$

$$\mathbf{f}_{stat/dyn} = \begin{cases} dynamic & \mathbf{f}_{stat/dyn} > 0.05 \\ static, & otherwise \end{cases} \quad (3.8)$$

where values  $\mathbf{f}_{stat/dyn}$  exceeding  $m_{thresh} = 5\text{cm}$  are labeled as dynamic and conversely. The threshold corresponds to a velocity of  $1.8 \frac{km}{h}$ . Furthermore, they introduced a metric  $AEE_{50-50}$ , which represents the mean of AEE computed for static and dynamic flow.

### 3.3 Model

For our experiments we choose the SLIM framework [1] which shows good results and incredible generalizability across the dataset. Furthermore, the model has the capability to simultaneously predict flow and perform motion segmentation within a single forward pass and effectively integrating these two components.



### 3.3.1 Architecture

The network is composed from three components: Point Cloud Encoder, Flow backbone and final output Decoder.

**Point Cloud Encoder.** Initially, the input pointclouds  $\mathcal{P}_t$  and  $\mathcal{P}_{t+1}$  are cropped to a square of  $|x, y| \leq 35m$ , where  $x$  and  $y$  represent the horizontal axes. This cropping step removes the farthest part of the scene, where reasoning becomes challenging due to sparse data.

We create a pseudo image with dimensions of 640 pixels by 640 pixels, this transformation represents the discretization of pointclouds into the pillars, as introduced in the PointPillars [13]. This discretization corresponds to a pillar size of approximately 11 centimeters. The centers of each pillar are then computed. Subsequently, each cropped pointcloud is encoded into 6D embeddings to pseudo image, consisting of  $\{PC_x, PC_y, PC_z, OF_x, OF_y, OF_z\}$ , where  $PC$  denotes the coordinates of the center of the pillar where the points correspond, while  $OF$  represents the offset from this center in each coordinate. An alternative approach involving a 8-dimensional embedding, where the additional features are an intensity and elongation, this embeddings was also explored. [1]

The resulting pseudo images are processed by the Pillar Feature Network, resulting in Pillar Embeddings denoted as  $\mathcal{I}_t, \mathcal{I}_{t+1} \in \mathbb{R}^{64 \times 640 \times 640}$ . The Pillar Feature Network consists of a simple linear layer followed by batch normalization and rectified linear unit (ReLU) activation. [1]

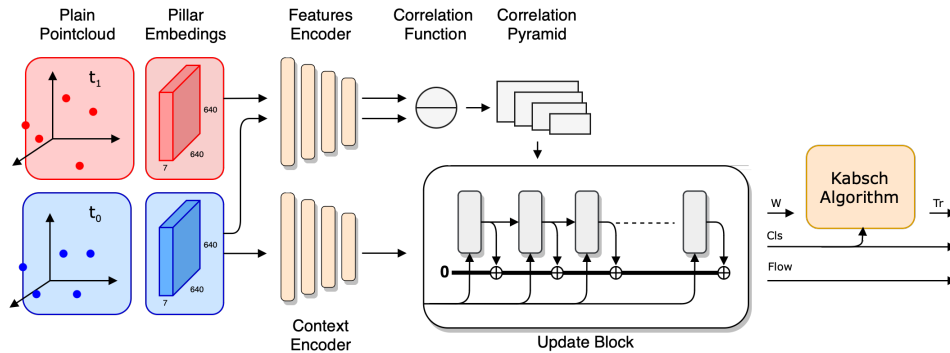


Figure 3.2: Overview of the SLIM architecture

**Flow Backbone.** By transforming the point cloud data into pillar embeddings, the authors were able to directly apply the RAFT [22] architecture to the  $\mathcal{I}_t, \mathcal{I}_{t+1}$  pillars embeddings.

Firstly, each pillar embedding is processed by a Feature Encoder, resulting in a latent space representation of the embeddings denoted as  $\mathcal{A}_t$  and  $\mathcal{A}_{t+1}$ , both of which have dimensions of  $\mathbb{R}^{128 \times 80 \times 80}$ . The Feature Encoder is constructed by composing three residual blocks followed by a convolutional layer. Instance normalization is used as the normalization layer. It is important to note that both  $\mathcal{I}_t$  and  $\mathcal{I}_{t+1}$  (the pseudo images) are processed by the same encoder, meaning that the encoder shares its weights between the two inputs.

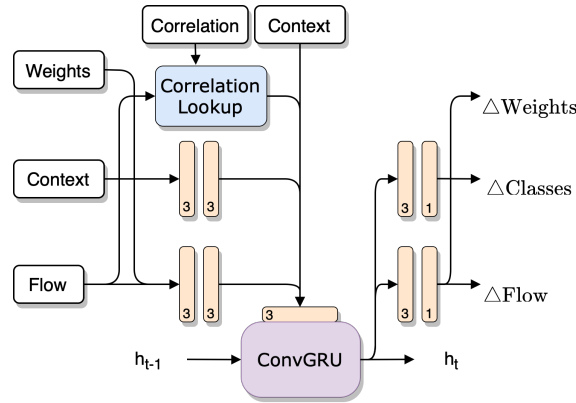


Figure 3.3: SLIM update block

The Context Encoder, in contrast to the Feature Encoder, takes only the  $\mathcal{I}_t$  pillar embeddings as input. The architecture of the Context Encoder is identical to that of the Feature Encoder, with the only difference being the exclusion of a normalization layer.

The correlation function and its implementation follow a same approach as described in Section 2.3.4. A crucial component of the backbone architecture is the update block, which is similar to the one depicted in fig 2.6 of the RAFT architecture.

To handle motion segmentation and generate weights for weighted Kabsch, two new signals are incorporated into the update block depicted in fig 3.3. The input to the Kabsch algorithm can be weighted based solely on the static class probabilities obtained from the softmax operation. However, the accuracy of predicted flow can vary significantly in a scene, particularly when dealing with featureless surfaces that are not suitable for flow estimation. To address this issue, the authors introduce the weights logits. They show that this improve the performance of predicting the odometry in their experiments. All logits and flow are initialized to zero at the beginning of the new input passed to the model. [1]

**Output Decoder.** After six iterations of the update block, the resulting tensor has a shape of [batch size, 7, 80, 80], where the last two dimensions correspond to the dimensions of the pillars. In order to achieve a consistent shape for the output, RAFT applies bilinear upsampling to upscale the tensor into the shape [batch size, 7, 640, 640]. The tensor consists of seven channels, which are interpreted as follows: two channels for motion segmentation logits (static and dynamic), four channels for the two flow components (static and dynamic), and one channel for the weights.

