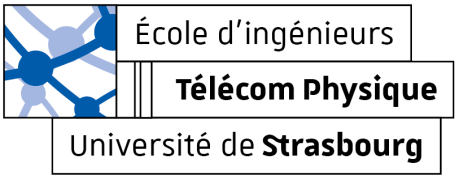


Real-Time LiDAR SLAM with Neuro-Symbolic and Ontological Mapping

M. Mohammed ELAMINE

Data Science and Artificial Intelligence 2020-2023
Academic Year: 2022-2023

Host Organization: SnT, University of Luxembourg
Internship Dates: 13/03/2023 - 31/08/2023

Supervisors:
Prof. Dr. Thomas LAMPERT (TPS)
Dr. Sk Aziz ALI (SnT, University of Luxembourg)

Acknowledgments

Above all, I would like to express my deepest gratitude to my supervisor, **Dr. Sk Aziz Ali**, for his expert guidance, endless patience, and constant motivation. His invaluable insights have played a pivotal role in shaping the trajectory of this project.

I am truly thankful to **Prof. Djamila AOUADA**, the Head of the CVI^2 research group, for opening the door to join her team and for the unwavering support she has provided throughout.

I extend my heartfelt appreciation to **Prof. Abd El Rahman SHABAYEK**, **Dr. Wassim Rharbaoui Dr. Nilotpal Sinha**, **Dr. Carl Schneider**, **Mr. Romain Hermary**, **Dr. Vincent Gaudilliere**, , and every member of the CVI^2 research group. Their guidance, encouragement, and camaraderie have transformed my internship at CVI^2 into an immensely rewarding journey.

I also extend my gratitude to the dedicated educators and administrators at Telecom Physique Strasbourg. Their efforts have contributed significantly to the high-quality education we've received.

Lastly, I want to express my thanks to all those who have contributed directly or indirectly to the successful completion of this endeavor. Your support and influence have made a meaningful impact.

Contents

List of Figures

Introduction

Autonomous driving represents a fascinating realm of research, where continuous advancements are being made in technologies aimed at assisting and monitoring driving. The goal is to automate numerous tasks, thereby creating a more enjoyable and comfortable driving experience for users.

To achieve successful autonomous driving, a vehicle must emulate human capabilities: perceiving and comprehending its surroundings ("Sense"), processing information, making strategy for driving ("Think"), and safely executing those plans ("Act") [1]. Consequently, when in an unfamiliar setting, the vehicle should be able to accurately localize itself and construct a detailed map of its surroundings. This map proves invaluable for various functions, including re-localization, path planning, risk analysis, collision avoidance, and decision-making.

In the realm of perception, various sensors can be utilized to capture environmental data, such as cameras and LiDAR (Light Detection and Ranging). On one hand, cameras excel in tasks like object identification and interpretation of road signs, yet they are hindered by adverse weather and limited visibility. On the other hand, LiDAR sensors emit laser pulses and gauge the time taken for light to rebound off objects, creating precise 3D maps, known as "point clouds." These "point clouds" $\mathbf{PC} = \{x_i | x_i \in \mathbb{R}^3\}_{i=1}^{N}$ consist of discrete data points in space, each with its own Cartesian coordinates (x, y, z). They offer accurate distance measurements and are unaffected by adverse weather, making LiDAR reliable for safe autonomous driving, even in challenging conditions.

Furthermore, LiDAR-SLAM (Simultaneous Localization And Mapping) addresses the need to explore unknown environments by enabling the vehicle to determine its position relative to its starting point and construct a comprehensive map of its surroundings. This map is formed by combining perceived point clouds, transformed into world coordinates using the output of the localization component. However, while accumulating point clouds offers abundant information for map construction, it presents challenges in terms of data storage and descriptive capabilities. Despite its apparent richness and vehicle-specific nature, this generated map is inefficient in terms of data storage and cannot be directly used by decision-making components. Additional post-processing is required for planning and decision-making purposes.

Thus, for an autonomous vehicle to promptly and precisely react to real-world events, it necessitates a map that is both dense and lightweight. To optimize performance, the vehicle must generate real-time, accurate, and succinct local maps that seamlessly integrate with decision-making, minimizing any delays between perception and action. This entails creating informative yet lightweight maps, constituting a comprehensive system that adeptly perceives, processes, and translates environmental data into a format that decision-makers can efficiently use.

In this pursuit, the role of a visual language becomes essential. Analogous to how humans interpret instructions and align them with reality through visual faculties for decision-making, autonomous vehicles must acquire a similar capability. They need to understand the ontological and conceptual meaning of textual instructions, enabling them

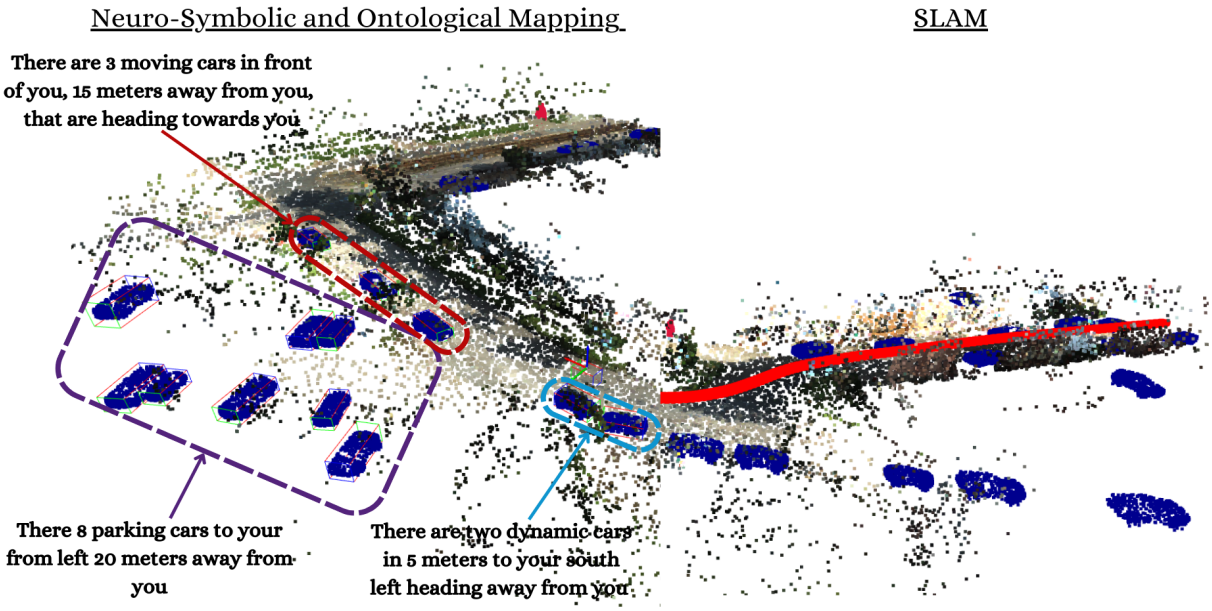


Figure 1: NSOM versus SLAM

to correlate these with visual input from sensors. Fundamentally, the vehicle must acquire a visual language that bridges the gap between textual tokens and visual signals.

By developing this visual language, the vehicle gains the ability to generate a lightweight map derived from textual descriptions. Instead of relying solely on extensive point cloud data, the vehicle can utilize textual descriptions to create a more concise and manageable representation of the environment. Moreover, these textual descriptions serve as a powerful tool for the vehicle to locate objects in the real world, analyze potential risks, and effectively plan future actions.

Thesis outline Addressing this well-defined challenge, the SNT (Interdisciplinary Centre for Security, Reliability and Trust) research laboratory has established an internship position to explore the potential of this innovative approach and delineate its boundaries. This report encompasses a comprehensive and thorough analysis of the issue, beginning with an introduction to the hosting organization (SNT). Subsequently, it delves into the core problem of constructing a lightweight environment map through the fusion of visual and textual advancements, divided into two sections. The first section introduces a deep learning method for localization in unfamiliar settings using LiDAR point clouds, known as PWCLO-Net (Pyramid, Warping, and Cost volume LOcalization Network). The second section presents another deep learning framework designed to create concise textual maps using visual input. This model is named Neuro-Symbolic and Ontological Mapping (NSOM) and draws its architectural inspiration from the 3DJCG (3D Joint Captioning and Grounding) approach. Numerous experiments are conducted within each section, and the results are thoroughly examined to evaluate the efficacy of the two models. The report concludes by suggesting avenues for advanced research that can build upon this work.

Chapter I

Host organization

I.1 Presentation

SnT (Interdisciplinary Centre for Security, Reliability and Trust) is the largest hub of information technology research within Luxembourg that belongs to the *university of Luxembourg* among two other interdisciplinary centers for instance *Luxembourg Centre for Systems Biomedicine (LCSB)* and *Luxembourg Centre for Contemporary and Digital History (C^2DH)*. SnT excels in multi-disciplinary research with a focus on cyber-security, Financial Technology, IoT, autonomous vehicles, and space systems. Their work ranges from securing critical infrastructures and developing financial technologies to optimizing IoT solutions, enhancing autonomous mobility, and supporting Luxembourg's space exploration goals.

Since its foundation in 2009, SnT has grown as a team and became one of the popular research centres that attract a large number of qualified scientists and doctoral candidates from all over the globe. The current numbers state that SnT has succeeded in building a big asset that contains more than 127 *research scientists* and 169 *doctoral candidates* from 68 *nationalities*. This diversified human resources constitute the key to the continuous and exponential evolution of the centre towards achieving their vision . In order to offer the best research experience for its scientists, SnT provides 2 *campuses* with 10 *highly equipped labs* where one is situated in *Belval* and the other one is placed in *Kirchberg*.

(a) Belval SnT campus

(b) Kirchberg SnT campus

Figure I.1: SnT campuses in Luxembourg

I completed my master's thesis internship at Kirchberg's campus within the *Computer Vision, Imaging and Machine Intelligence Research Group (CVI^2)*.

sector of activity CVI^2 group led by Dr. Djamila Aouada, focuses on computer vision, image processing, image analysis, visual data comprehension, and machine learning.

Projects, including *Space Situational Awareness Instrumentation*, *MEET-A*, *DIOSSA*, *AUREA*, *SPRING*, *DETECT*, *ENERGETIC*, *Proving Digital Asset Integrity Using Deepfake Detection*, *FakeDeTer*, *UNFAKE*, *Automated CAD Modelling*, *FREE-3D*, *CASCADES*, *ELITE*, and *RoboComp*, encompass a diverse range of initiatives aimed at advancing various domains through the integration of deep learning and advanced technologies.

Size At the present capacity of the University of Luxembourg, there are more than 7,000 students and approximately 2,400 employees from around the world. There are more than 100 courses offered over two semesters in a year and distributed across 18 Bachelor degree and 46 Master degree programs mostly in French/English or French/German. Is is also mentioned by the university that it is among the top 25 Young Universities in the world as per Times Higher Education ranks.

I.2 Organization

I have completed my internship in the CVI^2 group, led by Dr. Djamila Aouada, a seasoned expert in the field. The group boasts a team of three dedicated research scientists who oversee and guide the various projects within the group. Among this vibrant environment, a cohort of research associates plays a pivotal role as the driving force behind our endeavors. These individuals are a diverse assembly of competent researchers, carefully selected from different nationalities, lending a rich tapestry of perspectives. One noteworthy figure among them is my supervisor, Dr. Sk Aziz Ali. He earned his Ph.D and M.Sc in Computer Science from the RPTU Kl and Technical University of Kaiserslautern (TU Kl), Germany, in 2022 and 2017 respectively, specializing in computer vision, graphics, and scientific visualization. Dr. Ali joined CVI^2 in 2021, and his contributions have been remarkable, boasting an impressive tally of 10 recognized publications.

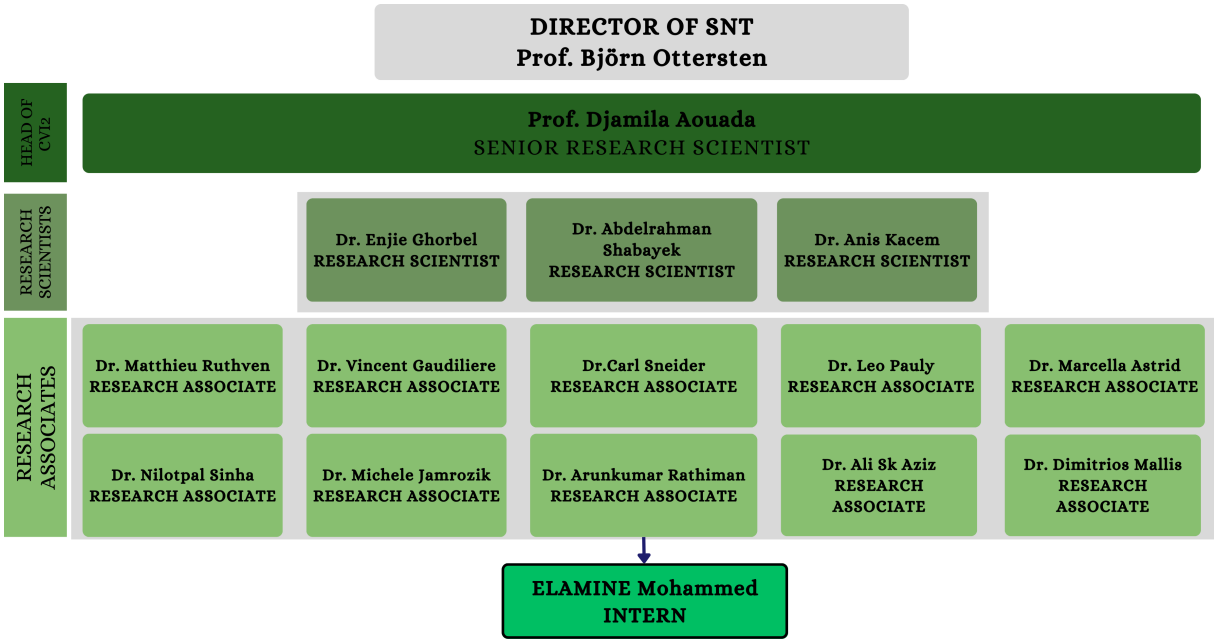


Figure I.2: Organizational chart of CVI^2 group

I.3 Financial details

CVI^2's projects require a substantial financial investment, totaling over €4.5 million. These projects are funded by multiple entities including national research organizations, international partnerships, and defense funds. Over a span of several years, these projects aim to address critical challenges in areas such as space situational awareness, digital asset integrity, reverse engineering, and more. Through these investments, these projects seek to bridge the gap between simulated and real-world data, facilitate autonomous operations, enhance anomaly detection capabilities, and advance the broader fields in which they operate.

Chapter II

LiDAR Odometry

To gain a comprehensive understanding of the surrounding environment, a vehicle must possess the ability to generate highly accurate maps, enabling it to analyze potential risks and plan for future decisions effectively. However, creating maps from 3D point clouds is a complex undertaking, as it necessitates precise matching between point clouds from consecutive frames through point cloud registration methods. In this context, LiDAR-SLAM (Simultaneous Localization and Mapping) emerges as a vital and intriguing solution that facilitates the seamless matching of point clouds between two frames, subsequently calculating the pose transformation required to convert the first point cloud to the second. This process allows for the accumulation of point clouds, resulting in the creation of dense and detailed maps representing the surrounding environment.

LiDAR odometry is a crucial component of LiDAR-based Simultaneous Localization and Mapping (SLAM) systems. Its primary objective is to estimate the vehicle's motion or pose using LiDAR point cloud data captured from consecutive frames [2]. Accurate odometry is essential for understanding the vehicle's movement through the environment, which in turn enables precise mapping and navigation.

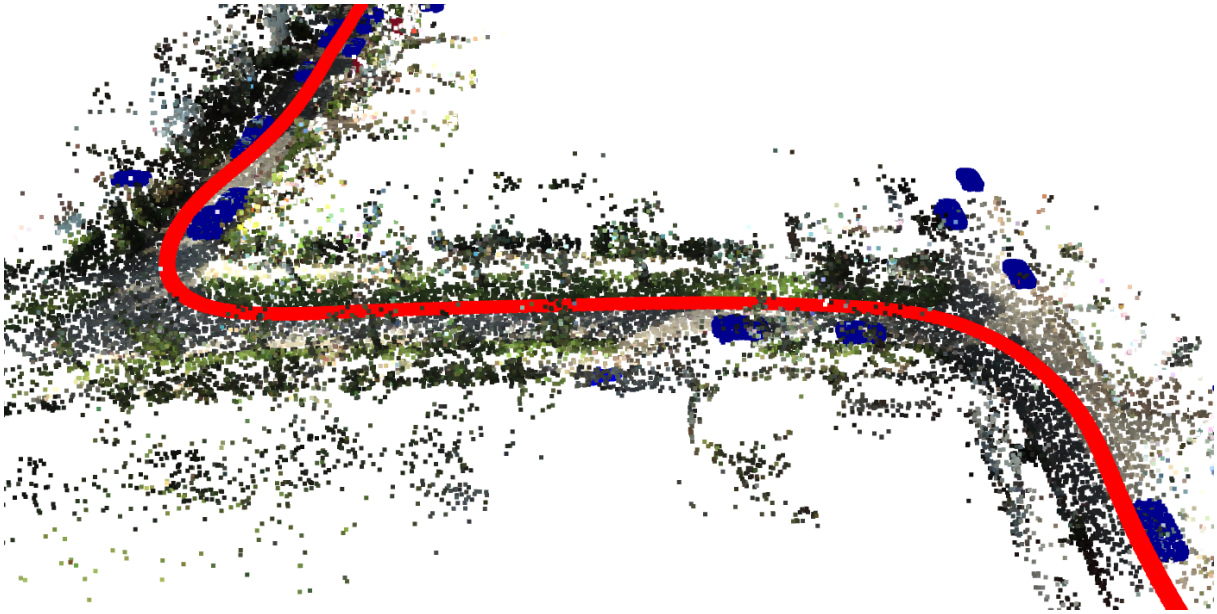


Figure II.1: Trajectory predicted using LiDAR odometry

II.1 Background

Frame-to-Frame and Frame-to-Model Methods LiDAR odometry approaches can be categorized into frame-to-frame and frame-to-model methods. Frame-to-frame methods use point clouds from two consecutive frames to estimate the pose change between them through point cloud registration. On the other hand, frame-to-model methods leverage a previously constructed map (aggregated point clouds) and the current frame's point cloud to find matching points and regress the pose from world coordinates to the current frame.

Iterative Closest Point (ICP) and its Variants One of the core methods in LiDAR odometry is the Iterative Closest Point (ICP) algorithm. It is widely used for aligning two point clouds by iteratively minimizing the distance between corresponding points in order to estimate the optimal rigid transformation between the two point clouds as illustrated in Figure II.2. However, while ICP is straightforward and efficient, it may suffer from convergence to local minima and sensitivity to initial guesses.

Several variants of ICP have been developed to address its limitations. Some methods incorporate robust cost functions to reduce the impact of outliers in the point cloud data. Others use feature-based matching techniques to improve convergence and handle partial overlap between point clouds.