The segmentation logits are passed into softmax layer to normalize the prediction of each class. We referred these normalized logits as staticness and dynamicness. To prepare the weights for Kabsch algorithm, the staticness is masked based on whether the pillar in staticness was occupied by any point or not. Next, the sigmoid activation function is applied to the weights, to maps it into a range between 0 and 1. These weights are then multiplied

element-wise with the masked staticness and the tensor is normalized to sum up to 1. This tensor results in the final weights passed into Kabsch algorithm [1].

The Kabsch algorithm operates in a manner similar to the description provided in Section 2.1.1, but with the key difference that each point in the calculation is weighted, so as an input the algorithm takes the point-wise static flow, pointcloud  $\mathcal{P}_t$ , along with the corresponding point-wise weights. By incorporating the weights during the Kabsch transformation, the algorithm effectively takes into account the importance or significance of each point in the static flow, resulting into a single static aggregated flow that represents the vehicle’s odometry and minimizes the objective as

$$\mathcal{T}_{t \rightarrow t+1} = \operatorname{argmin}_{T \in \mathbb{R}^{4 \times 4}} \sum_i w_i \|(T - I_4)\mathbf{p}_i - \mathbf{f}_i\|^2 \quad (3.9)$$

where  $w_i$  is computed weight and  $\mathbf{f}_i$  corresponding static flow. It should be noted that the objective function employed in the minimization process specifically focuses on minimizing the static flow, while excluding the current point cloud  $\mathcal{P}_{t+1}$  from consideration. Consequently, there is a potential risk of obtaining a static flow value of zero, which in turn would result in a zero odometry estimate.

All outputs are transformed into point-wise representation. The final flow denoted as aggregated flow is composed from static aggregated (flow from computed estimated odometry) and the raw flow (dynamic flow from raft). We decide which flow to take based on the comparison the dynamic logits after softmax compared to classification threshold. The aggregated flow is

$$\mathbf{f}_{\text{aggr}_i} = \begin{cases} (\mathcal{T}_{t \rightarrow t+1} - I_4)\mathbf{p}_i & \text{if } cls_{dyn} \geq p_{stat} \\ \mathbf{f}_{raw_i} & \text{if } cls_{dyn} < p_{stat} \end{cases} \quad (3.10)$$

where  $cls_{dyn}$  is dynamic logits after softmax and  $p_{stat}$  is classification threshold. The classification threshold is an online mechanism for iteratively adjusting the threshold during training. This is achieved by tracking a moving average of the global classification threshold. [1]

### 3.3.2 Losses

The network’s capability to predict motion segmentation allows the introduction of novel self-supervised losses to enhance the regularization of the training phase.

**Nearest Neighbor Loss.** In particular, the authors utilize a well-established self-supervised loss called the nearest neighbor loss, described in section 2.2.1. This loss is applied to both flows, the raw flow (generated by RAFT) and the static aggregated flow (from the Kabsch algorithm).

The final NN loss is as

$$L_{nn} = \frac{1}{\|\mathcal{P}_t\|} \sum_i e_i(\mathbf{f}_{raw}) + e_i(\mathbf{f}_{stataggr}) \quad (3.11)$$

where  $e_i()$  is an error function defined in equation 2.3.

**Rigid Cycle Loss.** Authors of SLIM proposed Rigid Cycle Loss. This approach was motivated by the successful use of cycle consistency losses in recent applications [23, 17]. The network is applied not only to the original pair of point clouds  $(P_t, P_{t+1})$  to predict transformation matrix, but also in the reversed order  $(P_{t+1}, P_t)$  to predict the inverse rigid motion  $T_{t+1 \rightarrow t}$ . The expected outcome of applying these transformations to the previous frame is to ensure that the frame is in the same position as before, because these two rigid transformation should be inverses of each other. To evaluate the error, both transforms are applied to the input point cloud and the resulting position is compared to original one. [1]

The rigid cycle loss is defined as

$$L_{rcc} = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} |(T_{t+1 \rightarrow t} T_{t \rightarrow t+1} - I_4) \mathbf{p}_i| \quad (3.12)$$

where  $\mathcal{P}_t$  is the original pointcloud and  $T_{t \rightarrow t+1}$  and  $T_{t+1 \rightarrow t}$  are transformation matrices produced by Kabsch algorithm.

The rigid cycle loss exhibits numerous local minima, such as zero matrices, thus necessitating the regularization of networks through the incorporation of additional losses.

**Artificial Label Loss.** Lastly, they proposed the Artificial Label Loss, which is used to train motion segmentation for static and dynamic classes. Its purpose is to guide the network in predicting the correct class for each point and also to decide whether to use static flow or dynamic flow. The loss is closely related to the nearest neighbor loss, as the decision for pseudo-labels is based on errors from static and dynamic flow. If the error from static flow is smaller, it is assumed that the semantic label of this point is static, and vice versa. The standard binary cross-entropy loss is used, which relies on artificially assigned labels depending on the relative magnitudes of the NN errors. The loss is defined as

$$L_{al} = - \sum_{\mathbf{p}_i \in \mathcal{P}_t} [e_{dyn_i} < e_{stat_i}] \log \sigma(l_{cls,i}) + [e_i \geq e_{r,i}] \log (1 - \sigma(l_{cls,i})) \quad (3.13)$$

where  $e_{stat_i}$  is an error from nearest neighbor loss for static flow and  $l_{cls,i}$  is semantic class produced by the model. [1]

### 3.3.3 Baseline

We aimed to integrate the model into a larger self-supervised pipeline that is currently being developed. The authors of the original work provided TensorFlow code, but our requirement necessitated the model to be implemented using the PyTorch framework. Consequently, we decided to implement the

type	$AEE_{Dyn} \downarrow$	$AEE_{Stat} \downarrow$	$AEE_{50-50} \downarrow$
official	<b>0.1050</b>	0.0925	<b>0.0988</b>
ours	0.2207	<b>0.0799</b>	0.1503

**Table 3.1:** Comparing our implementation with the official on Nusncenes dataset.

model from the scratch based on the details provided in the paper<sup>1</sup>. As a result, there may exist minor variations in the implementation from the official version.