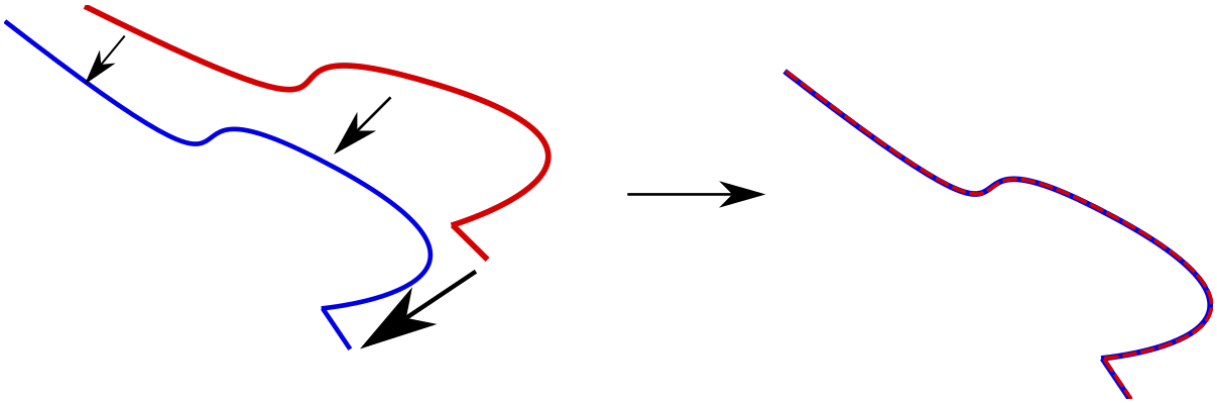


Figure II.2: Iterative Closest Point

Geometric Methods Various geometric methods leverage ICP with different point cloud representations. For instance, LOAM (Lidar Odometry and Mapping) [38] is a well-known LiDAR-SLAM system that seamlessly combines LiDAR odometry and mapping. It employs feature-based scan matching to estimate the vehicle's motion between frames using 2D range images [36] of the 3D point clouds. The mapping module generates a consistent and dense map by leveraging the estimated odometry and incorporating loop closure techniques to optimize the map further.

An extension of LOAM, LeGO-LOAM (Lightweight and Ground-Optimized LOAM) [28], introduces an additional feature association strategy to reduce computational complexity and improve real-time performance. By incorporating a lightweight loop closure approach, LeGO-LOAM achieves faster processing speeds while maintaining comparable accuracy.

Deep Learning LiDAR Odometry While traditional geometric techniques have been widely used in LiDAR-SLAM, recent progress in deep learning has sparked interest in

exploring neural network-based solutions for odometry and mapping tasks. However, the challenge lies in effectively working with raw and unordered 3D point cloud data, given their inherent nature as a collection of unstructured points. This poses an obstacle to the direct application of conventional convolutional layers. To surmount this hurdle, some approaches endeavor to transform point clouds into structured formats, thus enabling the utilization of standard convolutional layers.

Certain deep learning strategies aim to leverage the potency of 2D convolutional layers by translating 3D point clouds into 2D representations. Among the popular options are range images and voxel grids. Range images involve the projection of 3D points onto a 2D grid, creating an organized representation conducive to 2D convolutional operations. Conversely, voxel grids partition the 3D space into smaller units, or voxels, and encode pertinent point cloud data accordingly.

In the pursuit of advancing the realm of LiDAR-SLAM, researchers continue to explore novel and innovative deep learning techniques tailored to address the unique complexities posed by 3D point clouds. A particularly intriguing avenue of inquiry involves the development of deep learning models capable of directly processing the raw, untransformed 3D point clouds, thereby capitalizing on the entirety of the intricate information they encapsulate. This approach is oriented towards retaining the full richness of 3D data, thus sidestepping potential information loss during the conversion process.

Remarkably, PointNet++ [26] has risen to prominence as a foundational architecture for such deep learning models. This framework adeptly handles unordered point clouds by leveraging set abstraction and feature propagation layers, ensuring the preservation of the vital permutation invariance property intrinsic to point clouds. This property is pivotal for accurate pose regression and precise mapping. Concurrently, a notable trend gaining momentum involves the integration of attention mechanisms within deep learning models tailored for LiDAR-SLAM. These mechanisms endow the model with the capability to emphasize salient regions of the point cloud while effectively dampening the influence of irrelevant or noisy data. The adaptability of these attention mechanisms contributes to improved map quality and more reliable localization outcomes in real-world settings.

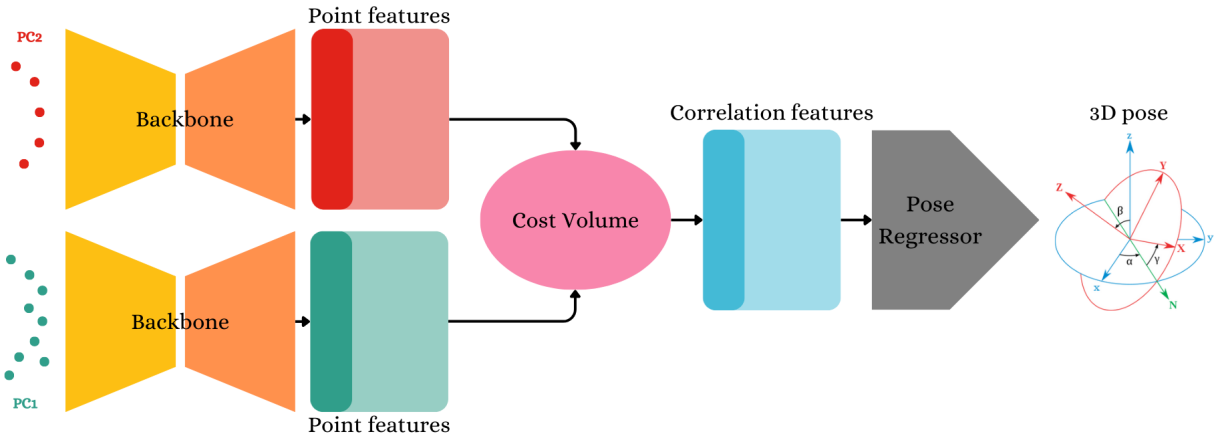


Figure II.3: Common pose regression models architecture

The majority of deep learning methodologies in this domain adhere to a standardized architecture, founded on the concept of using two point clouds from consecutive frames as input, subsequently generating the relative pose connecting them. Central to this architecture (as illustrated in Fig. II.3) is a backbone, responsible for extracting spatial and geometric features from the two point clouds. These derived feature maps are then directed into a cost volume [37], a crucial component for learning the correlation details

between the point clouds. Ultimately, the refined correlation features find their way to a pose regressor, which plays a pivotal role in accurately predicting the pose.

Notably standing out in this context is the PWCLO-Net [32] methodology, which distinguishes itself as one of the select few deep learning approaches that directly process raw 3D point clouds for LiDAR-SLAM. PWCLO-Net leverages the potency of the PointNet++ backbone, a robust deep learning architecture specifically designed to grapple with the intricacies of point clouds. By effectively retaining the unordered nature of 3D points through PointNet++'s incorporation of set abstraction and feature propagation, PWCLO-Net efficiently achieves accurate pose regression directly from the unprocessed raw point cloud data.

II.2 PyLiDAR-SLAM: A Complete Toolbox

Aligned with the ambitious goals of advancing LiDAR-SLAM research, the *pyLiDAR-SLAM* [11] toolbox emerges as a comprehensive tool addressing the needs of LiDAR odometry and point cloud registration. This versatile toolkit provides modular and user-friendly Python and PyTorch implementations of various LiDAR odometry methods, positioning itself as a valuable resource for evaluation and comparison across diverse public datasets.

One of the standout features of pyLiDAR-SLAM is its commitment to generality and flexibility. By providing multiple ready-to-use geometric methods, including ICP, point cloud matching, point-to-line, and point-to-plane distance calculations, as well as Kd-Trees and range image generation from point clouds, it offers a rich array of functionalities. These implemented geometric methods lay the foundation for precise pose calculation and manipulation, optimization functions, loop closure detection, and pose graph [13] optimization. Moreover, the seamless integration of PyTorch enables efficient computation and allows users to leverage the power of deep learning methods, further extending the toolbox's capabilities.

Beyond its algorithmic implementations, the toolbox also facilitates data preparation through utilities tailored for various datasets like KITTI [15], KITTI-360 [21], Ford-Campus [3], NCLT (North Campus Long-Term) [7], and UrbanLoco [34]. This incorporation streamlines the data preparation process, making diverse datasets readily accessible for experimentation and testing.

By offering an all-in-one solution, pyLiDAR-SLAM becomes an invaluable asset for the LiDAR-SLAM community. It empowers researchers to better understand the inner workings of the LiDAR-SLAM pipeline, encourages the exploration of novel methods, and enables a more seamless comparison of results across different datasets and algorithms.

LiDAR-SLAM pipeline

The pyLiDAR-SLAM toolbox offers a comprehensive and user-friendly LiDAR-SLAM pipeline (fig. [2]), encompassing six main components that facilitate accurate and efficient point cloud registration and mapping.

1. ***Motion Initialization***
   Motion initialization is a critical step in LiDAR Odometries, especially for ICP-based methods with small convergence regions. pyLiDAR-SLAM provides three strategies for motion initialization:

   – *NI (No Initialization)*: No specific initialization is performed, relying solely on the algorithm's default settings.

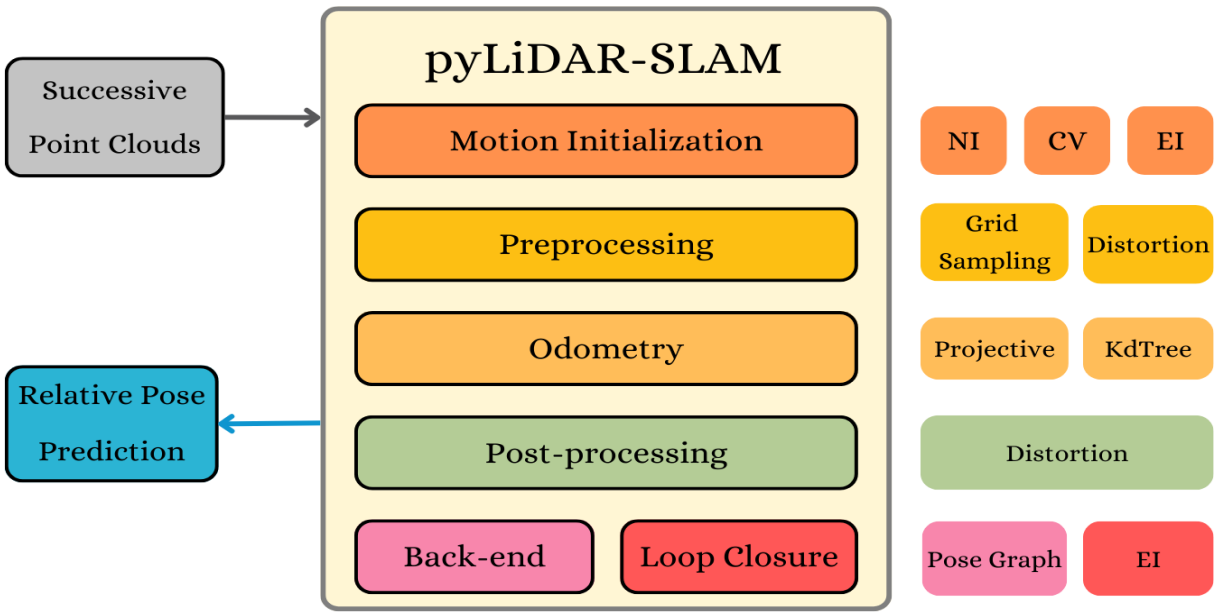


Figure II.4: pyLiDAR-SLAM pipeline

- *CV (Constant Velocity)*: The motion between two frames is assumed to be constant, and the estimated motion is the same as the previous one.
- *EI (Elevation Images)*: This approach employs 2D image feature-based registration to forecast vehicle motion within the 2D plane of movement. This technique requires only two translations and one rotation to estimate motion due to its focus on the 2D plane of vehicle motion. The process involves projecting point clouds into elevation images, converting each point's projection onto the chosen 2D plane into a pixel value representing its height. Despite its benefits, this approach has drawbacks, including possible data loss and susceptibility to alterations in sensor orientation.

2. ***Preprocessing***
   Preprocessing involves addressing distortions caused by the LiDAR sensor's motion during a sweep (complete rotation). Additionally, to manage the large amount of point cloud data efficiently, context-aware point sampling is employed to select relevant points while preserving the overall data quality.

3. ***Odometry***
   pyLiDAR-SLAM implements frame-to-model odometry, which estimates the relative pose of a new frame in relation to the previous registered poses. The odometry process includes the following steps:

   (i) Estimating the motion of the new frame

   (ii) Preprocessing the new frame for alignment

   (iii) Registering the new frame against a Local Map (built from previous registered frames)

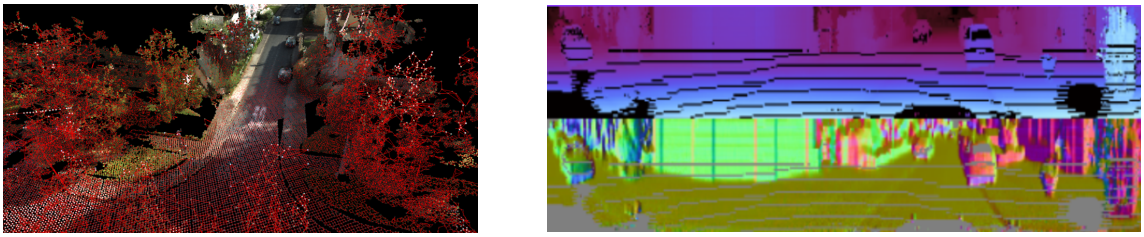
   (iv) Searching for neighbor points and their normals in the map

   (v) Minimizing a robust energy (typically point-to-plane residuals) using Gauss-Newton optimization

   (vi) Updating the map with the new frame's information

Besides, two implementations of a Local Map are provided:

- *KdTree*: A k-d tree [1], short for k-dimensional tree, is a data structure used to efficiently arrange points in a multi-dimensional space. This tree is structured as a binary tree, with each node representing a point in the multi-dimensional space. Non-leaf nodes create dividing hyperplanes that partition the space, enabling points to be organized into subtrees based on their position relative to these hyperplanes. In LiDAR-SLAM, k-d trees are constructed from transformed point clouds to enable fast querying. However, this construction process can be computationally intensive. To tackle this challenge, point sampling techniques are employed to balance the trade-off between computational speed and the retention of critical information needed for LiDAR-SLAM tasks.
- *Projective (Range Image)*: A range image serves as a two-dimensional depiction of three-dimensional point cloud data, mapped onto a grid where each grid cell corresponds to a specific part of the three-dimensional space. The range image captures depth information, representing the distances from the LiDAR sensor to various objects or surfaces. In the context of LiDAR-SLAM, a sequence of previous point clouds is adjusted to align with the most recent pose coordinates and is then projected into a two-dimensional space using spherical projection. This approach enables projective data association, although it may not be optimal for sparsely populated LiDARs due to the definition of neighborhoods based on shared pixel values. Despite this limitation, this technique facilitates real-time SLAM implementation, even if there is a compromise in precision.



(a) 3d-tree (b) 2D range image

Figure II.5: 2D representations of 3D point cloud

4. ***Post-processing***
   After estimating the relative pose between consecutive frames, the input point cloud is transformed using this estimation to align it with the local map. This alignment enriches the map and adds more information.

5. ***Loop Closure***
   Traditionally, odometry relies on integrating velocity measurements over time to estimate the robot's position, making it sensitive to errors. These errors can accumulate and lead to inaccuracies in position estimates over prolonged periods, especially in dynamic and unknown environments. To counter this issue, robots may need to continuously detect loop closures to correct the pose graph constructed from odometry estimates. Loop closure detection involves identifying previously visited locations or landmarks to refine the map and alleviate the impact of accumulated errors.

6. ***Back-end (Pose Graph Optimization)***
   Upon detecting loop closures, the pose graph optimization process corrects accu-

mulated errors, enabling continuous adjustments and improvements in pose estimations.

The pyLiDAR-SLAM toolbox offers a flexible and efficient solution for LiDAR-SLAM, empowering researchers and engineers to understand, experiment, and test LiDAR-SLAM on various datasets. Its modular design and integration of geometric methods and deep learning techniques provide a powerful platform for customizing and extending the pipeline. By uniting the strengths of computer vision and LiDAR-based perception, pyLiDAR-SLAM paves the way for advancing autonomous navigation, mapping, and perception systems.

Although pyLiDAR-SLAM offers various features, it lacks implementations of well-known deep learning methods such as PWCLO-Net or SUMA++. In this study, a user-friendly PWCLO-Net adaptation that adheres to the pyLiDAR-SLAM fraework will be presented.

II.3 PWCLO-Net: A Novel Approach for Addressing Out-of-Order Distribution in 3D Point Clouds

II.3.1 Deep Learning Challenges in Processing 3D Point Clouds

The input is a subset of Euclidean space points with three distinct properties [32]:

(i) *Unordered Nature*: Unlike images or volumetric grids, point clouds lack a predetermined order, making them sets of points. When dealing with N 3D point sets, a network must be immune to the N! permutations stemming from varying input orders.

(ii) *Point Interaction*: Points emerge from a space with a distance metric, indicating that they aren't isolated entities; rather, neighboring points compose a coherent subset. Hence, a model should adeptly grasp local structures from nearby points and the complex interactions between these structures.

(iii) *Transformation Invariance*: As a geometric entity, the learned representation of a point set must remain unaltered under specific transformations. For instance, collectively rotating and translating points should neither alter the global point cloud's category nor the segmentation of points.

II.3.2 PointNet++: Feature learning from raw 3D point clouds

In order to build a model that is invariant to permutation inside the same input point cloud, PointNet++ propose a novel layer that can learn and extract features directly from raw 3d point clouds. Their main idea is to use a symmetric function to aggregate the features from the neighbors of previously selected points. Hence this symmetric function will remove the affect of order in point feature learning.

Their novel layer is called *Set Abstraction (SA)* layer or *Set Conv* layer (like *Convolutional* layers) and its purpose is to learn important and distinguishable geometric and spatial features from the point cloud. Given N number of points in a 3D point cloud $\mathbf{X} = \{x_i\}_{i=1}^{N}$, The SA layer first pre-process $\mathbf{X}$ by sub-sampling it to $\widehat{\mathbf{X}} = \{\hat{x}^{(j)}\}_{j=1}^{N'}$ with N' number of points using the iterative *Farthest Point Sampling* algorithm [26] such that $\hat{x}^{(j)}$ is the most distant point (in metric distance) from the set $\{\hat{x}^{(1)}, \hat{x}^{(2)}, ..., \hat{x}^{(j-1)}\}$ with

regard to the rest points. This will allow to select a subset of points while preserving an equal density distribution over all the regions of the point cloud. Then, it selects K nearest neighbors $\mathbf{X^{(i)}} = \{x_j^{(i)}\}_{j=1}^{K}$ from the surrounding of each selected centroid $\hat{x}^{(i)}$ by FPS. These neighbors can be chosen either by using the *knn* algorithm or by querying a ball and selecting the points that belongs to that ball. Once the neighbors of the selected centroids are computed, it aggregates their features into the centroids using a symmetric function such as *Max Pooling* or *Average Pooling* layers. Hence, the output of the SA layer can be expressed by the following equation:

$$\mathbf{SA}\left(\mathbf{X}\right) \approx \left\{\mathbf{MaxPool}\left(\mathbf{MLP}\left(\tilde{x}_1^{(i)} \oplus f_1^{(i)} \oplus \hat{f}^{(i)}\right), ..., \mathbf{MLP}(\tilde{x}_K^{(i)} \oplus f_K^{(i)} \oplus \hat{f}^{(i)})\right)\right\}_{i=1}^{N'} \tag{II.1}$$

where $\forall i \in [1, N']$, $\exists \mathbf{X^{(i)}}$ such as $\tilde{x}_j^{(i)} = x_j^{(i)} - \hat{x}^{(i)}$. $\oplus$ represents the concatenation operator and **MLP** function refers to a *Multi-Layer Perceptron* network that can be used for the spacial encoding of a point and $f_j^{(i)}$ and $\hat{f}^{(i)}$ are respectively the local features of the neighbor $x_j^{(i)}$ and the centroid $\hat{x}^{(i)}$. It's worth mentioning that by using relative coordinates together with point features, the SA layer can capture point-to-point relations in the local region.