We used the subsampled pointclouds, for faster training and inference of the model. We reduced each input pointcloud to 8192 points. The authors themselves experimented with this subsampling approach and observed that the model was capable of learning relevant features even when applied to these subsampled point clouds. However, it was noted that there was a decrease in performance, but employing the complete point cloud results into significantly longer training time.

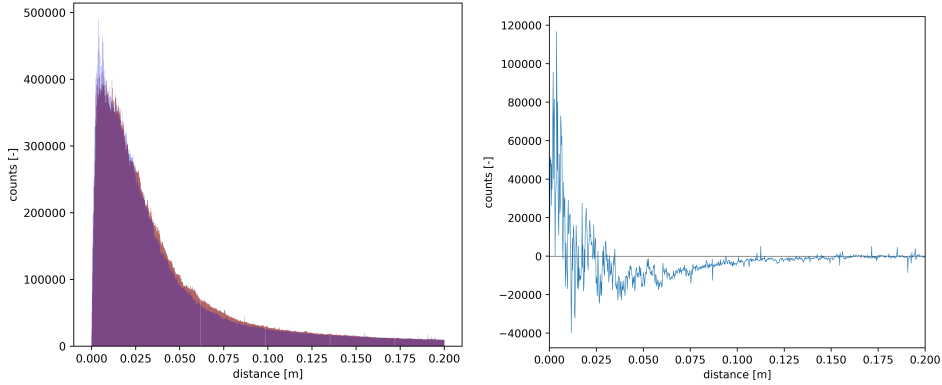
We also followed the same optimization process, which is consisted of two steps: firstly, the prediction of flow from time  $t$  to  $t + 1$ , followed by the reverse mode prediction. This two-step approach facilitated the acquisition of forward-backward odometry, which was subsequently employed for the rigid cycle loss computation.

For the purpose of comparing the different implementations, the authors conducted their study also using a Nusncenes dataset, but excluding the utilization of the Waymo dataset. They achieved an average endpoint error of 0.0925 on static points, whereas our approach yielded an average endpoint error of 0.0799. While these results appear promising, the author’s implementation resulted in an AEE of 0.1050 on dynamic points, compared to our implementation which get an average endpoint error of 0.2207. The results are depicted in table 3.1.

Our training process involved utilizing subsampled point clouds consisting of 8192 points. Unfortunately, the authors did not provide information regarding the number of points used in their training process. However, we think that they employed the full point cloud, as a bigger number of points has been shown to enhance scene understanding and consequently get improved results. Despite knowing the fact, the computational requirements dramatically increase with increased number of points in point cloud, thus we did not used bigger amount number in the pointclouds.

We have also observed that our model encounters problems in correct segmentation, resulting in incorrect flow assignments. Consequently, this leads to wrong outcomes for dynamic objects. In an attempt to enhance the segmentation process, we implemented several upgrades, which will be discussed in subsequent sections.

<sup>1</sup>Codes are available on <https://github.com/simonpokorny/MotionFeatureLearning>



(a) : Histogram of discretized endpoint error for the our Baseline (blue) and Official setup (red).

(b) : Differences between the histograms by subtracting the histogram of the official setup from the our baseline.

**Figure 3.4:** Analysis of endpoint error: Official setup vs. our baseline on Waymo Open Dataset

type	AEE ↓	AccR ↑	AccS ↑	Outl ↓	OutlR ↓
official	0.0701	<b>0.8673</b>	<b>0.7513</b>	0.3012	0.0354
baseline	<b>0.0644</b>	0.8629	0.6918	<b>0.3008</b>	<b>0.0267</b>

**Table 3.2:** Comparing the models with default and added losses, where we add smoothnes and static point loss; both are trained and tested on the waymo dataset

The original architecture only incorporates three mentioned losses: the Artificial Label Loss, Rigid Cycle Loss, and Nearest Neighbor Loss. These losses are assigned weights of 2, 1, and 0.1, respectively. This choice of weights is motivated by the observation that the most reliable pseudo labels are obtained through the nearest neighbor approach, which effectively work alongside with the rigid cycle loss. In cases where one loss reaches a degenerative minimum and is zero, the other loss is very high. The pseudo labels generated by the artificial label loss is improving during the training process; however, it remains challenging to determine with high probability the correct labels based on the conditions utilized within this particular loss function.

However, in order to enhance the original model, we incorporate two additional losses: the Smoothness Loss (see 2.2.2) and the Static Point Loss, which will be described in the next section. We posited that the integration of these losses would improve the performance. Table 3.2 demonstrates that these losses yielded to improved results in AEE metrics. Thus, we established the configuration as the baseline for subsequent experiments, resulting in total loss as

$$L_{total} = L_{nn}(\mathcal{F}_{raw}) + L_{nn}(\mathcal{F}_{rigid}) + L_{rcc} + 0.1 \cdot L_{al} + L_{spl} + L_{smooth} \quad (3.14)$$

where first two losses are Nearest Neighbor losses on raw flow from RAFT

and on rigid flow from Kabsch algorithm, then Rigid Cycle loss, Artificial label loss, Static point loss and Smoothness loss.

As depicted in fig 3.4, we have observed an increase in the count of data points exhibiting endpoint error ranging from 0 to 0.01 m. However, we have also noted a decrease in the number of points with endpoint error in range from 0.03 to 0.1 m. Moreover, a significant portion of these points falls within the previously mentioned interval of 0 to 0.01 m.

We also conducted a study how the adding additional channels as intensity on the input influence the performance. We used the Waymo official pointclouds, where are 5 features  $[x, y, z, elongation, intensity]$ .

inputs	AEE ↓	AccR ↑	AccS ↑	Outl ↓	OutIR ↓
3 features	<b>0.0644</b>	<b>0.8629</b>	<b>0.6918</b>	<b>0.3008</b>	<b>0.0267</b>
5 features	0.0939	0.7937	0.6218	0.3462	0.0494

**Table 3.3:** Comparing models trained and tested on Waymo: xyz-only versus xyz with elongation and intensity.

The results depicted in table 3.3 demonstrate that only utilizing the input feature 'xyz' yields better performance. However, the underlying reasons for this outcome remain uncertain, as the necessitating further investigation and analysis. We suspect that intensity values change based on frame-to-frame view point and observation angles and therefore unexpectedly change the estimates for flow.

Lastly, we evaluate the analysis of the domain transfer, wherein the model was trained on the Raw Kitti dataset [6] and then evaluated on the Nuscenes [3] and Waymo [9] dataset.

tested on	AEE ↓	AccR ↑	AccS ↑	Outl ↓	OutIR ↓
Waymo	0.0676	0.8784	0.7734	0.2563	0.0332
Nuscenes	0.0903	0.8105	0.6651	0.3235	0.0669

**Table 3.4:** Results of the models trained on the raw kitti train dataset and tested on Waymo

The numbers from table 3.4, demonstrate that the model is able generalize across the multiple datasets, despite the fact, they were captured with different LiDAR sensors or even with different frequency. Furthermore, it is apparent that the model trained on the Kitti dataset exhibits comparable performance to the model trained on the Waymo dataset, as their evaluation was done on the Waymo dataset.

### ■ 3.3.4 Static Point Loss

This loss is used in our baseline implementation for the SLIM architecture. The model utilize RAFT for obtaining the raw static motion flow and a Kabsch algorithm for computing the static aggregation flow based on semantic segmentation and the static flow. The goal of this loss is to improve the

accuracy of the static flow by encouraging the RAFT to produce a more homogeneous and smooth static flow. The loss is computed as a weighted mean square error, where the weights  $w_i$  correspond to the semantic class of each point indicating how much static is it. The static aggregation flow is compared to the raw static flow as follows

$$L_{spl} = \frac{1}{|\mathcal{P}_t|} \sum_{\mathbf{p}_i \in \mathcal{P}_t} w_i \|\mathbf{f}_{statagg_i} - \mathbf{f}_{stat_i}\|^2 \quad (3.15)$$

where  $\mathbf{f}_{statagg_i}$  is the static aggregation flow for point  $\mathbf{p}_i$  and  $\mathbf{f}_{stat_i}$  is the raw static flow for the same point. The optimization of this loss results in improved static flow estimation and consequently better odometry. [1]

## 3.4 Leveraging the Temporal Structure of the Data

To enhance the performance of the model, novel loss functions for self-supervised learning on point cloud data are introduced. These losses are specifically designed to leverage the sequential nature of the data, thereby providing additional signals for improved self-supervision.

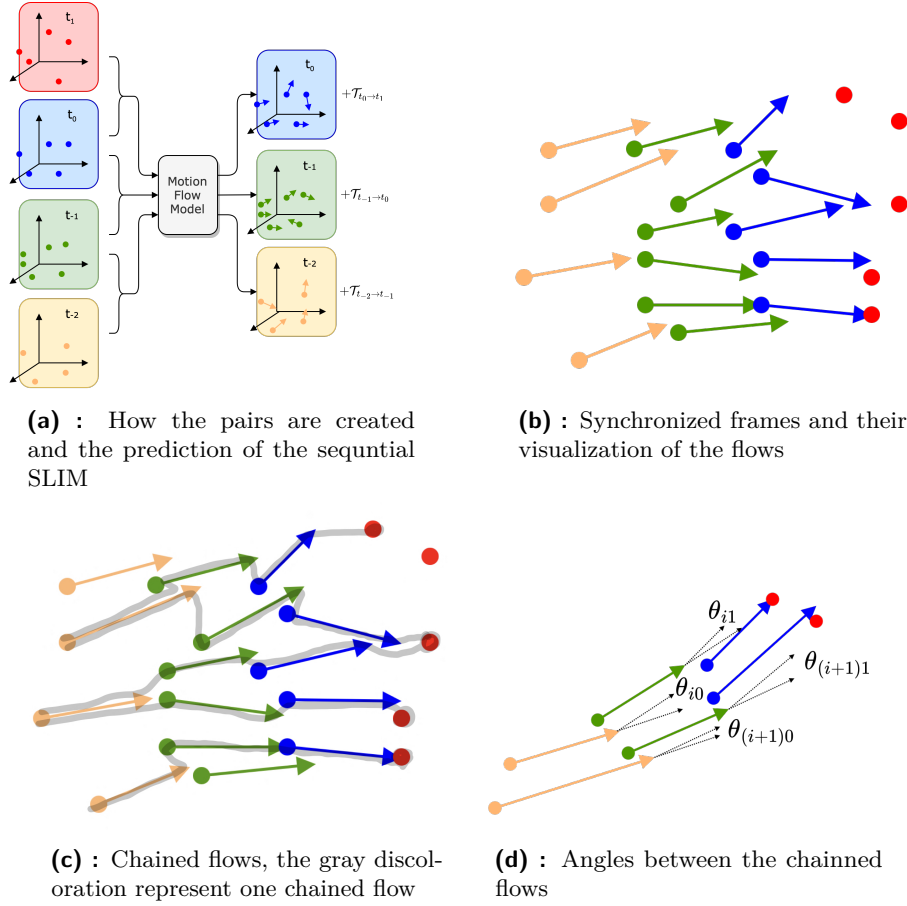
### 3.4.1 Velocity Loss (VC)

To the best of our knowledge, there is no approach specifically addressing the regularization of flow within sequential data, taking into consideration the sequential nature of automotive datasets. To accomplish this, we propose a strategy in which we grouped four consecutive images at times  $t_{-2}, t_{-1}, t_0, t_1$ , resulting in three pairs:  $(t_{-2}, t_{-1}), (t_{-1}, t_0), (t_0, t_1)$ . This approach eliminates the need to modify the model’s architecture. Instead, we feed each pair into the model individually, optimizing the model’s parameters after all three pairs have gone through the forward-pass as it is pictured in fig 3.5a.

Through the forward pass of these paired sequences, we obtain three distinct flows and corresponding predictions of odometry. During our experimental analysis, we observed that the rigid transformation achieves high accuracy within a few iterations in the original SLIM. Consequently, we can rely on it as a solid prediction for true odometry after this initial convergence. To leverage this observation, we synchronized all four point clouds using these predictions, resulting in a sequence of four consecutive frames. Fig 3.5b visually illustrates the projected flow alongside the corresponding points within this synchronized point cloud. In our notation, the pointcloud used to predict the flow for  $t_0$  is represented in blue, while the subsequent point cloud is shown in red. The point cloud and flow for  $t_{-1}$  are depicted in green, and for  $t_{-2}$  is in yellow.