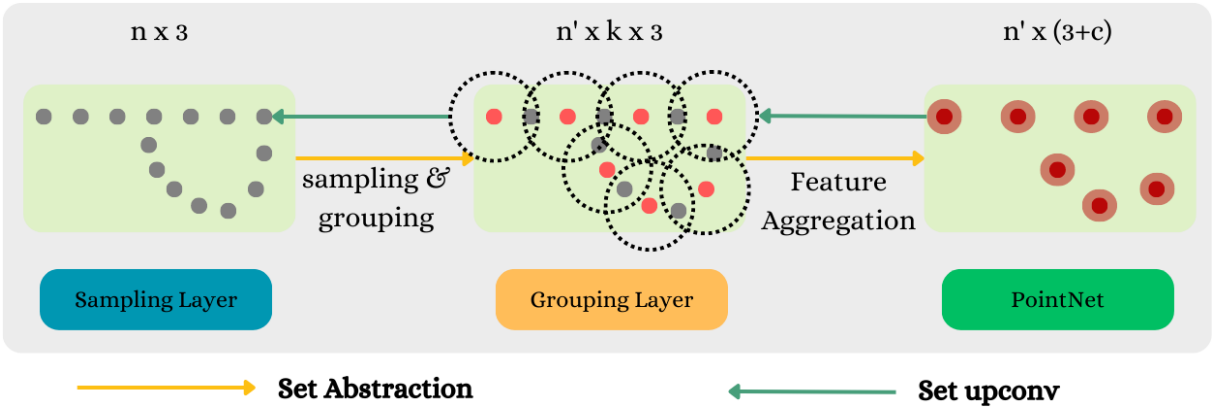


Figure II.6: Set Abstraction and Set Up-Conv layers

II.3.3 Improving LiDAR odometry using 3D scene flow analysis

Scene flow is the 3D motion field of points in the scene [30]. It is a low-level understanding of a dynamic environment, without any prior knowledge of structure or motion of the scene. One of the well-known deep learning models that is able to predict the scene flow of a 3D scene is *FlowNet3D* [22]. In their paper, the suggested two novel layers called *Flow Embedding* layer and *Set Up-Conv* layer. While the first one learns how to encode the flow embeddings between two point clouds, the second one tries how to propagate the learned features. In fact, the Set Up-Conv layer does the reverse function of the Set Abstraction layer that was introduced in Sec. II.3.2.

Given two 3D point clouds with two number of points N_1 and N_2 respectively from two successive frames (N_1 and N_2 can be different), FlowNet3D outputs the 3D scene flow $\{fl_i\}_{i=1}^{N_1}$ such as the euclidean distance between the warped point cloud $\{\bar{x}_i\}_{i=1}^{N_1}$ and the target point cloud $\{y_j\}_{j=1}^{N_2}$ tends to zero.

$$\forall i \in [1, N_1] \quad \bar{x}_i = fl_i * x_i \tag{II.2}$$

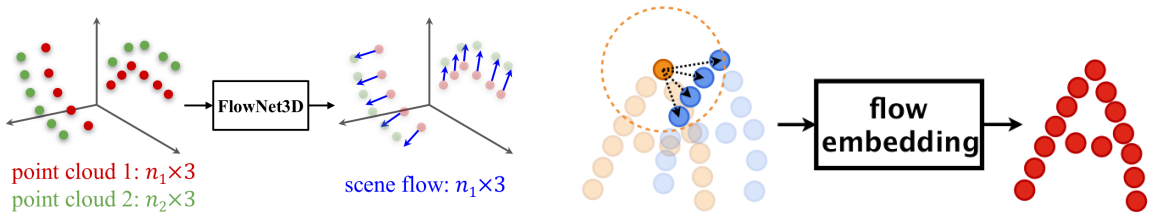
Flow Embedding layer (FE) This layer learns to aggregate both (geometric) feature similarities and spatial relationships of points to produce embeddings that encode point motions [22]. In contrast to the set convolutional layer, which operates on an individual point cloud, the flow embedding layer processes a pair of point clouds $\mathbf{PC_1} = \{p_i = (x_i, f_i)\}_{i=1}^{N_1}$ and $\mathbf{PC_2} = \{q_j = (y_j, g_j)\}_{j=1}^{N_2}$. Here, each point possesses its XYZ coordinate $x_i, y_j \in \mathbb{R}^3$, along with a feature vector $f_i, g_j \in \mathbb{R}^C$. Within this layer, a flow embedding is acquired for every point within the initial frame, denoted as $\mathbf{E} = \{e_i | e_i \in \mathbb{R}^{C'}\}_{i=1}^{N_1}$.

$$\textbf{SA}\left(\mathbf{PC_1}\right) \approx \left\{\textbf{MaxPool}\left(\textbf{MLP}\left(z_1^{(i)} \oplus g_1^{(i)} \oplus \hat{f}^{(i)}\right), ..., \textbf{MLP}(z_K^{(i)} \oplus g_K^{(i)} \oplus \hat{f}^{(i)})\right)\right\}_{i=1}^{N_1} \quad \text{(II.3)}$$

where $\forall i \in [1, N_1]$, $\exists \{y_j^{(i)}\}_{i=1}^{K}$ such as $z_j^{(i)} = y_j^{(i)} - \hat{x}^{(i)}$ and $\{y_j^{(i)}\}_{i=1}^{K}$ represents the K nearest neighbors of the centroid $\hat{x}^{(i)}$ in $\mathbf{PC_2}$. The distinction between the set abstraction layer II.1 and the flow embedding layer lies in their focus. The former considers solely the points and features within a single point cloud, while the latter incorporates the relative coordinates and features of both point clouds. This inclusion facilitates the layer's ability to effectively capture correlation details between $\mathbf{PC_1}$ and $\mathbf{PC_2}$.

Although the flow embedding layer succeeds in capturing the correlation information between the source and target point clouds, also known in the literature as the *Cost Volume* [37], its K nearest neighbors search method does not take into consideration the case where these points can belong to another object with a different rigid motion. For instance, let's suppose that we want to encode the flow embedding of a source point belonging to a car that is next to a tree. Since the *knn* algorithm uses the euclidean distance solely as a metric, it will result on selecting nearest neighbor points from both the car and the tree. However, if the car is dynamic, then the flow motion of the car is totally different that the tree's flow motion. Thus, considering the points belonging to the tree for flow encoding will deteriorate the results. To resolve this issue, PWCLO-Net proposes a novel cross attention mechanism called *Attentive Cost Volume* that will be discussed in details in Sec. II.3.4.

Set Up-Conv layer As in 2D convolutional layers, Set Up-Conv layer mitigates the role of up-convolutional layers by propagating the features back in a learnable way. This layer is not much different from the SA layer and uses the same principle used for leaning the features. The only difference is that the output points are now known in advance. Thus, given N input 3D points $\mathbf{X} = \{x_i\}_{i=1}^{N}$ with their features $\mathbf{F} = \{f_i\}_{i=1}^{N}$ and N' target 3D points $\widehat{\mathbf{X}} = \{\bar{x}_i\}_{i=1}^{N'}$, the output of the Up-Conv layer is the new set of points $(\widehat{\mathbf{X}}, \widehat{\mathbf{F}})$ where $\widehat{\mathbf{F}} = \{\hat{f}_i | \hat{f}_i \in \mathbb{R}^{C'}\}$ is the new set of aggregated features from the K nearest points of each point in $\widehat{\mathbf{X}}$ (Fig. II.6).



(a) 3D scene flow [30]

(b) Flow Embedding layer [30]

Figure II.7: 2D representations of 3D point cloud

To summarize, a 3D scene flow intends to predict the flow motion of each point in the point cloud. Compared to segmentation and classification in 2D images, 3D scene flow

problem can be considered as a low-level view of the relative pose estimation problem. In the latter, a relative pose is estimated that can bring the original and target point clouds closer. In this pursuit, PWCLO-Net builds its architecture based on extracting the flow embeddings of each point in the source point cloud using an attention mechanism in order to estimate the relative pose of the ego-vehicle. Additionally, it proposes a novel embedding mask to filter out the points that shouldn't contribute to the pose estimation such as dynamic or occluded points.

II.3.4 PWCLO-Net - architecture

As discussed in Sec. II.3.3, PWCLO-Net [32] exploits the success of the proposed layers by PointNet++ [26] (*Set Abstraction*) and FlowNet3D [22] (*Flow Embedding* and *Set Up-Conv*) both in point feature encoding and point correlation encoding (respectively). Additionally, PWCLO-Net proposes two novel layers, for instance *Attentive Cost Volume* and *Mask Embedding*, in order to accurately estimate the relative pose given the flow embeddings between the two point clouds.

The Fig. II.8 illustrates the overall structure of PWCLO-Net. The inputs to the network are two 3D point clouds $\mathbf{PC_1} = \{x_i\}_{i=1}^{N}$ and $\mathbf{PC_2} = \{y_j\}_{j=1}^{N}$ respectively sampled from two successive frames. These two point clouds are first encoded by a *Siamese Point Feature Pyramid* that consists of multiple set abstraction (SA) layers as introduced in Sec. II.3.2. Then the attentive cost volume is used to encode the flow embeddings between the two point clouds using the flow embedding (FE) layer introduced in Sec. II.3.3 and attention mechanism. To regress the pose from these attentive flow embeddings, PWCLO-Net generates an embedding mask to give more weights to the static points. Lastly, PWCLO-Net uses an iterative pose refinement architecture (Pose Warp-Refinement) that works on improving the pose estimation in a coarse-to-fine approach. This pose is represented by the *quaternion* [19] parameters $q \in \mathbb{R}^4$ for rotation and a translation vector $t \in \mathbb{R}^3$.

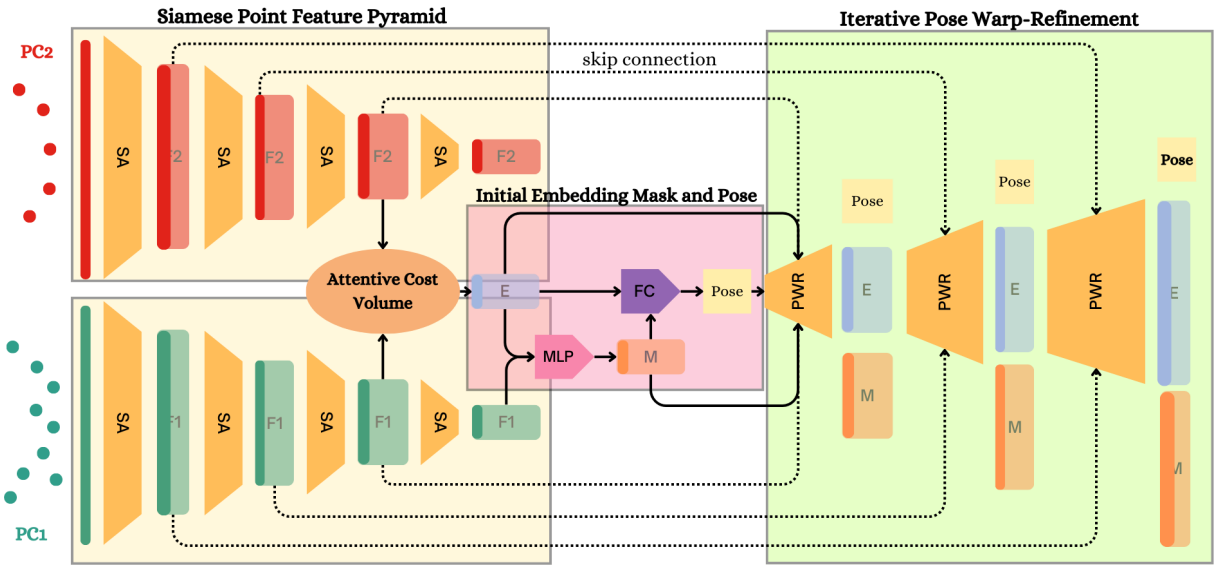


Figure II.8: PWCLO-Net architecture

(i) *Siamese Point Feature Pyramid*
Given two disorganized and sparse point clouds, PWCLO-Net works first on extracting and learning spatial and geometric hierarchical features using several set abstraction layers (see Sec. II.3.2). These layers construct a pyramid that encodes

point features while reducing the number of points and enriching the features dimension. It is called *siamese* because the learning parameters are shared between the two blocks of set abstraction layers. It is constructed this way in order to allow the hierarchical feature pyramid to learn to encode features from both input point clouds jointly. This will help the attentive cost volume to encode the correlation between the two point clouds efficiently.

The Siamese Point Feature Pyramid takes as input the source and target point clouds: $\mathbf{X} = \{x_i\}_{i=1}^{N}$ and $\mathbf{Y} = \{y_j\}_{j=1}^{N}$ respectively and returns their encoded features for each level $\mathbf{F_l} = \{f_i^l | f_i^l \in \mathbb{R}^{C_k}\}$ and $\mathbf{G_l} = \{g_j^l | g_j^l \in \mathbb{R}^{C_l}\}$ respectively, where $l \in [1, 4]$ (the presented architecture in Fig. II.8 has 4 levels) and C_l is the output number of channels of the feature map of the l^{th} level.

(ii) *Attentive Cost Volume*
Once the spatial features of the two point clouds are encoded, the attentive cost volume uses a flow embedding layer (see Sec. II.3.3) to encode the point correlation information between the two frames along with a cross attention mechanism that ensures to consider only useful nearest neighbors for the flow encoding. As mentioned in Sec. II.3.3, the flow embedding layer suffers from inaccuracies especially in dynamic and complex environment such as driving scenarios. In fact, considering the nearest neighbors based only on the euclidean distance can create some logical mistakes (see Sec. II.3.3 for more details). To resolve this issue, PWCLO-Net uses a cross attention mechanism that takes as input the points set of the source point cloud $(\mathbf{X}, \mathbf{F_3})$ and their corresponding neighbors $(\mathbf{Y^{(i)}}, \mathbf{G_3^{(i)}})$ and returns learned weights for each neighbor between 0 and 1. These weights should have higher values for the neighbor points that significant in contributing to the flow encoding of the source point. This way, the flow embedding layer will only consider neighbor points in $\mathbf{PC_2}$ that belongs to the same object as the queried points in $\mathbf{PC_1}$ (for more details, please refer to the PWCLO-Net paper [32]. Then, to achieve spatial smoothness and address uncertain scenarios necessitating extensive receptive fields for flow calculation (like points on a translating car's surface) a set abstraction is further applied to the generated embeddings. This results on an attentive flow embeddings $\mathbf{E} = \{e_i | e_i \in \mathbb{R}^{C_4}\}_{i=1}^{N_4}$ that capture the correlation information between $\mathbf{PC_1}$ and $\mathbf{PC_2}$, where N_4 is the number of points output by the 4^{th} level of the siamese point feature pyramid.

(iii) *Initial Embedding Mask and Pose*
In order to accurately estimate the relative pose, the attentive cost volume generates an embedding mask of the scene that will allow to filter out the points that shouldn't contribute to the pose estimation process. To deal with this, the flow embedding features $\mathbf{E}$ and the spatial embedding features $\mathbf{F}$ of the first point cloud PC_1 are used to construct the embedding mask as follows:

$$\mathbf{M} = \mathbf{softmax}(\mathbf{sharedMLP}(\mathbf{E} \oplus \mathbf{F})) \tag{II.4}$$

where $\mathbf{M} = \{m_i | m_i \in \mathbb{R}^{C_4}\}_{i=1}^{N_4}$ represents the trainable mask for prioritizing flow embedding features of N_4 points in $\mathbf{PC_1}$. Thus, each point has a corresponding weight from 0 to 1 where the lower the weight is the more the point won't be considered for the relative pose estimation. The Fig. II.10 shows one driving scenario where the embedding mask brings valuable information about the rigid motion of the objects. The dynamic car on the left has lower weights (blue points) than the steel bars around the highway (red points).

To regress the relative pose $\mathbf{P}_4 = (t_4, q_4)$ from the generated attentive flow embeddings, a weighted sum is performed along with *Fully Connected (FC)* layers as follows:

$$q_l = \frac{\mathbf{FC}\left(\sum_{i=1}^{N_l} e_i \cdot m_i\right)}{\left|\mathbf{FC}\left(\sum_{i=1}^{N_l} e_i \cdot m_i\right)\right|}, \tag{II.5}$$

$$t_l = \mathbf{FC}\left(\sum_{i=1}^{N_l} e_i \cdot m_i\right), \tag{II.6}$$

where $\cdot$ represents *dot product*.

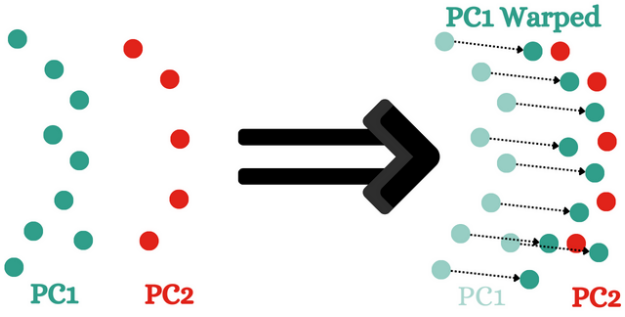


Figure II.9: Point warping

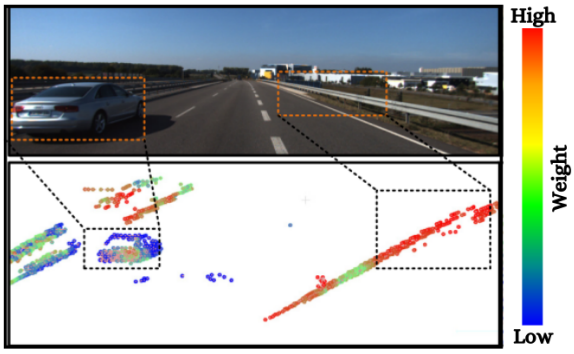


Figure II.10: Embedding mask

For more fluent explanation in the future sections, let's fuse the last two modules (attentive cost volume and initial embedding mask and pose) inside one big module that we call *Masked Attentive Cost Volume (MACV)*. This module takes as input two point clouds with their corresponding spatial features $\mathbf{PC}_1^l = (\mathbf{X}^l, \mathbf{F}^l)$ and $\mathbf{PC}_2^l = (\mathbf{Y}^l, \mathbf{G}^l)$ and returns:

- The attentive flow embeddings $\mathbf{E}^l\{e_i^l | e_i^l \in \mathbb{R}^{C_4}\}_{i=1}^{N_l}$ of the the points in $\mathbf{PC}_1^l$.
- The corresponding embedding mask $\mathbf{M}^l\{m_i^l | m_i^l \in \mathbb{R}^{C_l}\}_{i=1}^{N_4}$.
- The predicted pose $\mathbf{P}^l = (t^l, q^l)$ between the two input point clouds.

where l refers to the current level ($l \in [1, 4]$). This module can be interpreted as a function:

$$\mathbf{MACV}(\mathbf{X}^l, \mathbf{Y}^l, \mathbf{F}^l, \mathbf{G}^l) = (\mathbf{E}^l, \mathbf{M}^l, \mathbf{P}^l). \tag{II.7}$$

(iv) *Iterative Pose Warp-Refinement (PWR)*
In order to refine the predicted pose, PWCLO-Net proposes an iterative pose refinement process. Using the last predicted relative pose, the source 3D point cloud $\mathbf{X}_l$ can be warped into a new 3D point cloud $\mathbf{X}_{warped}^l = \{x_{i,warped}^l | x_{i,warped}^l \in \mathbb{R}^3\}$ as follows:

$$[0, x_{i,warped}^l] = q^{l+1}[0, x_i^l](q^{l+1})^{-1} + [0, t^{l+1}]. \tag{II.8}$$

The Fig. II.9 shows that the warped point cloud $PC_{1,warped}^l$ is now closer to the target point cloud PC_2^l. Thus estimating the residual relative pose between these two new point clouds is easier at the l^{th} level.