The number of scanned points in each frame varies, leading to a non-bijective mapping of flow between frames. However, we assume that the scanning density within the automotive datasets is sufficiently high. Therefore, even if a point does not have a direct correspondence in another frame, it is likely to have a similar point within its close neighborhood. To address this problem, we employ point matching across the sequence, leveraging the flow information.





**Figure 3.5:** Overall visualization for Time Consistency Loss

We consider the point cloud at time  $t_0$  as our anchor. By using a nearest neighbor function, we identify the corresponding matched points in frame  $t_1$  by finding the closest neighbors to the points in  $t_0$  based on their associated flow vectors. Similarly, we perform this matching process in reverse, seeking the closest neighbors for points from  $t_0$  in the point cloud at  $t_{-1}$  using their corresponding flow vectors. Additionally, we apply a similar approach to find the closest neighbors for points in  $t_{-1}$  within the point cloud at  $t_{-2}$  with their flow vectors. This ensures that we establish correspondences across consecutive frames, enabling a comprehensive mapping of points throughout the sequence. The resulting mappings, referred to as chained flow, is visualized in fig 3.5c. As depicted in the figure, it is evident that certain chained flow mappings have identical correspondences at certain points in time. This issue is inevitable and cause that the some points will have not any loss a thus no gradients will not be propagete from these points.

We compute the angles between correspondings flows in each chained flow by utilizing the atan2 function on their respective x and y coordinates. This calculation yields two angles for each chained flow,see fig 3.5d. Given that the datasets are captured at a frequency of 10Hz or 20Hz, the four consecutive

frames represent a duration of 0.4 seconds or 0.2 seconds, respectively. Within this small time frame, the directional changes are expected to be smooth. We then put these angles to mean square error as

$$L_{angle} = MSE(\theta_{i0}, \theta_{i1}) \quad (3.16)$$

for each chained flow  $i$ .

Furthermore, we extend the assumption of smooth flow to the magnitude of the flow vectors within this small time frame. The change between the pairs  $(t_{-2}, t_{-1})$  and  $(t_{-1}, t_0)$  should also exhibit a smooth behavior. We make the assumption that we are capturing a dynamic rigid object, where the velocity derivative within this time frame remains constant. Resulting in the velocity loss as

$$L_{velocity} = MSE(\|\mathbf{f}_{it_{-2}}\|_2 - \|\mathbf{f}_{it_{-1}}\|_2, \|\mathbf{f}_{it_{-1}}\|_2 - \|\mathbf{f}_{it_0}\|_2) \quad (3.17)$$

where index  $i$  correspond to chained flow and  $t$  to capturing time of flow.

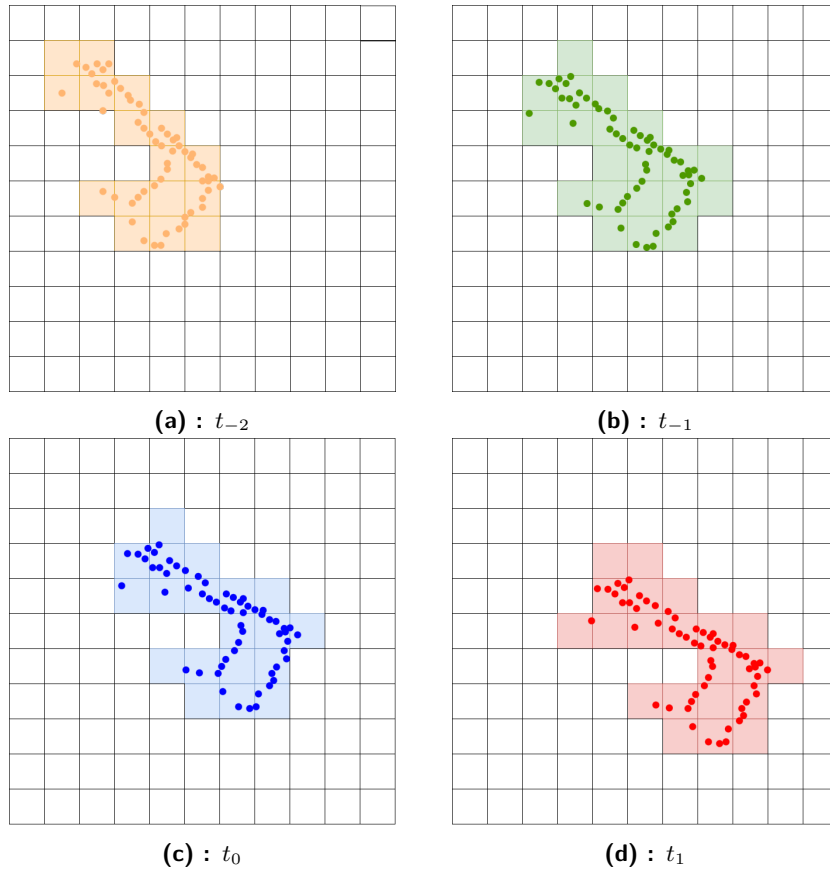
### 3.4.2 Dynamic Consistency Loss (DC)

We also want to utilize the sequential data to create pseudo labels to further improve the motion segmentation. After the scene synchronization of four consecutive frames by predicted odometry, it can be shown that static object stays in the same pillars, however we want to make extra pseudo labels for dynamic objects. As it is visualized in fig 3.6, the moving car on the pillars change its position in time. We are able to make relatively strong assumption about points on the front and the back of the car. We will also employ the chained flow. If every flow in one chained flow is heading to an unoccupied pillar or heading from occupied and the pillar will be in the next frame unoccupied, we can assume that were the edges of the moving vehicle.

In order to leverage these identified points, we attempted to update only these specific points, resulting in a small subset of points assigned with this pseudo labels. We do not extend this information into some clusters, because this approach makes weak pseudo labels, as it heavily depends on the correct predicted odometry. Additionally, this method generates labels not only for dynamic objects, but also for the ground, as the curvature of the Waymo dataset surface needs a more sophisticated approach to remove all ground points, while the naive cropping of the pointcloud in z coordinates does not remove all ground points.

### 3.4.3 Experiments

For the setup involving the use of consistency losses, we begin training with the baseline model, which was trained using only 25,000 training update iterations. This checkpoint provides accurate predictions of the odometry. The effectiveness of consistency losses relies heavily on accurate odometry predictions during their pseudo creation or training regularization. However, these methods cannot be trained directly from the beginning in the final



**Figure 3.6:** Behavior of the dynamic object in synchronized pointclouds on the pillars.

setup. Once the consistency losses are added, they do not harm the training process and are not prone to encountering degenerative minima. However, the chained flow has not been correct, and it can create incorrect regularizations. We trained the model on additional 50,000 train iterations, incorporating losses from the baseline along with an additional consistency loss.

type	AEE ↓	AccR ↑	AccS ↑	Outl ↓	OutlR ↓
Baseline	0.0566	0.8904	0.7579	0.2839	<b>0.0245</b>
VC	<b>0.0529</b>	<b>0.9122</b>	<b>0.8108</b>	<b>0.2837</b>	0.0248
DC	0.0876	0.8100	0.5252	0.3361	0.0324

**Table 3.5:** Standard metrics for consistency methods; baseline is trained with 75,000 iterations, while the models with consistency losses are initialized with baseline trained for 25,000 iterations and then trained for 50,000 iterations.

As observed in tab 3.5, the model trained using Velocity Loss (VC) outperformed the baseline model in various metrics, particularly in AccR and AccS. This indicates that the model is capable of refining the flow to ensure temporal consistency, resulting in slightly improved results with a higher proportion of smaller endpoint errors. However, it fails in predicting the

segmentation, as demonstrated in tab 3.6. The performance of this setup was notably poor in terms of average endpoint error for dynamic objects, while the segmentation predicted static flow.

type	AEE ↓	AEE <sub>Dyn</sub> ↓	AEE <sub>Stat</sub> ↓	AEE <sub>50-50</sub> ↓
Baseline	0.0566	0.1751	0.0516	0.1134
VC	<b>0.0529</b>	0.3509	<b>0.0402</b>	0.1955
DC	0.0876	<b>0.1173</b>	0.0864	<b>0.1018</b>

**Table 3.6:** Separate metrics for dynamic and static parts for consistency methods.

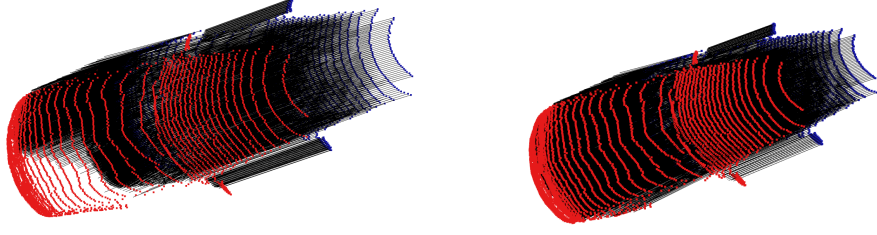
The Dynamic Consistency loss (DC) primarily aims to regularize the segmentation in order to enhance the assignments of the flow. This improvement led to a decrease in average endpoint error on dynamic objects by approximately 5 cm. This improvement results from the better segmentation. However the static flow performs worse, because the model’s predictions for dynamic classes were more accurate, but it occasionally mislabeled static areas as dynamic resulting in worse static flow.

### 3.5 Object Classification Consistency

It has been observed that motion segmentation predictions often fail to consider the continuity of objects. This issue is particularly noticeable in the case of cars, where the majority of points belonging to a car may be predicted as dynamic, while certain parts are classified as static. An example of such an wrong prediction can be seen in fig 3.7, where the front part of the car is inaccurately classified as static. This leads to significant errors due to the substantial disparity between the static aggregated flow obtained through the Kabsch algorithm and the dynamic flow estimated using the RAFT method. Conversely, when the network incorrectly predicts dynamic motion as static, the resulting error is comparatively smaller, as the dynamic flow can still provide a rough estimation of the flow for static objects.

In order to tackle the aforementioned issue, we have introduced a refinement module that serves as a non-learnable refinement layer. This module is only activated during the evaluation mode. Moreover, we aimed to address this inconsistency of classification during the training phase as well. To achieve this, we tried to enhance the artificial label loss that was originally proposed in SLIM [1].

Both approaches need to utilize instance segmentation to address the problem. Our objective does not involve training an additional model component for precise instance prediction, because the methods have not required accurate instance classification. They can effectively handle oversegmented pointclouds, thus we integrate the DBSCAN algorithm [4]. The results from the algorithm is pictured in fig 3.8.



(a) : Prediction without refinement module      (b) : Prediction with refinement module

**Figure 3.7:** Motion segmentation, where the front portion of the car is inaccurately classified as static (static aggregated flow is used for these points) and the segmentation with Refinement module

### 3.5.1 Self-supervised Refinement Module

We attempted to incorporate the refinement module, although the absence of labels do not allowed us follow the approaches employed in previous works such as Non-Rigid Residual Flow and Ego-Motion, FlowNet3D or PV-RAFT [23, 15, 24]. In those studies, the authors utilized refinement training to fine-tune the last few convolution layers in order to enhance and refine the flow. To circumvent this limitation, we proposed a refinement module that does not require any training.

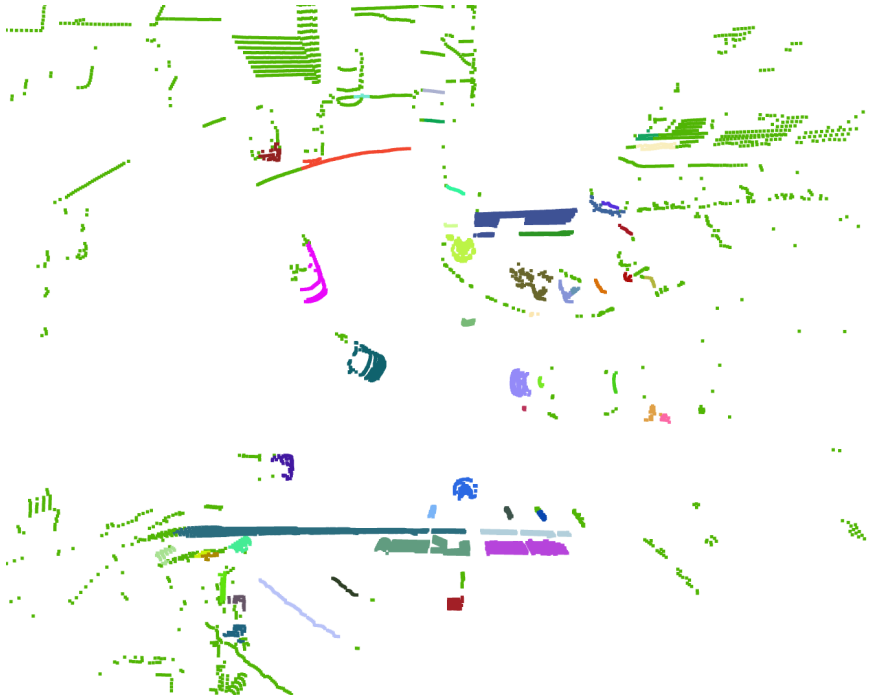
The self-refinement module can operate on the top of the motion flow network. Our model predicts both the static aggregated flow and the raw dynamic flow for each point. Subsequently, we apply a self-supervised DBSCAN algorithm to cluster the point cloud. The clustering results are illustrated in fig 3.8. For each cluster, we make a decision regarding which flow (static or dynamic) to utilize as