Pursuing the same method as described before, in order to regress the residual relative pose, we need to generate a new embedding mask and new flow embedding features. However, to take advantage of predictions of the last level $l+1$, the last

predicted attentive flow embedding features $\mathbf{E}^{l+1}$ and the last predicted embedding mask $\mathbf{M}^{l+1}$ can be refined along with the relative pose for better residual motion estimation. Hence, these three pieces of information will be transmitted to the dense layers of the point cloud and optimized in a coarse-to-fine approach during the hierarchical pose warp-refinement process. This progressive approach enhances the accuracy of the final mask estimation and pose transformation calculation.

First, in order to propagate the embedding mask $\mathbf{M}^{l+1}$ and the flow embedding features $\mathbf{E}^{l+1}$ to the l^{th} layer, the set up-conv layer, introduced in Sec. II.3.3, can be used. As already described, this layer learns to propagate the features from sparse to dense levels while preserving the information encoded in the features. As a result, a new coarse embedding mask $\mathbf{CM}^l$ and a new coarse flow embedding features $\mathbf{CE}^l$ are generated that need to be optimized in the l^{th} level.

A slightly modified version of the predefined function **MACV** in Eq. II.7 can be used to optimize $\mathbf{CM}^l$ and $\mathbf{CE}^l$ and regress the residual pose:

$$\mathbf{MACV}(\mathbf{X}^l, \mathbf{Y}^l, \mathbf{F}^l, \mathbf{G}^l | \mathbf{CE}^l, \mathbf{CM}^l) = (\mathbf{E}^l, \mathbf{M}^l, \mathbf{P}^l), \tag{II.9}$$

that takes two additional inputs ($\mathbf{CE}^l$ and $\mathbf{CM}^l$) and works on optimizing them. Hence, the Eq. II.7 represents a special case of the new version where $\mathbf{CE}^l = \{e_i^l | e_i^l = 0\}$, $\mathbf{CM}^l = \{m_i^l | m_i^l = 1\}$.

In order to regress the residual pose transformation between the warped point cloud $PC^l_{1,warped} = (\mathbf{X}^l_{warped}, \mathbf{F}^l)$ and the target point cloud PC^l_2, the function II.9 can be called with $\mathbf{X}^l_{warped}$ as the source 3D points and the features can be used from the output of the siamese point feature pyramid using *skip connections* [35]. Additionally, the optimization of the flow embedding features can simply be done by using a **MLP** layer as follows:

$$\mathbf{E}^l = \mathbf{MLP}\left(\mathbf{CE}^l \oplus \mathbf{RE}^l \oplus \mathbf{F}^l\right), \tag{II.10}$$

where $\mathbf{RE}^l$ represents the residual flow embeddings output by the attentive cost volume. The embedding masked, on the other hand, can be optimized by simply taking into consideration the coarse mask prediction in Eq. II.4 as follows:

$$\mathbf{M}^l = \mathbf{softmax}(\mathbf{sharedMLP}(\mathbf{E}^l \oplus \mathbf{CM}^l \oplus \mathbf{F}^l)). \tag{II.11}$$

This process allows estimating the residual pose transformation $\Delta P_l = (\Delta t_l, \Delta q_l)$ that can be used to refine the previously estimated pose $P_{l+1} = (t_{l+1}, ql + 1)$ with the following calculation:

$$q_l = \Delta q_l q_{l+1}, \tag{II.12}$$

$$t_l = t_{l+1} + \Delta t_l \tag{II.13}$$

II.3.5 PWCLO-Net - training loss

Quaternion q^l and translation vector t^l are produced by the network across four distinct point cloud levels. The results from each level are fed into a tailored loss function to compute the supervised loss l^l. As there exists a dissimilarity in scale and units between

the translation vector t and quaternion q, two adaptable parameters, namely s_t and s_q, are incorporated such that the training loss function at the l-th level is:

$$l^l = \left\|t_{gt} - t^l\right\| exp(-s_t) + s_t + \left\| q_{gt} - \frac{q^l}{\left\|q^l\right\|} \right\|_2 exp(-s_q) + s_q, \tag{II.14}$$

where $\|.\|$ and $\|.\|_2$ denotes the l_1-norm and the l_2-norm respectively. t_{gt} and q_{gt} are the ground-truth translation vector and quaternion respectively. Then, the total training loss l is calcualted using a multi-scale supervised approach:

$$l = \sum_{l=1}^{L} \alpha^l l^l, \tag{II.15}$$

where L is the total number of warp-refinement levels and α^l denotes the weight of the l-th level. When training, the proposed model has 4 layers ($L = 4$) and the chosen weights of the loss are $\alpha_1 = 1.6$, $\alpha_2 = 0.8$, $\alpha_3 = 0.4$ and $\alpha_4 = 0.4$

II.3.6 PWCLO-Net - experiments and results

LiDAR odometry is an interesting field of research in the domain of computer vision. A lot of work has been developed to answer to this problem. In this pursuit, many works choose to use KITTI [15] dataset for its popularity and for the availability of a large benchmark on this dataset. Thus, the metric proposed by KITTI gained an enormous popularity in LiDAR odometry because of their capability to evaluate effectively the methods and their simplicity. In this section, a short description of the KITTI dataset will presented II.3.6.1 along with a new dataset KITTI-360 [21] in order to give clear vision about the input data the model manipulates. Then, the KITTI metrics II.3.6.2 for evaluating the odometry will be explained and the experimental results will be shown and discussed II.3.6.3.

In order to standardize the evaluation method and allow a better comparison between the proposed methods, many works agree to use same datasets and same metrics. two metrics were defined to measure the performance of any proposed solution.

II.3.6.1 KITTI and KITTI-360 datasets

KITTI [15] represents an autonomous driving dataset created jointly by the Karlsruhe Institute of Technology and the Toyota Technological Institute at Chicago. It is a collection of images and LIDAR data, serving as a resource for computer vision research, including stereo vision, optical flow, visual odometry, 3D object detection, and 3D tracking. KITTI is the most important and famous dataset for SLAM research. It was publicly released in 2012 to allow researchers to conduct their experiments and develop new methods to improve autonomous driving technologies. It's a challenging real-world computer vision dataset that was collected using a standard station wagon equipped with two high-resolution color and grayscale video cameras. LiDAR raw data was captured by a Velodyne laser scanner and accurate odometry ground truth was provided by a GPS localization system. In total, the KITTI dataset provides 11 sequences for training and testing. These sequences cover a range of scenarios and environments, including urban and highway scenes, different weather conditions, and various traffic scenarios.

On the other hand, KITTI-360 [21] is another large-scale dataset that was recently released in 2020. It contains rich sensory information and full annotations for both static and dynamic 3D scene elements with rough bounding primitives. Its setup for data collection is similar to the one used in KITTI, except that it offer a full 360° by using

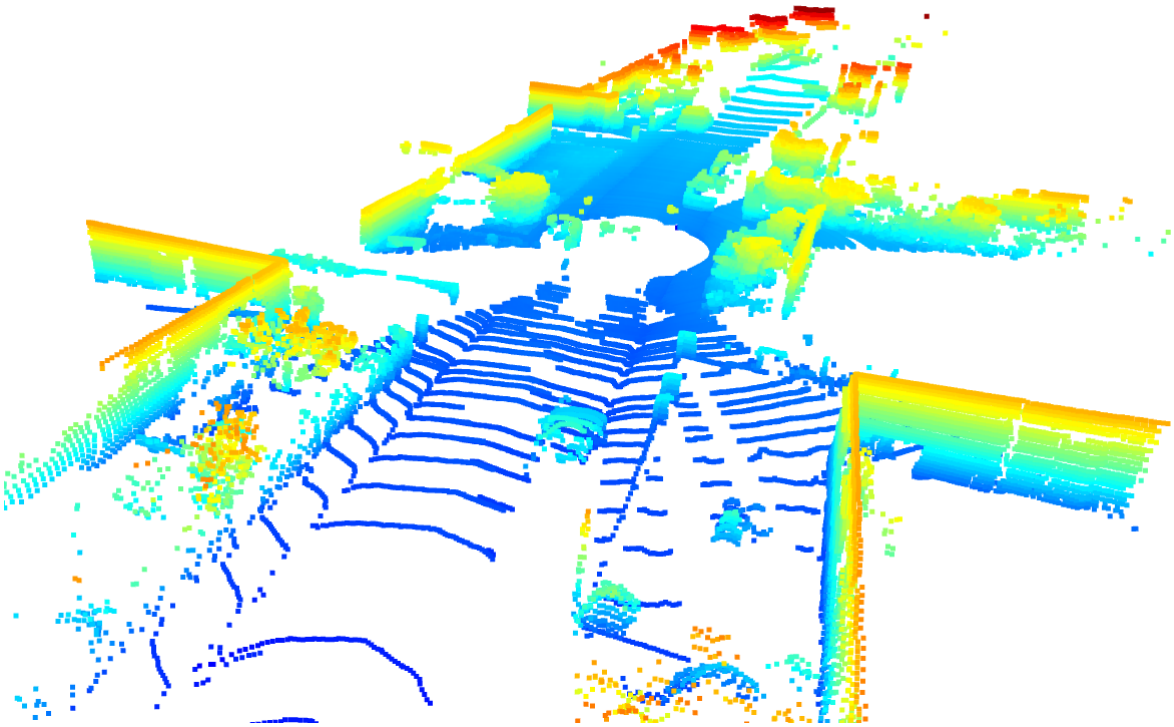


Figure II.11: Local frame point cloud visualized with colors indicating height

additional fish-eye cameras. Moreover, KITTI-360 also provides 9 sequences that were collected in the same environment as KITTI. These sequences are recognized for their increased complexity and length compared to the sequences in the KITTI dataset, as they feature a higher number of loops, leading to additional rotations. While there are a lot of pre-processing tools for the KITTI dataset, rare are the ones that are interested in KITTI-360. For this reason, and in order to have a better understanding of this new dataset, visualization and processing tools were developed. In fact, KITTI-360 provides labels only for the aggregated sub-maps II.12 and doesn't offer labels for each local point cloud II.11. To address this concern, the ground truth pose transformations were employed to align all local point clouds to a consistent world coordinate framework. The transformed point clouds were then merged to create sub-maps resembling the original ones in the KITTI-360 dataset. To extract labels from the generated sub-maps, a nearest neighbor search was conducted. To facilitate this, a *k-d tree* (see Fig. II.5a) was constructed over the sub-map to enhance space partitioning. Once nearest neighbors were identified, the queried points inherited labels from their neighboring points. The results are presented in Figure **??**.

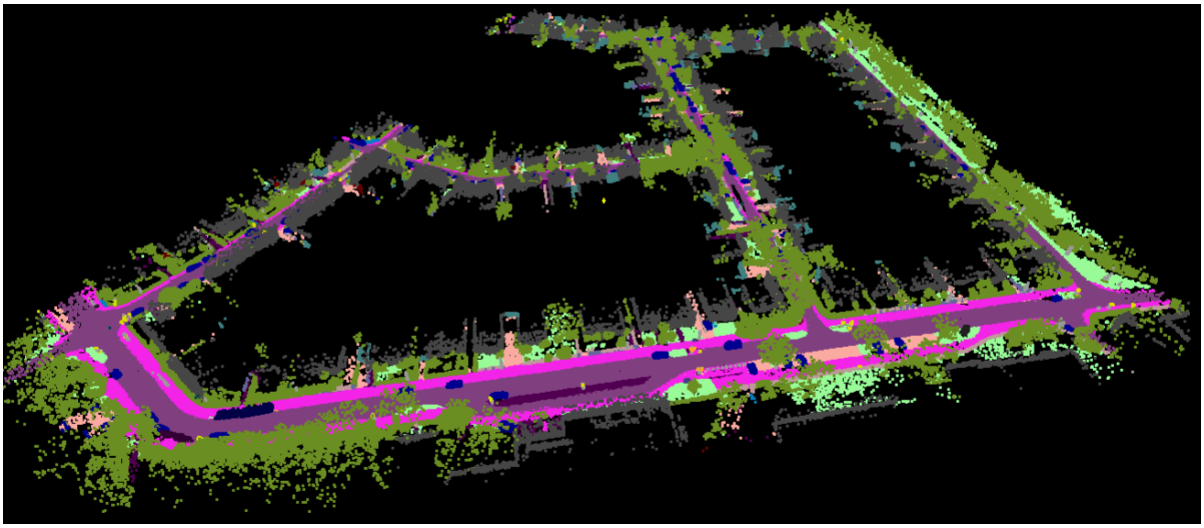
II.3.6.2 PWCLO-Net - metrics

These two metrics are average translation and rotation errors (ATE and ARE) for all possible sub-sequences of length (100,...,800) meters defined as follows:

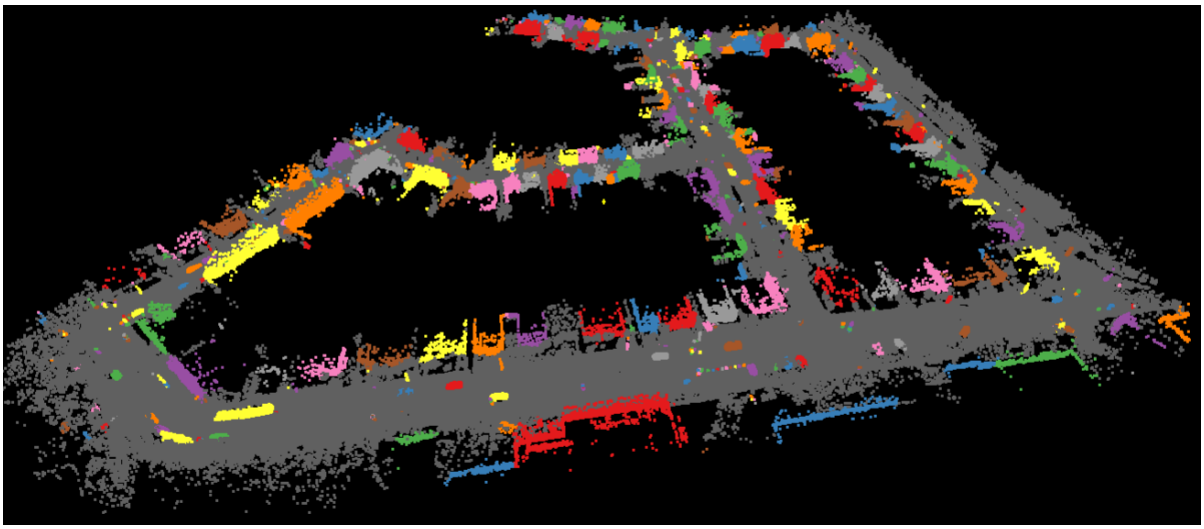
$$ATE = \frac{\sum_{f=1}^{N_{frames}} TE(f,100) + TE(f,200) + ... + TE(f,800)}{8 * N_{frames}}, \tag{II.16}$$

where N_{frames} is the total number of frames and $TE(f,l)$ represents the translation error from the frame f until the frame f_l such as the distance travelled by the ego-vehicle between f and f_l is equal to the length l ($l \in 100, 200, ..., 800$). The average translation

(a) RGB map



(b) Semantic map

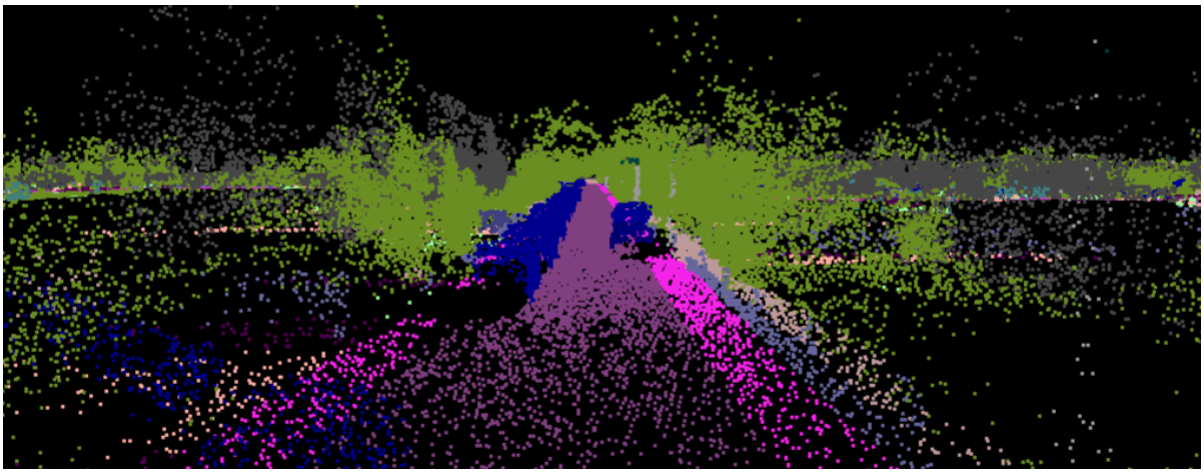


(c) Instance map

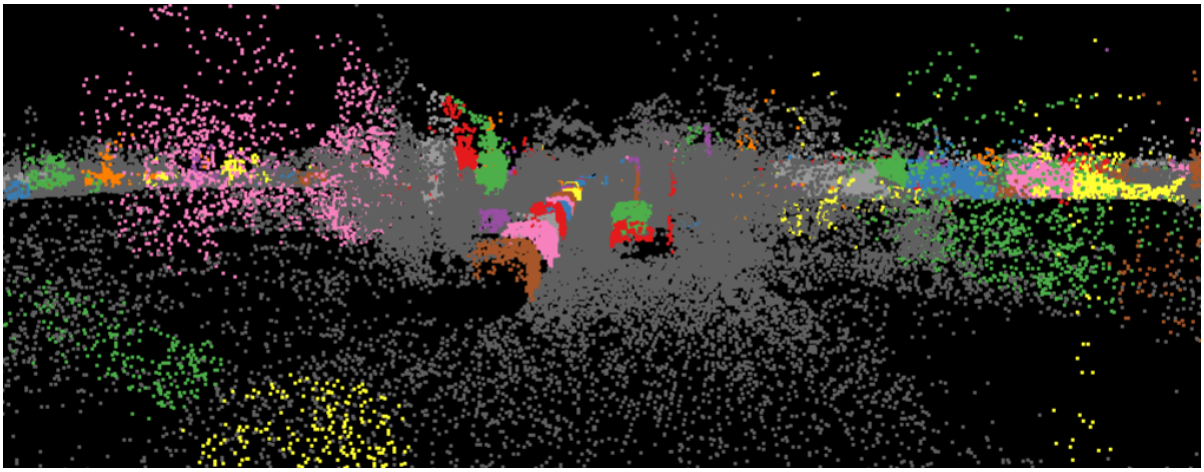
Figure II.12: KITTI-360 labeled maps

error is measured in percent (%). Similarly, the average rotational error can be defined

(a) RGB local point cloud



(b) Semantic local point cloud



(c) Instance local point cloud

Figure II.13: KITTI-360 labeled local point clouds

as follows:

$$ARE = \frac{\sum_{f=1}^{N_{frames}} RE(f, 100) + RE(f, 200) + ... + RE(f, 800)}{8 * N_{frames} * 100}, \tag{II.17}$$

where $RE(f, l)$ represents the rotational error from the frame f until the frame f_l. As shown in Eq. II.17, we further divide the rotation error by 100 to have bigger values since rotation in a normal driving scenario are too small. The average rotation error is

measured in degree per 100 meters ($\frac{\text{deg}}{100m}$).

II.3.6.3 PWCLO-Net - experimental results

In the original work, PWCLO-Net was implemented in Tensorflow version 1.12 which is not compatible with most of the research models (*e.g. pyLiDAR-SLAM* II.2) that adopt PyTorch as their main deep learning framework. Thus, in order to integrate this odometry model inside other larger frameworks and in the pursuit of potential future improvements and extended usability, we had to re-implement PWCLO-Net from scratch in PyTorch. This resulted on three main problems:

- Lack of documentation: Starting from 2019, TensorFlow transitioned to a more Python-compatible version, referred to as *Tf. v2*. Consequently, the documentation for the preceding TensorFlow version, *Tf. v1*, has substantially reduced. Notably, the approach to implementing neural networks in TensorFlow *v1* was distinct and rather complex for Python developers to grasp. It was precisely due to these considerations that TensorFlow underwent a framework upgrade. This very evolution has also played a role in PyTorch's greater popularity.

- Starting from scratch: As the majority of the layers were entirely new, we needed to re-implement each line from TensorFlow into PyTorch. This process inevitably introduced certain incompatibilities due to variations in available layers, methods, and specific engineering tools between the two frameworks. Notably, even small details such as the feature dimension diverged — PyTorch adopts a BCHW tensor representation, while TensorFlow utilizes a BHWC tensor representation (with B for batch, C for channels, H for height, and W for width). Consequently, meticulous consideration of input and output shapes for every layer becomes essential.

- Errors in the original code: In the process of comprehending the original code provided by PWCLO-Net (https://github.com/IRMVLab/PWCLONet), a number of logical errors were uncovered. Notably, discrepancies emerged between the information presented in the paper and the corresponding code, such as the case of translation refinement (as indicated in Eq. II.13). A significant error was also detected in the code, concerning the absence of feature propagation in the Siamese Point Feature Pyramid (see Sec. II.3.4). The features generated by each set abstraction layer were not accurately propagated to the subsequent set abstraction layers, as each new encoding layer solely incorporated positional features (3D coordinates) for encoding, disregarding the features previously extracted.

Conversely, certain widely used layers such as *set abstraction*, *flow embedding*, and *set up-conv* have already been implemented in PyTorch and necessitated only minor adjustments to conform to the PWCLO-Net framework. Their implementations can be examined in the GitHub repositories provided: https://github.com/yanx27/Pointnet_Pointnet2_pytorch and https://github.com/hyangwinter/flownet3d_pytorch.

To summarize, in this section, we will represent 4 different models that were trained and tested on the KITTI dataset:

1. **PWCLONET_KITTI_1**: representing our proposed PyTorch version containing feature propagation and translation refinement as explained in Eq. II.13.

2. **PWCLONET_KITTI_2**: representing the pre-trained model suggested by PWCLO-Net in there GitHub repository.

3. **PWCLONET_KITTI_3**: representing the exact PyTorch version of what was suggested in the original code. This model doesn't contain the feature propagation in the siamese point feature pyramid and the translation refinement is defined as follows:

$$[0, t^l] = q^l[0, t^{l+1}](q^l)^{-1} + [0, \Delta^l], \tag{II.18}$$

where q^l is the refined quaternion as defined in Eq. II.12 and Δ^l represents the residual translation in the l-th level as described in the iterative pose warp refinement.

4. **PWCLONET_KITTI_4**: representing our proposed PyTorch version containing the feature propagation. The translation refinement used for this model is the same as the one described in the PWCLO-Net paper [32]:

$$[0, t^l] = \Delta q^l[0, t^{l+1}](\Delta q^l)^{-1} + [0, \Delta^l], \tag{II.19}$$

where Δq^l represents the residual quaternion between the warped point cloud and the target point cloud at the l-th level.

The figure **??** displays the ground truth and predicted paths for all the sequences. The models underwent training on 7 sequences ($[0, 6]$) and were subsequently tested on 4 sequences ($[7, 10]$). The results reveal that **PWCLONET_KITTI_2** outperforms the other models. However, the proposed models exhibit slightly inferior performance compared to the pre-trained model, suggesting a requirement for further fine-tuning involving adjustments to batch sizes, scheduling, data augmentation, regularization, and other factors. Given the distinct nature of the PyTorch framework in comparison to TensorFlow, the models might necessitate different parameters, modified layers, or additional data pre-processing steps.

Table II.1: The LiDAR odometry experiment results on KITTI odometry dataset [15]. The bold values indicate the best-performing model, and the underlined values indicate the best performing model among the PyTorch proposed models.

	Training														Inference							
Method	00		01		02		03		04		05		06		07		08		09		10	
	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
ICP-po2po	6.88	2.99	11.21	2.58	8.21	3.39	11.07	5.05	6.64	4.02	3.97	1.93	1.95	1.59	5.17	3.35	10.04	4.93	6.93	2.89	8.91	4.74
ICP-po2pl	3.80	1.73	13.53	2.58	9.00	2.74	2.72	1.63	2.96	2.58	2.29	1.08	1.77	1.00	1.55	1.42	4.42	2.14	3.95	1.71	6.13	2.60
CLS†	2.11	0.95	4.22	1.05	2.29	0.86	1.63	1.09	1.59	0.71	1.98	0.92	0.92	0.46	1.04	0.73	2.14	1.05	1.95	0.92	3.46	1.28
LOAM	15.99	6.25	3.43	0.93	9.40	3.68	18.18	9.91	9.59	4.57	9.16	4.10	8.91	4.63	10.87	6.76	12.72	5.77	8.10	4.30	12.67	8.79
LO-Net	1.47	0.72	1.36	0.47	1.52	0.71	1.03	0.66	0.51	0.65	1.04	0.69	0.71	0.50	1.70	0.89	2.12	0.77	1.37	0.58	1.80	0.93
PWCLO_1	2.94	1.46	1.65	0.83	3.36	1.63	3.22	2.34	0.91	0.88	2.05	1.21	1.59	1.00	2.48	1.68	2.70	1.47	1.48	1.05	2.14	1.23
PWCLO_2	**0.80**	**0.39**	**0.33**	**0.15**	**0.97**	**0.44**	**1.37**	**0.65**	0.58	**0.39**	**0.78**	**0.40**	**0.62**	**0.25**	**0.78**	**0.54**	**1.39**	**0.62**	**0.72**	**0.36**	**1.11**	**0.72**
PWCLO_3	1.71	0.96	4.24	0.93	2.15	0.74	0.93	0.58	3.13	2.13	1.31	0.67	1.00	0.72	1.03	0.86	2.13	0.99	2.81	1.06	2.72	1.31
PWCLO_4	1.21	0.67	1.07	0.35	1.51	0.68	2.03	1.00	**0.29**	0.55	1.21	0.67	0.71	0.59	1.68	1.11	1.78	0.84	1.38	0.86	2.04	0.95

Table II.2: The LiDAR odometry experiment results on KITTI-360 dataset [21]

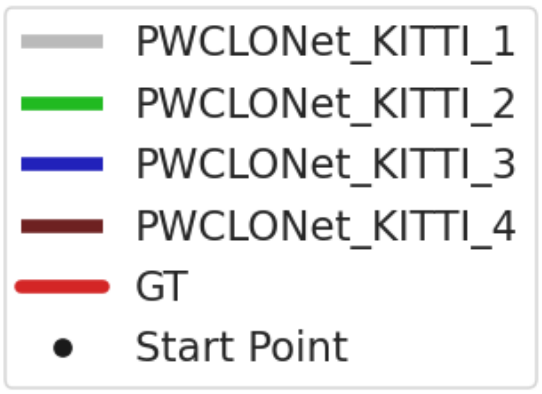
	Training												Inference			
Method	00		03		04		05		06		07		09		10	
	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
PWCLONet	3.43	1.62	2.81	1.06	2.76	1.24	2.65	1.33	2.35	1.03	3.82	1.12	2.98	1.72	8.24	2.52

The table II.1 validates the visual representations provided by the paths. The results shows that the proposed model with feature propagation and translation refinement as

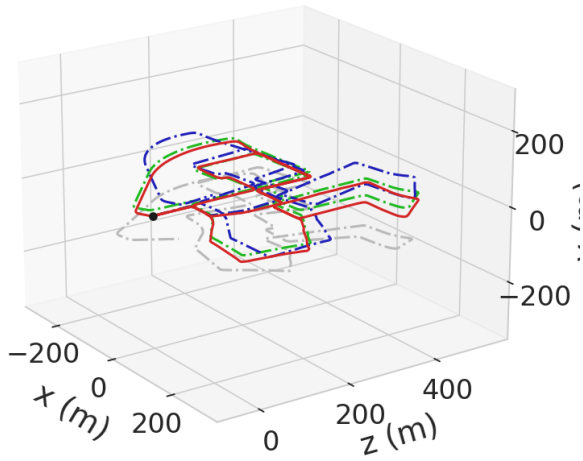
suggested in the paper (**PWCLONET_KITTI_4**) is the best model among the PyTorch proposed models.

Furthermore, this table incorporates several widely recognized LiDAR odometry methods, including *ICP-po2po* (ICP point to point), *ICP-po2pl* (ICP point to plane), *CLS* [31], *LOAM* [38] and *LO-Net* [20]. The outcomes for these methods are sourced from the PWCLO-Net paper [32]. Notably, it becomes evident that our proposed methods exhibit superior results compared to a majority of contemporary LiDAR odometry techniques. This encouraging observation suggests the potential for refining existing models and achieving enhanced performance levels.

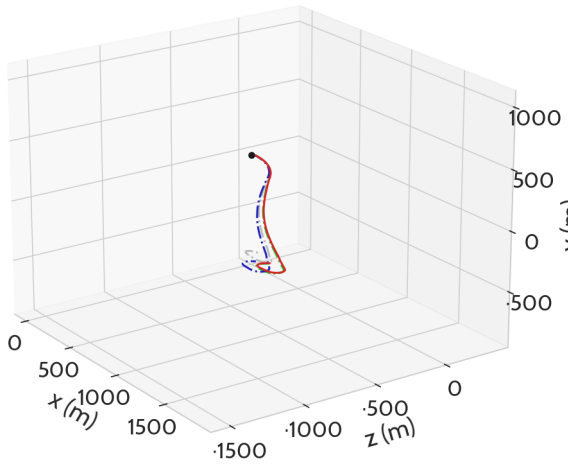
Given the recent nature of the KITTI-360 dataset, it has not been employed to assess LiDAR odometry models until now. Thus, the objective of this study is to evaluate the performance of PWCLO-Net using this new dataset and to establish the method's boundaries within the context of this intricate dataset. Our approach involves training a model on six sequences and subsequently subjecting it to testing on two separate sequences. Illustrated in Figures II.17 and II.16 are the ground truth trajectories juxtaposed with the predicted paths. A discernible trend emerges wherein the model exhibits superior performance on less intricate sequences like 03, 07, and 09, which notably lack loop closures.



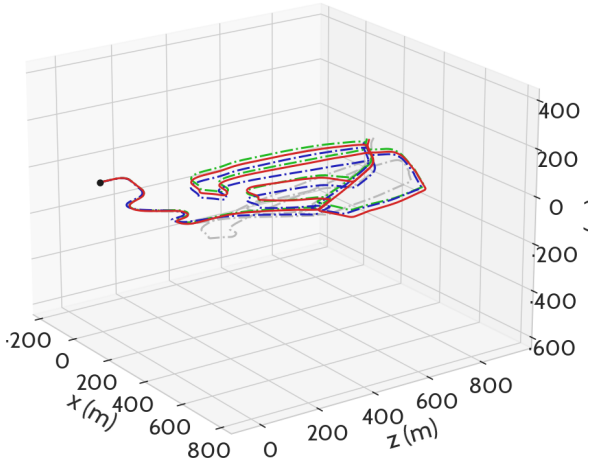
(a) Legend



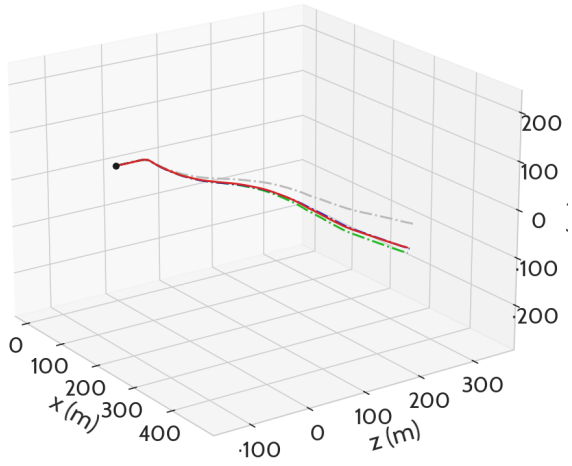
(b) Sequence 00



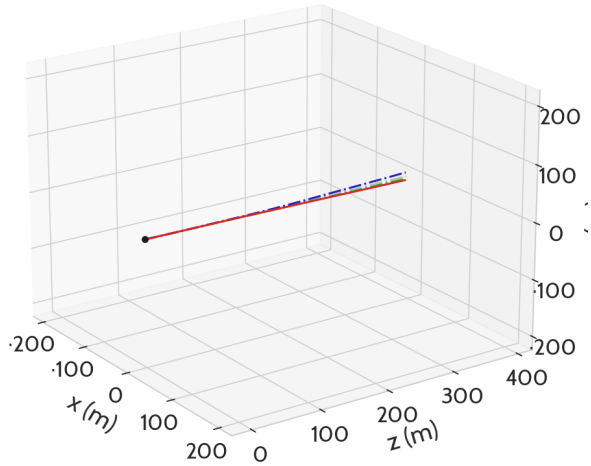
(c) Sequence 01



(d) Sequence 02

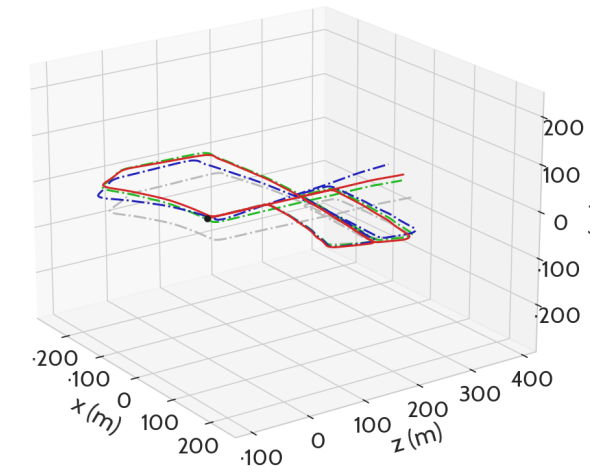


(e) Sequence 03

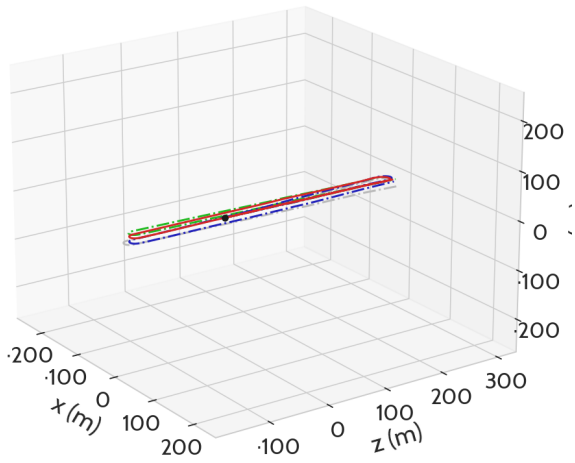


(f) Sequence 04

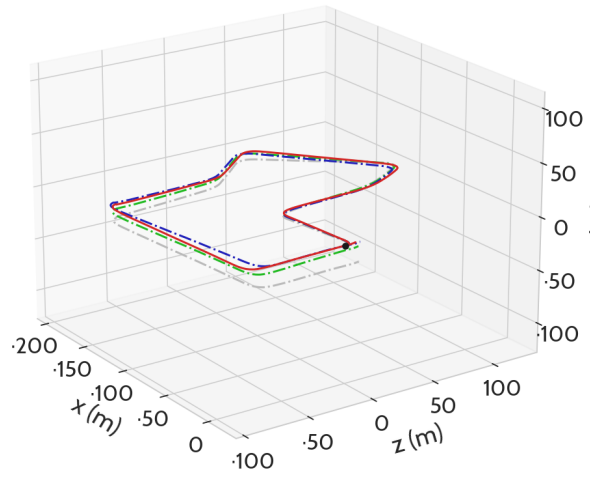
Figure II.14: Odometry evaluation on KITTI (Part1)