$$\mathbf{f}_{cluster} = \arg \min_{\{\mathbf{f}_{stataggr}, \mathbf{f}_{dyn}\}} \frac{1}{|\mathcal{C}_j|} \sum_i (NN_{\mathcal{P}_{t+1}}(\mathbf{p}_i + \mathbf{f}_i) - (\mathbf{p}_i + \mathbf{f}_i)) \quad (3.18)$$

where  $|\mathcal{C}_j|$  is number of points in the cluster and the sum is over the all point in cluster. Then, we select the flow that has a smaller nearest neighbor distance within each cluster. For points that are not clustered by the algorithm (i.e., noise points), a conventional classification approach is employed. Through the utilization of this refinement module, the flow and motion segmentation should to exhibit increased rigidity and homogeneity within the clustered objects, leading to improved predictions.

### 3.5.2 Object Aware Artificial Label Loss

The Artificial Label Loss proposed in the SLIM framework, developed by the authors, does not consider the object’s homogeneity and rigidity. The creation



**Figure 3.8:** Cluster pointcloud with DBSCAN [4]; the noise is light green and othres colors represent the clusters

of pseudo labels for motion segmentation is based on the smaller nearest neighbor distance between the static aggregated loss and the raw dynamic flow, performed on a point-wise basis. In an effort to enhance the scene description, we have extended this loss. Firstly, we cluster the pointcloud using self-supervised DBSCAN algorithm, as depicted in figure 3.8. For noise points, the loss behaves as a typical artificial label loss. For clustered points the decision differs in that a single pseudo label is created for each cluster, rather than on a point-wise basis. The selection of the pseudo label for cluster follows a similar approach, where the flow with a smaller nearest neighbor distance over all points in cluster is chosen as the representative class for the cluster.

We also considered the object’s rigidity. The flow should have the same characteristics within the cluster, and therefore, we also attempted to utilize L2 regularization for all flows within the cluster. As a pseudo label, we employed the mean value from all flows within the cluster. This minor enhancement is referred to as the Object Aware Artificial Loss with Smoothness.

### ■ 3.5.3 Experiments

We replicated the training configuration employed in training the baseline model. The refinement module utilized the baseline model as the underlying model for conducting refinement. In the Object-Aware Artificially Label Loss, the classical Artificial Label loss was replaced with an improved loss function.

type	AEE ↓	AccR ↑	AccS ↑	Outl ↓	OutlR ↓
Baseline	0.0644	0.8629	0.6918	0.3008	0.0267
Refine Module	<b>0.0540</b>	<b>0.8909</b>	<b>0.7678</b>	<b>0.2756</b>	<b>0.0224</b>
Object Aware AL	0.0645	0.8585	0.6988	0.2999	0.0279
OAAL with Smoothness	0.0647	0.8407	0.6786	0.3155	0.0304

**Table 3.7:** Classical metrics for object consistency methods on Waymo dataset

Based on the results presented in table 3.7, it is evident that the Object Aware Loss yielded comparable results to the baseline. Nevertheless, conducting a separate evaluation of the dynamic and static components in table 3.8, it becomes apparent that the loss function led to a improvement of 0.018 m in dynamic error, indicating a promising outcome. The model demonstrated its ability to predict classes in a more concise manner. However, the performance of this setup was slightly inferior for static points, which comprise the majority of the dataset. Furthermore, the 50-50 evaluation was enhanced.

type	AEE ↓	AEE <sub>Dyn</sub> ↓	AEE <sub>Stat</sub> ↓	AEE <sub>50-50</sub> ↓
Baseline	0.0644	0.1560	0.0605	0.1082
Refine Module	<b>0.0540</b>	<b>0.1314</b>	<b>0.0507</b>	<b>0.0911</b>
Object Aware AL	0.0645	0.1380	0.0614	0.0997
OAAL with Smoothness	0.0654	0.1452	0.0633	0.1042

**Table 3.8:** Separate evaluation for dynamic and static points for object consistency methods on Waymo dataset

The self-supervised non-learnable refinement module exhibited enhancements in comparison to the baseline, yielding notable improvements. This straightforward naive method effectively addresses the challenge of flow selection in the model. Specifically, the average endpoint error across all points was reduced by 0.0104 m, while the dynamic error showed an even more significant reduction of 0.0246 m. Moreover, the metrics pertaining to outliers indicate a better performance in terms of reasoning capability.

Moreover, we tried to regularize the Object Aware Artificial Label loss with Smoothing loss, resulting in "OAAL with Smoothness". This improvement behaves as Object Aware AL, but it also trying to smooth the flow within the clusters to behave same. For the noise from DBSCAN any smooth loss is not employed. However the results from table 3.7 and 3.8 shows that this improvements do not influenced the results very much.

Furthermore, we attempted to enhance the regularization of the Object Aware Artificial Label loss by incorporating a Smoothing loss, which we refer to as "OAAL with Smoothness." This modification not only preserves the Object Aware AL behavior but also aims to smooth all flows within the cluster in similar manner as in the original Smoothness loss. Despite these improvements, the results from tables 3.7 and 3.8 indicate that the impact on the results is minimal.

### 3.6 Combination

We also attempted to merge both types of regularization in order to benefit from both approaches. Regarding the temporal structured based losses, we decided to incorporate both approaches, as they have shown to enhance the model’s performance. For an object awareness losses, we selected the self-supervised refinement module (RM).

type	AEE ↓	AccR ↑	AccS ↑	Outl ↓	OutlR ↓
Baseline	0.0566	0.8904	0.7579	0.2839	0.0245
VC	0.0529	<b>0.9122</b>	<b>0.8108</b>	0.2837	0.0248
VC + RM	<b>0.0500</b>	0.9031	0.7886	0.2788	<b>0.0211</b>
DC	0.0876	0.8100	0.5252	0.3361	0.0324
DC + RM	0.0566	0.8847	0.7631	<b>0.2780</b>	0.0258

**Table 3.9:** Classical metrics on Waymo dataset for combining loss functions

Based on the observations from table 3.9, it seems that integrating the refinement module with losses that depend on the sequential structure of the data does not improve the performance of the model. However, improving the performance of the model using the same architecture is a difficult task.

The Waymo dataset consists mostly from static points, meaning that significant improvements in dynamic points have little impact, while even minor disturbances in static points can greatly change the results.


Therefore, a deeper understanding of how the losses impacted the flow prediction can be obtained from the table 3.10. Similar as before, the table divides the main metric (AEE) into static and dynamic components, and  $AEE_{50-50}$  where each part have equal weight of 50%.

type	AEE ↓	$AEE_{Dyn}$ ↓	$AEE_{Stat}$ ↓	$AEE_{50-50}$ ↓
Baseline	0.0566	0.1751	0.0516	0.1134
VC	0.0529	0.3509	<b>0.0402</b>	0.1955
VC + RM	<b>0.0500</b>	0.2133	0.0426	0.1282
DC	0.0876	<b>0.1173</b>	0.0864	0.1018
DC + RM	0.0566	0.1176	0.0541	<b>0.0858</b>

**Table 3.10:** Distinct metrics on Waymo dataset for dynamic and static parts for models integrating both approaches.

Based on the  $AEE_{50-50}$  metric, the model that performs the best includes all baseline losses, along with an additional Dynamic Consistency loss and Self-supervised Refinement Module. On average, this configuration was able to enhance performance by 2.76 cm in comparison with baseline. Although the performance on the static part of the data was not improved, there was a notable reduction of 5.75 cm in the average endpoint error for the dynamic points in the dataset. This improvement is significant, considering that the baseline model struggled with dynamic data.





## Chapter 4

### Conclusion

In order to achieve precise training within the self-supervised paradigm, it is important to incorporate appropriate self-supervised losses that effectively leverage the temporal and spatial characteristics of the data. We explored an approach inspired by the sequential characteristics of the automotive dataset. As a result, we introduced a Velocity loss aimed at achieving smooth flow over time by considering the derivative of individual point velocities during small time interval. Additionally, we proposed the Dynamic loss, which generates motion segmentation labels by comparing the motion patterns with the ego vehicle. This observation led to creating a pseudo label for segmentation. Both of these losses heavily depend on accurate odometry predictions, making the creation of pseudo labels susceptible to incorrect predictions of the odometry.

We have also observed that our baseline model often encounters difficulties in segmentation, leading to incorrect flow assignment. Based on these observations, we propose an extension to the previously suggested loss and also self-supervised refinement module, which operates on top of the model. This extension loss, called the Object Aware Artificial label loss, aimed to uniformly assign the correct pseudo segmentation label to each object, avoiding different pseudo labels for a few points of the object. For clustering the objects from the scene, we utilize the widely known self-supervised DBSCAN clustering algorithm, which provides us with clusters from the scene. Within these clusters, a segmentation label is assigned based on the smaller average nearest neighbor distance to the next point clouds from corresponding endpoints. This decision-making process is employed for distinguishing between static and dynamic flow. The self-supervised refinement module works in a similar manner but eliminates the segmentation prediction based on logits from the network. The decision of the class is derived from the smaller average nearest neighbor distance within the clusters. It is important to note that this module is only added to the evaluation process.

Based on an experiment evaluated on the Waymo Open Dataset, we observed that the most effective configuration involves incorporating all losses utilized in the baseline model along with our proposed Dynamic Consistency loss and Self-supervised Refinement Module. This particular arrangement resulted in an enhancement of 2.76 cm in the mean of a static average endpoint

#### 4. Conclusion

error and dynamic endpoint error. Notably, the improvement was more significant for dynamic points, with an error reduction of 5.75 cm achieved. The results of our work suggest that self-supervised learning can effectively leverage also the dynamics in 3D point cloud data during larger time interval.

## Appendix A

### Bibliography

- [1] Stefan Baur, David Emmerichs, Frank Moosmann, Peter Pinggera, Bjorn Ommer, and Andreas Geiger. Slim: Self-supervised lidar scene flow and motion segmentation. In *International Conference on Computer Vision (ICCV)*, 2021.
- [2] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 1992.
- [3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [4] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, 1996.
- [5] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical flow modeling and computation: A survey. *Computer Vision and Image Understanding*, 2015.
- [6] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [7] Zan Gojcic, Or Litany, Andreas Wieser, Leonidas J. Guibas, and Tolga Birdal. Weakly supervised learning of rigid 3d scene flow. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [8] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. Hplflownet: Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [9] Philipp Jund, Chris Sweeney, Nichola Abdo, Zhifeng Chen, and Jonathon Shlens. Scalable scene flow from point clouds in the real world. *International Conference on Robotics and Automation (ICRA)*, 2022.

- [10] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 1976.
- [11] Yair Kittenplon, Yonina C. Eldar, and Daniel Raviv. Flowstep3d: Model unrolling for self-supervised scene flow estimation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [12] Harold W. Kuhn. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly*, 2, March 1955.
- [13] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [14] Ruibo Li, Chi Zhang, Guosheng Lin, Zhe Wang, and Chunhua Shen. Rigidflow: Self-supervised scene flow learning on point clouds by local rigidity prior. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [15] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [16] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [17] Himangi Mittal, Brian Okorn, and David Held. Just go with the flow: Self-supervised scene flow estimation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [18] Gilles Puy, Alexandre Boulch, and Renaud Marlet. FLOT: Scene Flow on Point Clouds Guided by Optimal Transport. In *European Conference on Computer Vision (ECCV)*, 2020.
- [19] C. Qi, L. Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.
- [20] Ziyang Song and Bo Yang. OGC: Unsupervised 3D Object Segmentation from Rigid Dynamics of Point Clouds. In *Advances in Neural Information Processing Systems (NIPS)*, 2022.
- [21] Olga Sorkine-Hornung. Laplacian mesh processing. In *Eurographics*, 2005.
- [22] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European Conference on Computer Vision (ECCV)*, 2020.