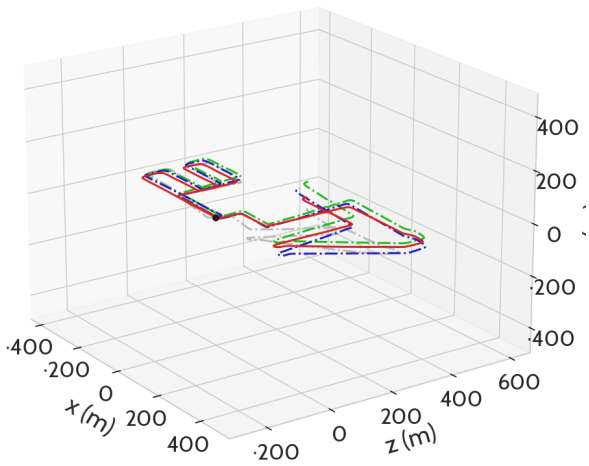
(a) Sequence 05



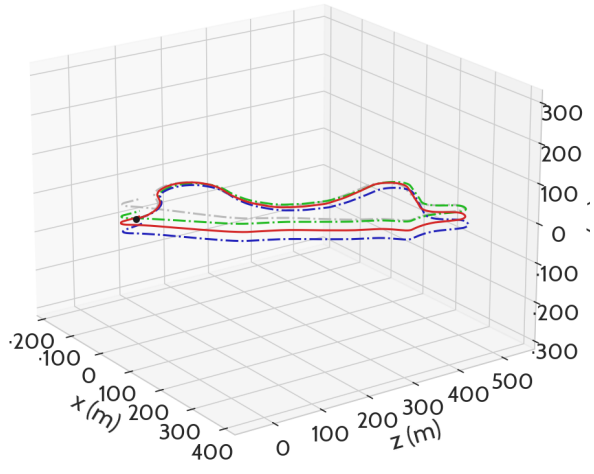
(b) Sequence 06



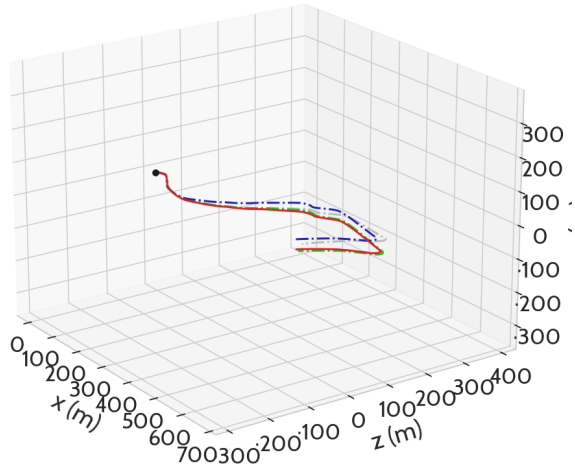
(c) Sequence 07



(d) Sequence 08

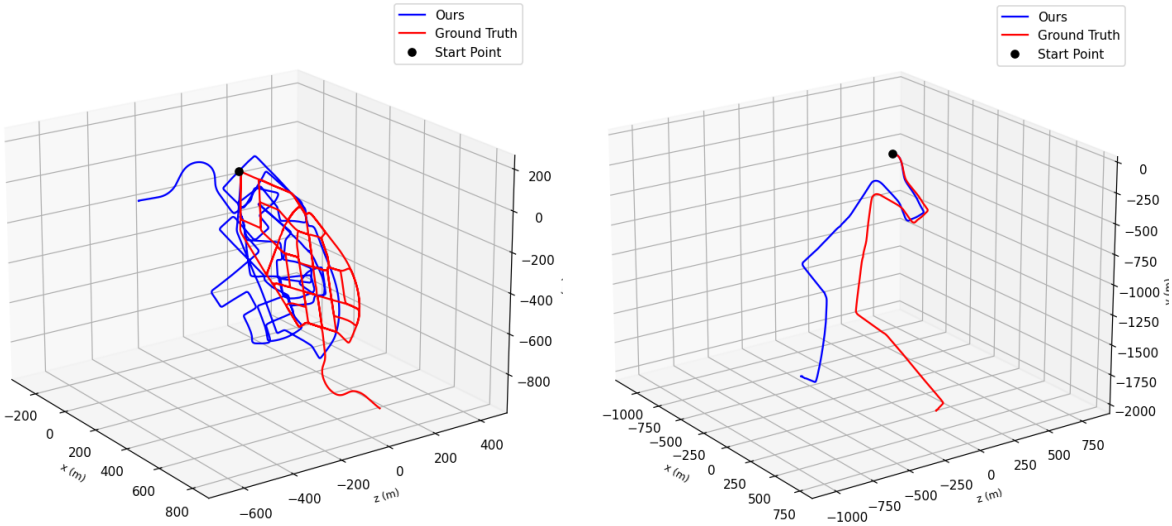


(e) Sequence 09



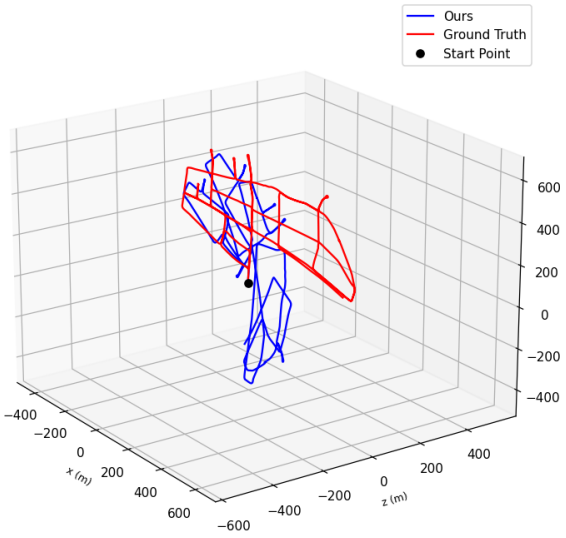
(f) Sequence 10

Figure II.15: Odometry evaluation on KITTI (Part 2)

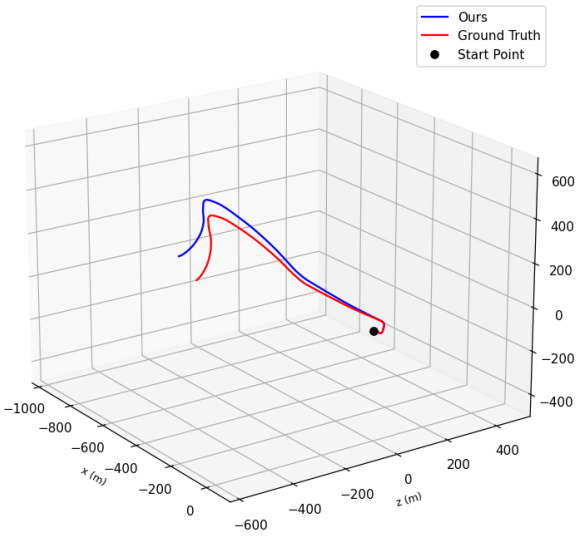


(a) Sequence 09 (b) Sequence 10

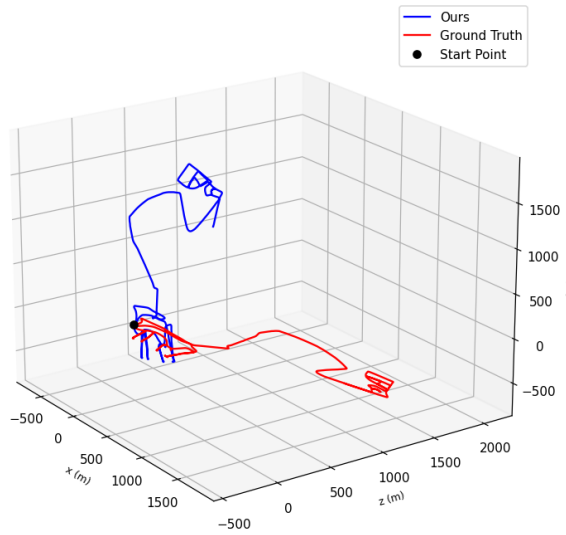
Figure II.16: Odometry evaluation on testing sequences of KITTI-360



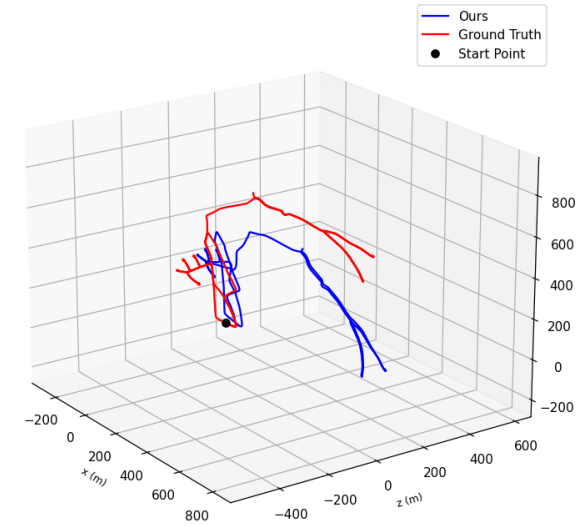
(a) Sequence 00



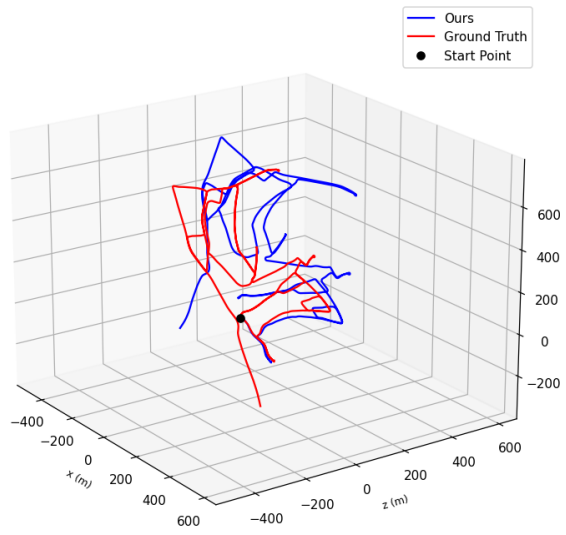
(b) Sequence 03



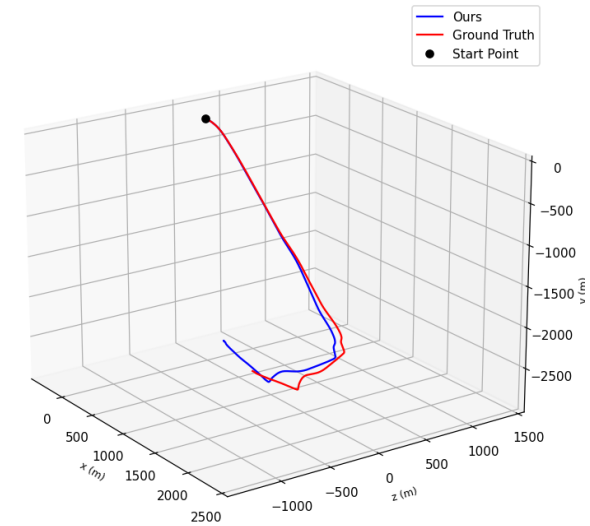
(c) Sequence 04



(d) Sequence 05



(e) Sequence 06



(f) Sequence 07

Figure II.17: Odometry evaluation on training sequences of KITTI-360

Chapter III

Neuro-Symbolic and Ontological Mapping

After estimating the vehicle's motion through LiDAR odometry, the next step in LiDAR-SLAM is to construct the map of the surrounding environment. This map serves as a representation of the physical space and the objects within it.

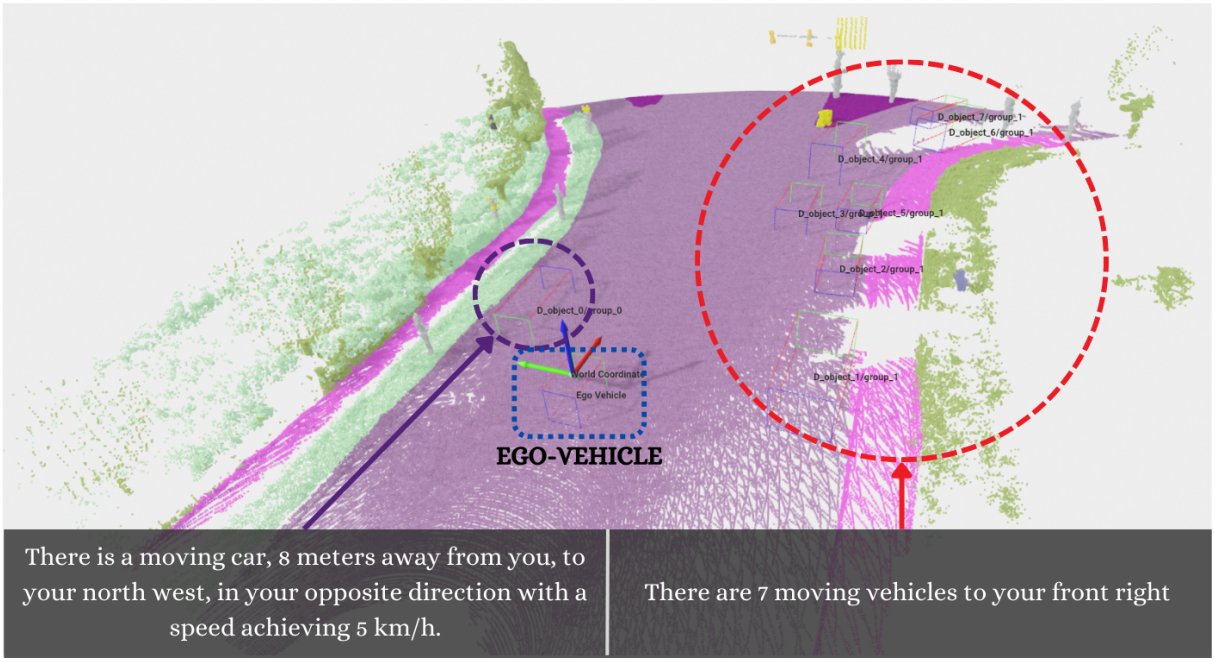


Figure III.1: Example of Neuro-Symbolic and Ontological Mapping

III.1 Background

Deep learning has been widely applied to LiDAR mapping tasks, complementing traditional approaches and offering new avenues for improved map construction and representation. Deep learning methods have shown promise in enhancing feature extraction, scene understanding, and loop closure detection, contributing to more accurate and robust mapping results. Although collecting point clouds provides a lot of information for creating maps, it also brings difficulties with storing data and describing the environment. Even though this map seems detailed and specific to the vehicle, it isn't efficient for storage and decision-making. To use it effectively, extra processing is needed for planning and decision-making tasks.

In recent years, the development of visual language models [16] has seen significant progress in the fields of 3D scene captioning and grounding [6, 40]. These advancements demonstrate the potential of uniting the powers of computer vision and large language models, such as ChatGPT [23] and others, in understanding natural human language and bridging the gap between textual descriptions and visual signals. The application of visual language models in these domains provides valuable insights into the integration of such models in the SLAM process for autonomous driving scenarios.

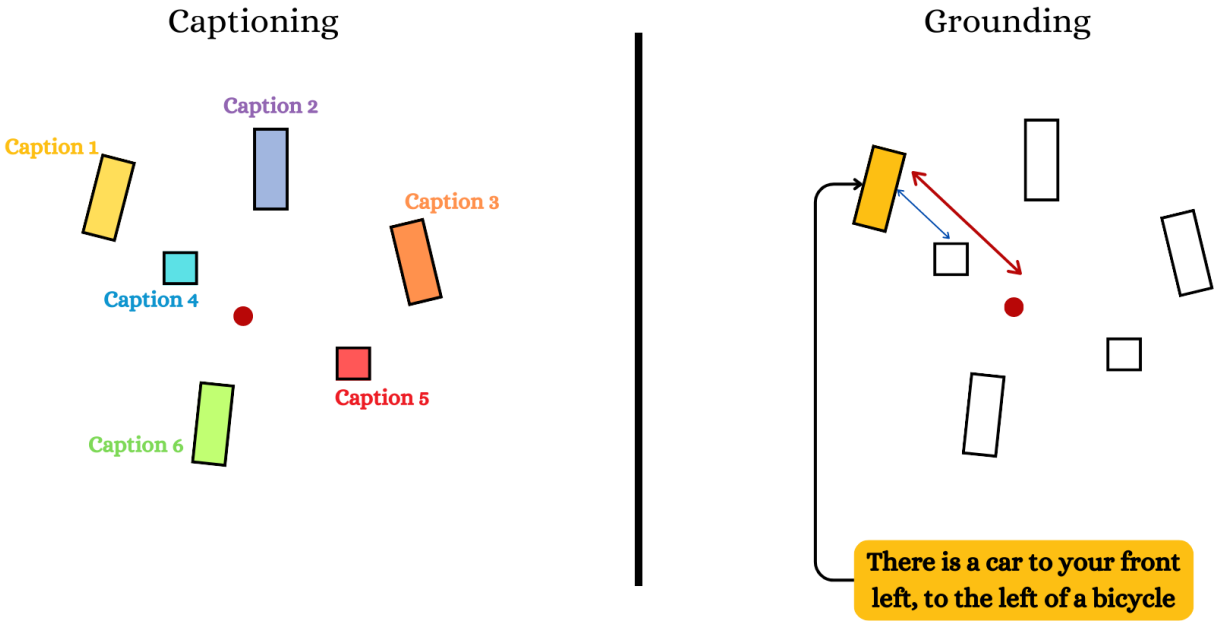


Figure III.2: Visualization of 3d dense captioning and 3d visual grounding

3D dense captioning 3D dense captioning [8, 9, 33] is the task of generating natural language descriptions for entire 3D scenes III.2. Visual language models, based on large language models [12, 18], have shown remarkable capabilities in comprehending complex scene representations and generating human-like textual descriptions. These models take advantage of their task-agnostic nature, making them versatile in understanding and processing various types of visual input, including point clouds and depth images. The combination of computer vision and natural language processing allows these models to extract meaningful features from 3D scenes and translate them into coherent textual descriptions.

3D visual grounding In the context of 3D visual grounding [17, 27, 39], the objective is to establish correspondences between textual phrases and specific objects or regions within the scene III.2. Visual language models excel in this task by learning to associate textual tokens with visual entities in an unsupervised manner. This process involves understanding the semantics of natural language instructions and grounding them to relevant visual cues. The integration of deep learning and natural language processing enables these models to comprehend complex textual descriptions and establish meaningful connections with visual elements in the 3D scene.

Visual Language and SLAM The success of visual language models in 3D scene captioning and grounding highlights their potential to contribute to the SLAM process for autonomous driving [10, 14]. The ability of these models to understand natural human

language and interpret textual descriptions opens up possibilities for closing the loop between human-generated instructions and the visual perception capabilities of autonomous vehicles.

By incorporating visual language models into SLAM systems, the vehicle gains the capacity to construct a dense and lightweight map of the environment based on textual descriptions. Instead of relying solely on vast amounts of raw point cloud data, the vehicle can leverage the power of language to create a more concise and easily manageable representation of the surrounding scene. Moreover, visual language models allow the vehicle to understand complex instructions and contextual information, enhancing its overall perception and comprehension.

III.2 Relation between neuro-symbolic and ontology

In a real-word driving scenario, the vehicle is in continuous interaction with surrounding objects. In order to be able to drive like a human, the vehicle should be aware of the other agents in the road and understand and predict the driving scene. The agents can represent other vehicles or pedestrians or even other components that rule and control the traffic such as lanes, traffic signs, traffic lights. Hence, comprehending the evolution of driving scenarios can't be separated from the actors that animate the scene. To this end, the vehicle must have a deep perception of the environment that's able to handle these elements as entities and to establish the relationship between them. This vision will offer the vehicle a rich and dense representation of the surrounding without any need for any further processing.

In this work, we refer to the entities and the relationship between them as *ontological symbols* and *ontological relations*. On the other hand, we intend to build a neural network that is able to understand the driving scene by incorporating these symbols into its thinking for what's known as *neuro-symbolic*. Neuro-symbolic [29] AI refers to AI systems that seek to integrate neural network-based methods with symbolic knowledge-based approaches. This approach will enable the model to capture relations and compositional structure, to reason symbolically with abstract concepts and to gain transparency.

III.3 Ontological mapping over 3D dense captioning

3D dense captioning is a very known computer vision problem that tries to bridge between visual perception and natural language. Recent works have achieved amazing results where they were able to describe a fully complex 3d scene and to extract the relationship between the objects. *Scan2Cap* [9] is one between many methods that are capable of bringing textual tokens and extracted visual features to the same learning space. The Fig. III.3 represents an example of a 3D dense captioning of an indoor scene. The model was able to recognize the objects and to establish the relations between them.

However, these models lack reasoning and logical thinking. Deeply in their structures, they work on extracting object-oriented features such as the localization, the size, the heading angle and uses a transformer-based module for captioning head that learns how to match between these features and the descriptive tokens. Our goal stands beyond these use-cases. We want to develop an intelligence that extracts object-wise features and learns inter-object relationships in order to conduct a logical thinking and to build a reasonable understanding of the 3D scene. The model must know how to exploit the visual information and be able to express what it sees and what it understands through textual prompt. The end goal is to teach the vehicle to think like human and to interact

with their users. In this pursuit, it is clear that object-oriented features can't solely encapsulate the complexity of 3D dynamic scene as in driving scenarios. For this reason, a supplementary module called *Relation Module* will be proposed in Sec. III.4 to learn the relationships between the entities.

Figure III.3: Scan2Cap 3D dense captioning [9]

III.4 3DJCG - 3D Joint Captioning and Grounding

3DJCG [6] made the observation that 3D dense captioning and 3D visual grounding tasks share complementary information §III.4.1. Therefore, they developed a unified framework §III.4.2 that addresses both tasks together, allowing the exchange of information between the two proxy tasks. By using shared task-agnostic modules (*i.e.*, independent of the specific task), they allow the model to learn representative features that encodes both object-oriented information and complex relations between the objects, that can further be used by two task-specific modules *i.e.* one for captioning and the other one for grounding.

III.4.1 Relation between 3D dense captioning and 3D visual grounding

In the proposed mapping approach, we intend to take advantage from the enormous success that computer vision and large language models have gained lately. To this end, we propose using a visual language that can map visual information and textual tokens into the same learning space. In the same pursuit, 3D dense captioning and 3D visual grounding appear to be complementary from the perspective of the nature of the learned information. Indeed, 3D dense captioning involves learning to generate meaningful textual descriptions from detected boxes (*i.e.*, from vision to language), while 3D visual grounding focuses on locating desired objects by comprehending provided textual descriptions (*i.e.*, from language to vision).

Empirically, it can be observed that 3D dense captioning is an object-oriented task, concentrating on acquiring precise features of the target objects while extracting only their primary relationship. Conversely, the 3D visual grounding task is more relation-oriented, prioritizing the extraction of deep connections among objects to distinguish between them, especially when they belong to the same semantic class. Therefore, it is favorable to create an integrated structure that combines the tasks of 3D dense captioning and 3D visual grounding, capitalizing on their mutual strengths to enhance the performance of both tasks.

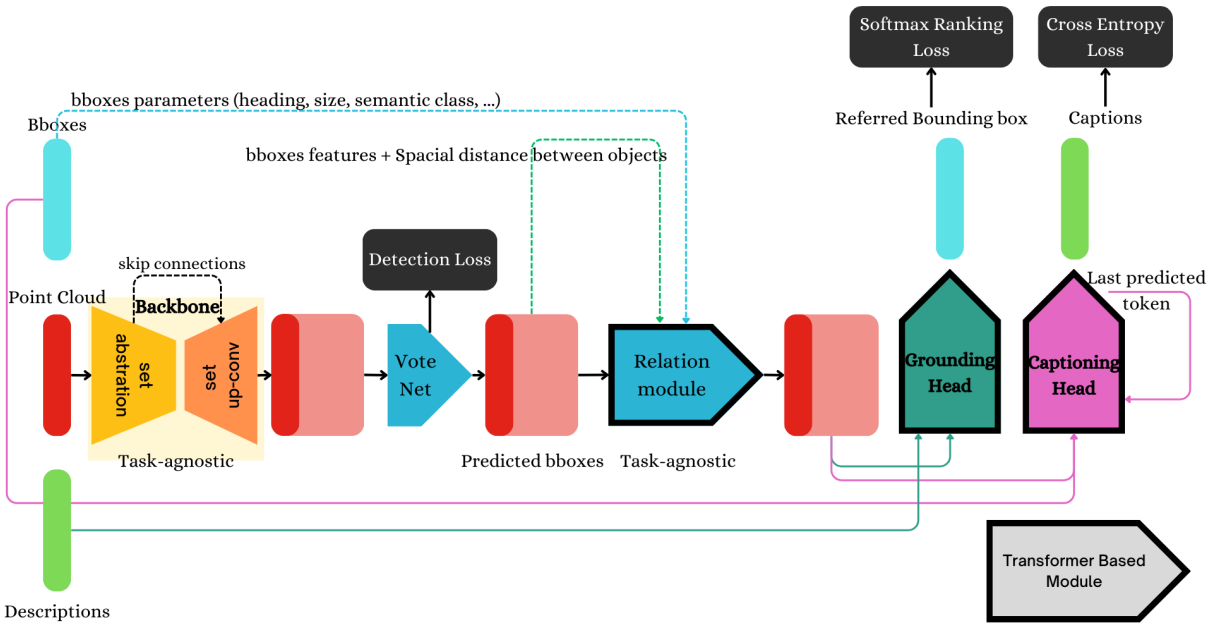


Figure III.4: 3DJCG's architecture

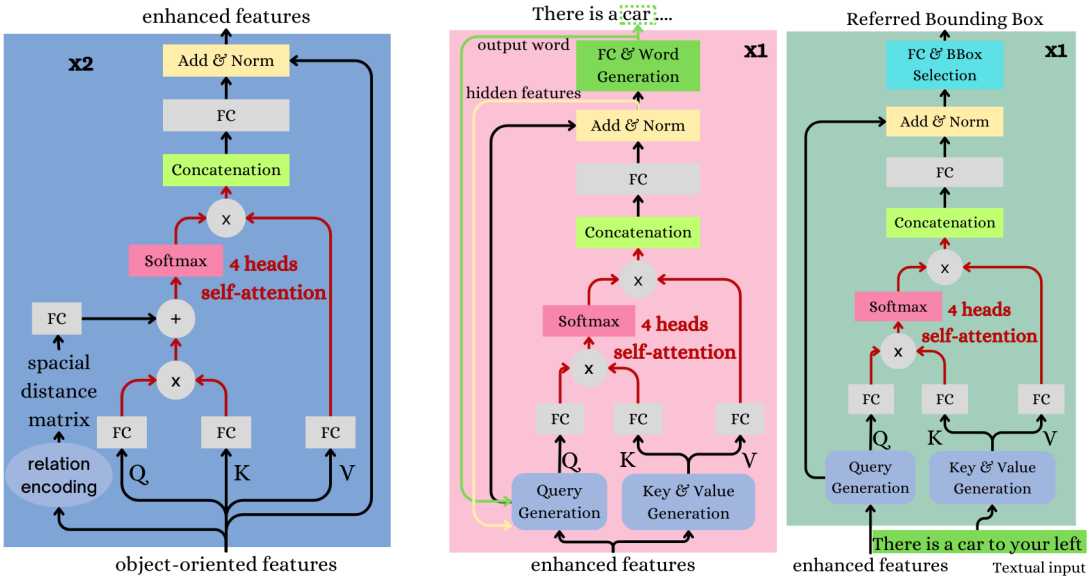
III.4.2 3DJCG - architecture

3DJCG adopts a simple architecture that is presented in Fig. III.4. It takes as input visual information (*i.e.* point cloud and bounding boxes) and textual descriptions and returns the 3d dense captioning of the scene represented by the point cloud and the referenced bounding box described by the text. To this end, it uses first a backbone to encode the information contained in the 3D point cloud. This backbone is constructed by several set abstraction layers II.3.2 followed by set up-conv layers II.3.3 to propagate the extracted features to the dense layers similarly to what was described in II.3. Then a detection module is used to predict the bounding boxes of the objects that exists in the point cloud. These bounding boxes are represented by few key parameters *e.g.* semantic class, center, size, heading angle, *etc.* that represent the object-oriented properties. The detection head that is used by 3DJCG is called *VoteNet* which builds its architecture on *deep hough voting* (for more details refer to this paper [25]). Next, these predicted bounding boxes are passed to the *Relation module* that will further add the relational features between the objects to the extracted features by the backbone. Once the complete feature map is generated, two lightweight task-specific modules will resolve the two tasks of 3D dense captioning and 3D visual grounding separately. The aforementioned backbone and relation module are both task-agnostic, while the captioning and grounding heads, as well as the relation module, employ transformer-based architectures. For more details refer to the original paper [6].

- *Relation module*
  The relation module is a novel component that was developed to enhance the object-oriented features encoded by the backbone with additional relation-oriented features. The initial features of object proposals generated by the detection module are discriminative with respect to different object classes, thanks to the detection-related loss. However, these features lack the essential deep relational information crucial for the achievement of both 3D captioning and 3D grounding. To resolve this issue, the relation module enhance these features with two multi-head self-attention layers with an additional relation encoding module as shown in Fig. III.5a. In order to capture the complex object relations and motivated by [39], the relation encoding

module encodes the pairwise distances between any two object proposals.

- *Captioning and Grounding heads*
  Given the nature of the task-agnostic modules (*i.e.*, backbone and relation module), the enhanced encoded features contain accurate and precise object-related features (*e.g.* localization, class, heading) and deep and complex relation features. This allows 3DJCG to use more lightweight task-specific captioning and grounding heads which are simpler that the state-of-the-art methods [9, 39].



(a) Relation module (b) Captioning head (c) Grounding head

Figure III.5: 3DJCG transfomer-based components

III.4.3 3DJCG - training loss

As shown in III.4, 3DJCG adopt a combination of three loss functions for training *i.e.*, *detection loss* $L_{detection}$, *captioning loss* $L_{captioning}$ and *grounding loss* $L_{grounding}$.

- Detection loss: The object detection loss is similar to the one proposed in VoteNet [25] where $L_{detection} = L_{vote_reg} + L_{objn_cls} + L_{sem_cls} + 200L_{box_reg}$.
- Captioning loss: For the dense captioning task, 3DJCG adopt a *Teacher Forcing* III.7 method in training by inputting the ground-truth words (or the predicted words with a probability of 10 %) sequentially. This will help to accelerate the training and to keep the model's predictions always close to the ground truth. Then $L_{captioning}$ is calculated as the average cross-entropy loss over all generated words.
- Grounding loss: For the visual grounding loss, *Softmax Ranking Loss* that is similar to *Cross Entropy Loss* except that it assigns probabilities to each proposal bounding box such as close objects to the one referred by the textual description will have higher probabilities. An example is shown in Fig. III.6 where referred bounding box by the textual description is the one in yellow. However, the gree bounding box can be easily confused by human since it has the same semantic class as the ground truth and is geometrically close to it. Hence, the green bounding box will have higher probability than the other objects but at the same time the highest

probability will be given to the yellow bounding box to push the model to refine its predictions. This loss will help the model to gradually understand the relational information encoded in the textual description with soft penalization.

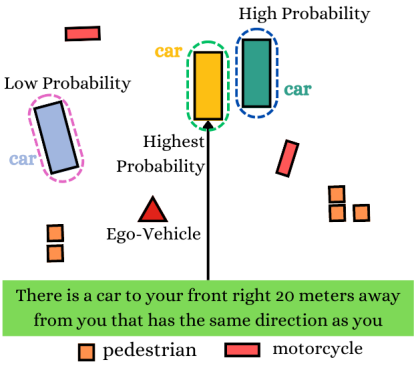


Figure III.6: Softmax Ranking Loss

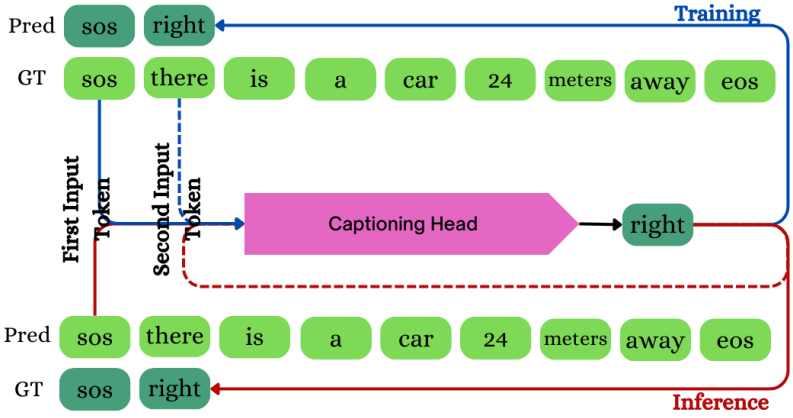


Figure III.7: Teacher forcing method

III.4.4 3DJCG - experiments and results

Since the proposed solution for mapping in autonomous driving scenarios is novel, we had to generate our own dataset *KITTI-360 Captions* that can be used for training and evaluation.

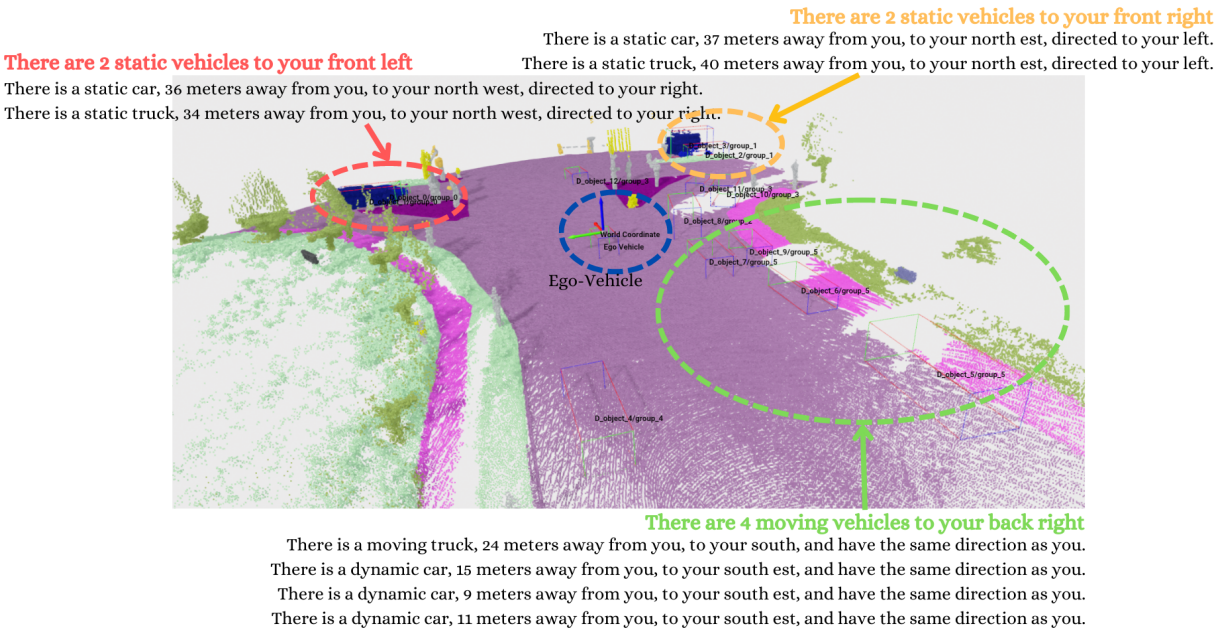


Figure III.8: 3D Scene with textual mapping

III.4.4.1 KITTI-360 Captions datasets

The dataset employs a local 3D space coordinate system with its origin positioned at the center of the ego-vehicle. The coordinate axes (X, Y, Z) are configured such that the X-axis points towards the front of the ego-vehicle, the Y-axis points towards the left of the ego-vehicle, and the Z-axis points upward. Additionally, KITTI-360 dataset provides

the bounding boxes of the objects in all the sequences. These bounding boxes contains the eight corners that define the borders of the object. In order to generate the captions, we need to extract the heading angle of the object and the distance of the object and its orientation with relatively to the ego-vehicle. The heading angle of the object is defined as angle formed between the center of the bounding box and the center of its front face as described in Fig III.9. This angle will allow us to identify the direction of the motion of the object in the scene and thus we will be able to describe its relative direction with respect to the ego-vehicle (that is always heading to the front). On other hand, the distance and the orientation angle between the object and the ego-vehicle can directly be extracted from the centers of both bounding boxes as shown in Fig. III.9.

The Fig. III.8 shows an example of the new generated dataset. It represents a real-world driving scenario with ontological mapping. The dataset offers also group-wise captions where multiple close objects that share the same semantic class and the orientation will be textually described as a group. Hence this option will enable us to have multiple levels of captioning where the first one focuses on the general view of the scene and the second is more fine-grained and describe each object separately.

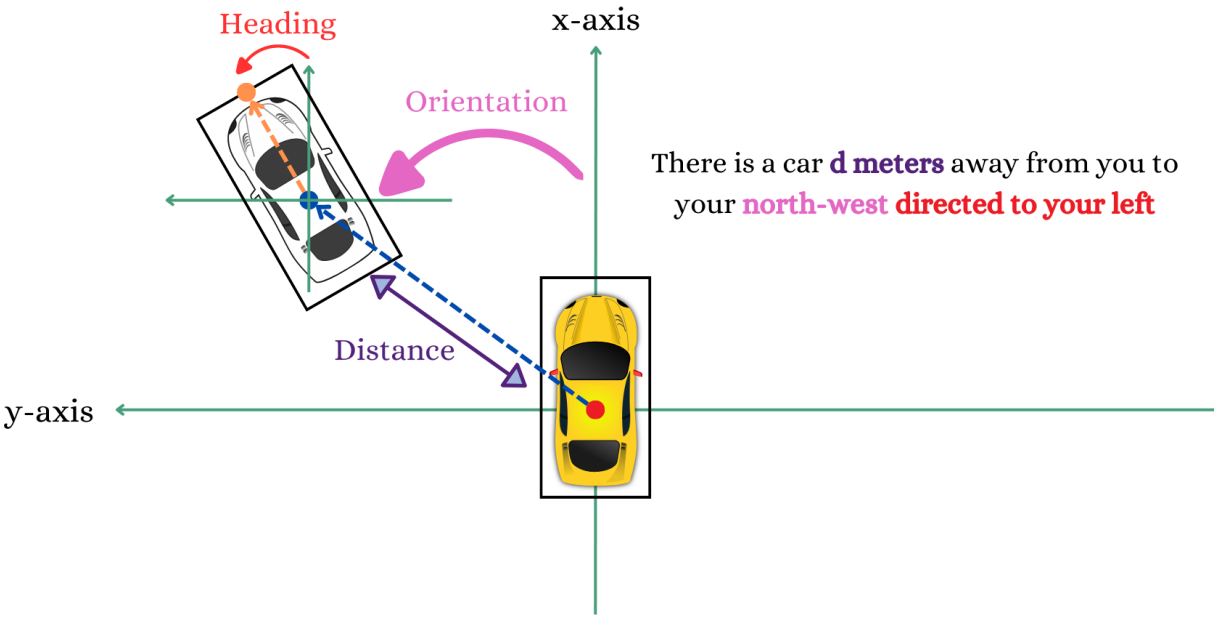


Figure III.9: KITTI-360 Captions dataset generation

III.4.4.2 Captioning and grounding metrics

In the realm of NLP (Natural Language Processing) applications (*e.g.* machine translation, chatbots, text summarization, captioning, *etc.*), many metrics were proposed to evaluate the accuracy of the predictions. However, *BLEU* [24] and *METEOR* [4] score have gained a large popularity for their simplicity and their similarity to human evaluation.

BLEU [24] score (*BiLingual Evaluation Understudy*) is a very popular metric that is mainly used in machine translation. It's far from being perfect but it's very used for it's simplicity and its interpretability. It takes values between 0 and 1 and usually a score of 0.6 or 0.7 is considered the best a model can achieve. Even two humans would likely come up with different sentence variants for a problem, and would rarely achieve a perfect match. It is defined as the product of the *Brevity Penalty* and the *Geometric Average of*

the Precision Scores which combines precision scores for different n-gram lengths (subsequences of n words):

$$\textbf{BLEU} = \textbf{Brevity Penalty} * \textbf{Geometric Average of the Precision Scores}. \quad \text{(III.1)}$$

Each precision $\mathbf{P}_N$ score represents the overlap of n-grams between the machine-generated translation and the reference translation as follows:

$$P_N = \frac{\text{number of overlapping n-grams between the machine} - \text{generated sentence}}{\text{number of total N-grams in the predicted sentence}}. \quad \text{(III.2)}$$

Higher n-gram lengths (*e.g.*, Trigrams, 4-grams) capture more contextual information about phrases and structures in the machine-generated sentence. While, lower n-gram lengths (*e.g.*, Unigrams, Bigrams) capture individual words and basic word pairs. Additionally, in order to avoid favoring overly short generated sentences, the score takes into account the length of the generated text by using brevity penaltiesdefined as follows:

$$\textbf{Brevity Penalty} = \begin{cases} 1, & \text{if Brevity Ratio} > 1 \\ e^{(1-\text{Brevity Ratio})}, & \text{if Brevity Ratio} \leq 1 \end{cases}, \quad \text{(III.3)}$$

where Brevity Ratio $= \frac{\text{predicted length}}{\text{reference length}}$

While the BLEU score provides a widely used and easily computable measure for assessing text generation quality, it has notable limitations. Its focus on n-gram precision might not capture fluency, syntactic correctness, or semantic nuances. BLEU is also sensitive to variations in word order and may favor overly conservative translations. Additionally, it only considers perfect matching between words and ignore synonyms that have the same meaning.

To overcome some of its limitations, *METEOR (Metric for Evaluation of Translation with Explicit ORdering)* [4] was introduced. It is calculated as the product of a harmonic mean between precision and recall (where recall weighted 9 times more than precision) with a penalty that favors candidates having longer matching n-grams as follows:

$$\textbf{METEOR} = F_{mean} * (1 - p), \quad \text{(III.4)}$$

where $F_{mean} = \frac{10PR}{R+9P}$ such as

$$R = \frac{\text{number of overlapping unigrams between the machine} - \text{generated sentence}}{\text{number of total N-grams in the reference sentence}} \quad \text{(III.5)}$$

and

$$P = \frac{\text{number of overlapping unigrams between the machine} - \text{generated sentence}}{\text{number of total N-grams in the generated sentence}}. \quad \text{(III.6)}$$

The research on overlapping unigrams can consider synonyms or plural versions of the word by utilizing a *stemmer*. The penalty p is calculated as follows:

$$p = 0.5 \left(\frac{c}{u_m} \right), \quad \text{(III.7)}$$

where c represents the number of chunks (longest matching n-grams) and u_m is the number of unigrams in the reference sentence. Thus, the penalty has the effect of reducing the F_{mean} by up to 50 % if there are no bigram or longer matches. One example can be given

for a reference sentence *"there is a car to your left"* with two candidates C_1 :*"to your left, there is a car"* and C_2 :*"a car is there to your left"*. C_1 has two chunks (*"to your left"* and *"there is a car"*), while C_2 has 4 chunks (*"a car"*, *"is"*, *"there"* and *"to your left"*). This will result on giving a higher score for C_1 favoring better syntax and correct structures.

On the other hand, in order to measure the performance of the model in 3D visual grounding, a reference accuracy metric is calculated as follows:

$$ref_acc = \frac{\text{number of correct referenced bounding boxes}}{\text{total number of ground truth descriptions}}. \tag{III.8}$$

In order to enhance the assessment of the referenced object's quality, we compute the *Intersection over Union* (IoU) score between the suggested bounding box for the referenced object and the actual ground truth bounding box for the object being described.

III.4.4.3 Experimental results

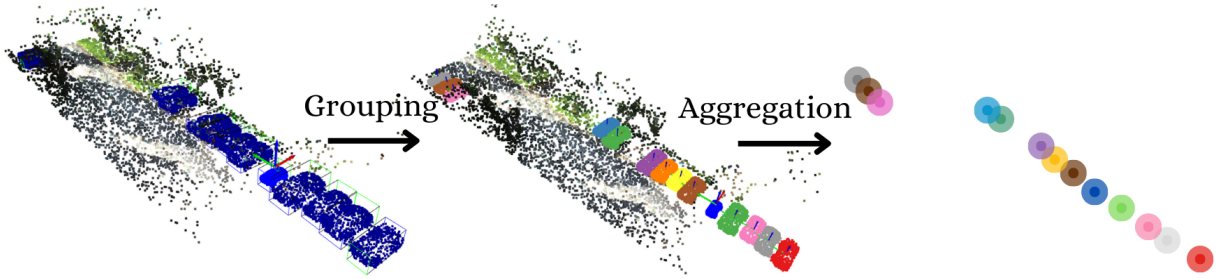


Figure III.10: Illustration of enhanced feature aggregation

Differently to 3DJCG, we trained our model with ground truth bounding boxes. This will allow us to focus on evaluating the model's performance in understanding the visual information it gets by removing the eventual error the detection head can have. In order to allow this training approach, some changes have been made to the model especially in the *Relation Module*. First by discarding the detection head from the architecture, the model is not anymore able to encode the object-oriented features that were regularized by the use of the detection loss. To resolve this issue, the ground truth bounding boxes parameters (*i.e.*, semantic class, center, heading angle, size, *etc.*) are now passed to multiple fully connected layers and aggregated to the encoded features in order to enhance the object-oriented properties. On the other hand, since we are also considering the relationship between the objects and the ego-vehicle, we further compute the distances between objects and the ego-vehicle as part of the *Relation Encoding Module* (see Figure III.5a). Following this, the generated enhanced features undergo grouping and aggregation at the centers of the bounding boxes, employing a methodology akin to that utilized in the *Set Abstraction* layer (refer to section II.3.2). The distinguishing aspect is that the specific points of interest are predetermined, corresponding to the centers of the bounding boxes. In this context, the grouping process is facilitated by identifying points situated within the respective bounding boxes (see Figure III.10). This approach diminishes the quantity of points allocated to the grounding and captioning heads, leading to an augmentation of the bounding box features, resulting in a denser feature map. Next, the new enhanced features are passed to the multi-head self-attention layers as described in II.3.4 in order to be passed later to the grounding and captioning heads.

After undergoing training on five distinct sequences, the results achieved during evaluation on the two designated test sequences attest to the commendable efficacy of the proposed method. As illustrated in Figure III.11, the trained model aptly translated the

input visual data into the anticipated textual descriptions, thus exhibiting a high degree of accuracy. This accomplishment extended to capturing object-oriented attributes, encompassing parameters like distance and semantic class. Remarkably, the model also adeptly extracted relational features, notably discerning the spatial orientations of objects relatively to the ego-vehicle. This proficiency is exemplified in phrases such as *"have the same direction as you"*, *"in your opposite direction"*, and *"to your right"*.

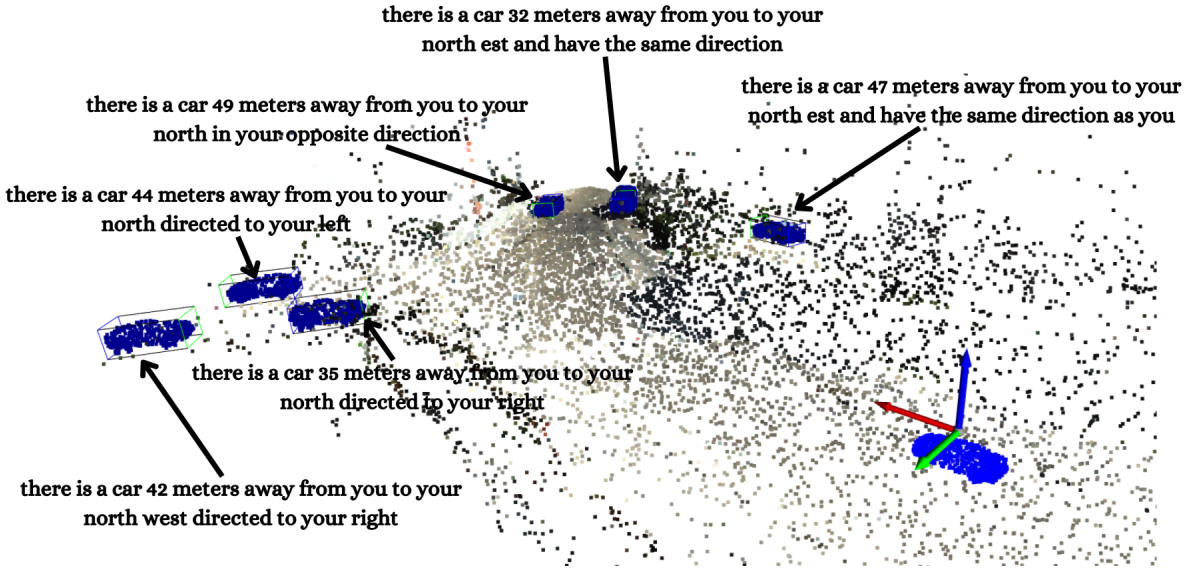


Figure III.11: Example from evaluation of 3DJCG on 3D dense captioning

Table III.1 affirms these observations, displaying that the trained model achieves an average *BLEU* score of 0.55 across a range of N-grams. This score is notably commendable, as mentioned earlier, considering that a value of 0.6 is widely regarded as the optimal achievable score. Additionally, in the context of the 3D visual grounding task, the model excels in comprehending textual descriptions and adeptly aligning them with the visual cues it receives as input, boasting a reference accuracy of 0.93. Notably, for this task, the model's input comprises ground truth bounding boxes, rendering Intersection over Union calculations superfluous, since these bounding boxes align perfectly with the ground truth.

Table III.1: The 3D Neuro-Symbolic and Ontological Mapping experiment results on KITTI360 Captions dataset

	3D Dense Captioning					3D Visual Grounding
Method	BLEU-1	BLEU-2	BLEU-3	BLEU-4	mean BLEU	reference accuracy
3DJCG	0.59	0.56	0.55	0.53	0.55	0.93

Because we're looking at simple driving situations with easy descriptions, the model seems to be doing a good job. This project is just the start of a bigger effort. People can use what we've done here to make more improvements and do even more in the future.

Chapter IV

Conclusion and Future Work

The work presented here marks the initial step towards a fresh method in LiDAR SLAM. This innovative approach is believed to advance autonomous driving technology beyond traditional brute-force machine learning. Our vision is to enable vehicles to comprehend unfamiliar and exceptional driving scenarios, mirroring human-like reasoning. Our aim is to narrow the gap between machines and humans by infusing AI with all human capabilities. Our method leverages both visual and textual cues, harnessing the strength of computer vision to analyze intricate visual scenes and the prowess of large language models to grasp human language. As a result, the model can learn a visual language similar to human perception, fusing information into a unified learning space for decision-making.

While our experiments concentrated on simple driving scenarios, they set the groundwork for deeper research in this intriguing approach. We crafted a new dataset suitable for training and evaluating our models. Its derivation from popular datasets like KITTI and KITTI-360 enhances its utility, as it can encompass the entire LiDAR SLAM challenge, from odometry and localization to mapping and scene understanding. For instance, the proposed PWCLO-Net can aid in vehicle localization within unfamiliar surroundings, enabling accurate mapping. Without localization, matching previously seen scenes with current ones becomes unfeasible, highlighting the significance of odometry for precise mapping.

We hold the belief that the flow embedding features learned by PWCLO-Net can be used to identify dynamic and static objects, similar to the concept in *PointFlowNet* [5]. These correlation features could also be employed in 3DJCG to enhance scene understanding and predict object motion, advancing 3D dense captioning. This holistic comprehension empowers the vehicle with a comprehensive perception of its surroundings, fostering complete autonomy. Combining the two models and their abilities to understand distinct scene aspects is viable. Joint training in an end-to-end manner could enhance the overall scene understanding on multiple levels.

Bibliography

[1] k-d tree - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/K-d_tree. [Accessed 21-08-2023]. 12

[2] Odometry - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Odometry. [Accessed 21-08-2023]. 7

[3] PeRL: The Perceptual Robotics Laboratory at the University of Michigan — Home — robots.engin.umich.edu. https://robots.engin.umich.edu/Main. [Accessed 25-08-2023]. 10

[4] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005. 38, 39

[5] Aseem Behl, Despoina Paschalidou, Simon Donné, and Andreas Geiger. Pointflownet: Learning representations for rigid motion estimation from point clouds, 2019. 43

[6] Daigang Cai, Lichen Zhao, Jing Zhang, Lu Sheng, and Dong Xu. 3djcg: A unified framework for joint dense captioning and visual grounding on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16464–16473, 2022. 32, 34, 35

[7] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. University of Michigan North Campus long-term vision and lidar dataset. *International Journal of Robotics Research*, 35(9):1023–1035, 2015. 10

[8] Sijin Chen, Hongyuan Zhu, Xin Chen, Yinjie Lei, Gang Yu, and Tao Chen. End-to-end 3d dense captioning with vote2cap-detr. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11124–11133, 2023. 32

[9] Zhenyu Chen, Ali Gholami, Matthias Nießner, and Angel X Chang. Scan2cap: Context-aware dense captioning in rgb-d scans. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3193–3203, 2021. iv, 32, 33, 34, 36

[10] Wenhao Cheng, Junbo Yin, Wei Li, Ruigang Yang, and Jianbing Shen. Language-guided 3d object detection in point cloud for autonomous driving, 2023. 32

[11] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. What's in my lidar odometry toolbox? In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4429–4436. IEEE, 2021. 10

[12] Michael R. Douglas. Large language models, 2023. 32

[13] Taosha Fan and Todd Murphey. Generalized proximal methods for pose graph optimization. In *The International Symposium of Robotics Research*, pages 393–409. Springer, 2019. 10

[14] Daocheng Fu, Xin Li, Licheng Wen, Min Dou, Pinlong Cai, Botian Shi, and Yu Qiao. Drive like a human: Rethinking autonomous driving with large language models, 2023. 32

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 10, 20, 25

[16] Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot navigation, 2023. 32

[17] Shijia Huang, Yilun Chen, Jiaya Jia, and Liwei Wang. Multi-view transformer for 3d visual grounding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15524–15533, June 2022. 32

[18] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models, 2023. 32

[19] Jack B Kuipers. *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton university press, 1999. 16

[20] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. Lo-net: Deep real-time lidar odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8473–8482, 2019. 26

[21] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3292–3310, 2022. 10, 20, 25

[22] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 529–537, 2019. 14, 15, 16

[23] OpenAI. Gpt-4 technical report, 2023. 32

[24] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002. 38

[25] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9277–9286, 2019. 35, 36

[26] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 9, 13, 16

[27] Junha Roh, Karthik Desingh, Ali Farhadi, and Dieter Fox. LanguageRefer: Spatial-language model for 3d visual grounding. In *Proceedings of the Conference on Robot Learning*, 2021. 32

[28] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018. 8

[29] Amit Sheth, Kaushik Roy, and Manas Gaur. Neurosymbolic ai - why, what, and how, 2023. 33

[30] S. Vedula, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):475–480, 2005. 14, 15

[31] Martin Velas, Michal Spanel, and Adam Herout. Collar line segments for fast odometry estimation from velodyne point clouds. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4486–4495. IEEE, 2016. 26

[32] Guangming Wang, Xinrui Wu, Zhe Liu, and Hesheng Wang. Pwclo-net: Deep lidar odometry in 3d point clouds using hierarchical embedding mask optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15910–15919, 2021. 10, 13, 16, 17, 25, 26

[33] Heng Wang, Chaoyi Zhang, Jianhui Yu, and Weidong Cai. Spatiality-guided transformer for 3d dense captioning on point clouds. *arXiv preprint arXiv:2204.10688*, 2022. 32

[34] Weisong Wen, Yiyang Zhou, Guohao Zhang, Saman Fahandezh-Saadi, Xiwei Bai, Wei Zhan, Masayoshi Tomizuka, and Li-Ta Hsu. Urbanloco: A full sensor suite dataset for mapping

and localization in urban scenes. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 2310–2316. IEEE, 2020. 10

[35] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with resnets. *CoRR*, abs/2002.05990, 2020. 19

[36] Tao Wu, Hao Fu, Bokai Liu, Hanzhang Xue, Ruike Ren, and Zhiming Tu. Detailed analysis on generating the range image for lidar point cloud processing. *Electronics*, 10(11):1224, 2021. 8

[37] Jia Xu, René Ranftl, and Vladlen Koltun. Accurate optical flow via direct cost volume processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1289–1297, 2017. 9, 15

[38] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and systems*, volume 2, pages 1–9. Berkeley, CA, 2014. 8, 26

[39] Lichen Zhao, Daigang Cai, Lu Sheng, and Dong Xu. 3dvg-transformer: Relation modeling for visual grounding on point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2928–2937, October 2021. 32, 35, 36

[40] Dave Zhenyu Chen, Qirui Wu, Matthias Nießner, and Angel X Chang. D3net: A unified speaker-listener architecture for 3d dense captioning and visual grounding. *arXiv e-prints*, pages arXiv–2112, 2021. 32

Abstract

Efficiently mapping intricate and dynamic environments stands as a crucial pursuit in achieving full autonomous driving capabilities. LiDAR-SLAM, known for its capability in constructing dense and precise maps of unfamiliar surroundings, divides the challenge into two key facets: first, localizing the vehicle relative to the starting point using odometry, and second, leveraging this data to compile perceived point clouds into an illustrative environmental map. Yet, this method struggle with storage demands for cumulative point clouds and necessitates additional post-processing to ensure comprehensive scene understanding and secure decision-making.

To address these limitations, we propose an innovative mapping approach that employs natural language to describe maps. Our method centers around the notion of cultivating a visual language capable of mapping visual and textual data to a unified learning space, enabling direct decision-making. Drawing on the potency of computer vision to dissect intricate visual details and the success of large language models in grasping human language, our approach emulates human-like reasoning and transcends conventional brute-force learning.

In this project, we comprehensively embrace the capabilities offered by pyLiDAR-SLAM toolkit. We first present a deep learning technique, PWCLO-Net, designed for LiDAR odometry and localization. Subsequently, we introduce a novel approach, underpinned by the generation of a new dataset—KITTI-360 Captions—derived from the KITTI-360 dataset. We elucidate our method, which borrows significant components from 3DJCG, a model that synergizes the complementary insights from both 3D dense captioning and 3D visual grounding tasks. These insights are harnessed through joint training to enhance 3D scene understanding and mapping, constituting a robust foundation for our forward-looking endeavor.

Résumé

La cartographie efficace des environnements complexes et dynamiques s'impose comme une quête cruciale pour atteindre une conduite autonome totale. LiDAR-SLAM, reconnu pour sa capacité à construire des cartes denses et précises d'environnements inconnus, divise le défi en deux facettes clés : premièrement, localiser le véhicule par rapport au point de départ en utilisant l'odométrie, et deuxièmement, exploiter ces données pour compiler les nuages de points perçus en une carte environnementale illustrative. Pourtant, cette méthode peine face aux exigences de stockage pour les nuages de points cumulatifs et nécessite un traitement postérieur supplémentaire pour assurer une compréhension complète de la scène et une prise de décision sécurisée.

Pour résoudre ces limitations, nous proposons une approche de cartographie innovante qui utilise le langage naturel pour décrire les cartes. Notre méthode tourne autour de l'idée de cultiver un langage visuel capable de cartographier des données visuelles et textuelles dans un espace d'apprentissage unifié, permettant une prise de décision directe. S'appuyant sur la puissance de la vision par ordinateur pour disséquer les détails visuels complexes et sur le succès des grands modèles de langage pour saisir le langage humain, notre approche émule un raisonnement de type humain et dépasse l'apprentissage conventionnel par force brute.