Advancing Camera-LiDAR Fusion for Robust Localization

22241324 윤주현
Dept. of Electrical and Computer Engineering
College of Engineering
INHA UNIVERSITY
Incheon, S. Korea

AUTONOMOUS NAVIGATION LAB

❖ Contents

Introduction

- **There are various studies for estimating the location of vehicles.**
  - The majority of location estimation studies use LiDAR data.
  - GNSS (GPS) is widely used for autonomous vehicle localization, but is unreliable in many real-world environments.
  - Urban canyons, tunnels, dense forests, and parking garages block or degrade GNSS signals.
  - Pure SLAM (Simultaneous Localization and Mapping) methods using only cameras or LiDAR can drift over time, especially in long sequences or with few loop closures.
- **but it struggles with semantics, sparse returns at range, occlusions, and lacks rich appearance features.**
  - There is a strong need for robust, scalable localization methods that combine multiple sensors and global cues, especially for safety-critical autonomous driving.

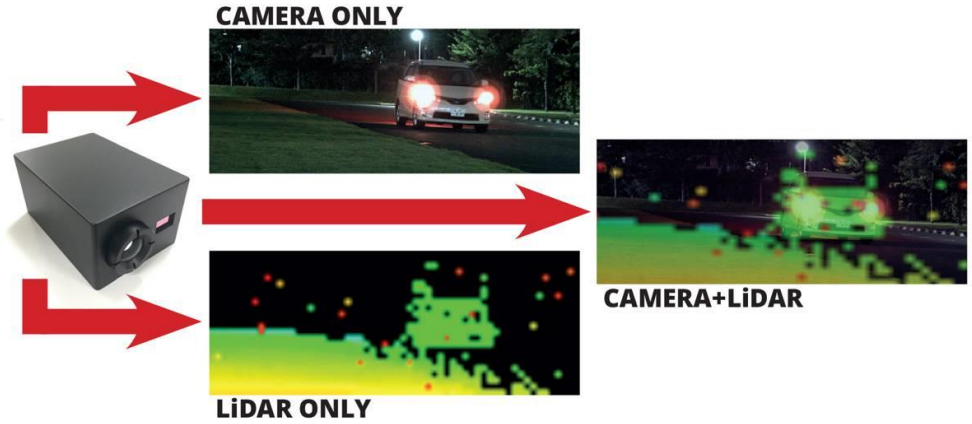
Baseline Approach Overview

- Direct Visual Odometry using image features to estimate motion over time.
- Loop Closure Detection to reduce accumulated drift using place recognition.
- Global Optimization via pose graph backend to refine trajectory consistency.
- Neural Implicit Mapping to create a dense map representation using point clouds and learned features.
- Sparse LiDAR Points are fused indirectly for scale correction and robustness.

Goals

❖ **Develop a practical, robust localization pipeline for autonomous vehicles that does not dep end on GNSS.**

- Use camera and LiDAR data for local pose estimation and correction.
- Leverage deep learning-based place recognition to provide global “anchor” points that correct drift a nd close loops.
- Make the method scalable to large environments and practical for real-world deployment.

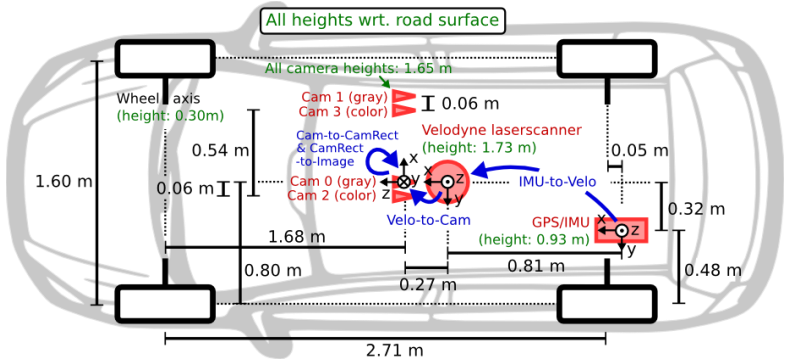


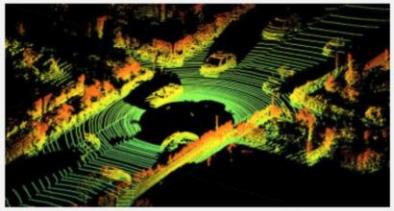
❖ Dataset: KITTI Odometry

➢ Data:

- **Camera images** (image_2): RGB camera.
- **LiDAR point clouds** (velodyne): 3D spatial info.

```
dataset
├── poses
└── sequences
    ├── 00
    │   ├── image_0
    │   ├── image_1
    │   ├── image_2
    │   ├── image_3
    │   └── velodyne
    ├── 01
    │   ├── image_0
    │   ├── image_1
    │   ├── image_2
    │   ├── image_3
    │   └── velodyne
    ...
    └── 21
        ├── image_0
        ├── image_1
        ├── image_2
        ├── image_3
        └── velodyne
devkit
└── cpp
```



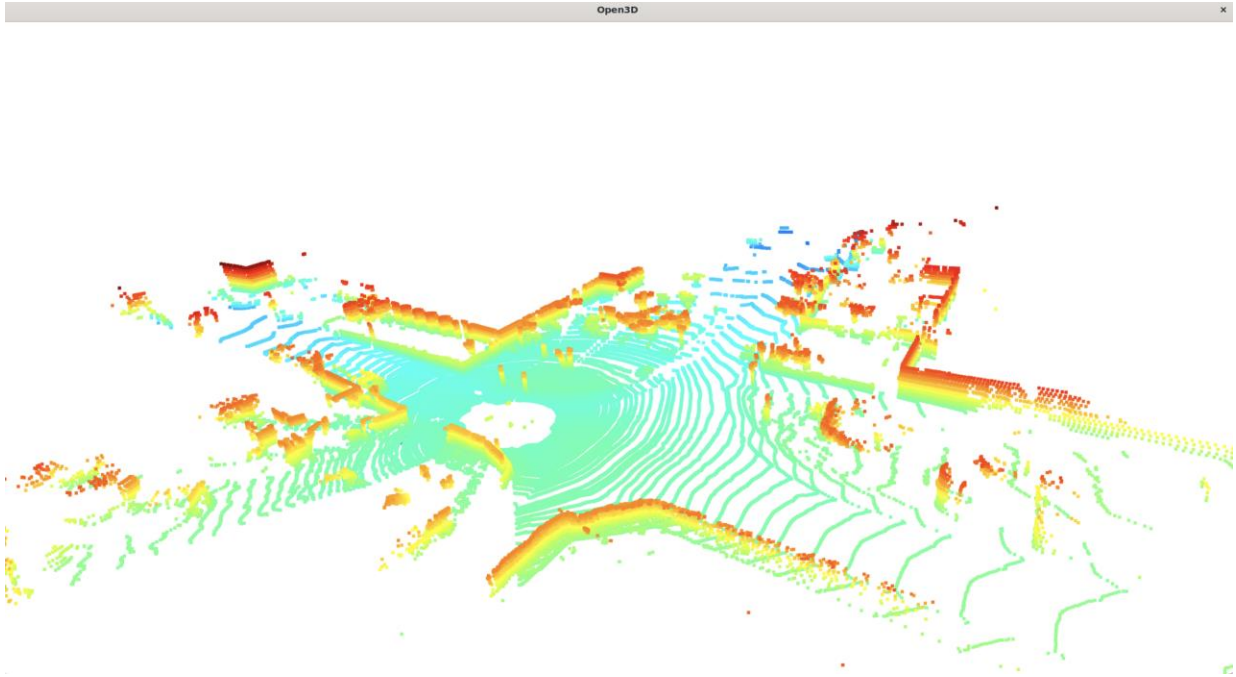



```
1 49.52000045776367 22.667999267578125 2.0510001182556152 0.0
2 49.428001403808594 22.81399917602539 2.049999952316284 0.0
3 48.09600067138672 22.658000946044922 2.006999969482422 0.05000000074505806
4 47.8129997253418 22.708999633789062 1.9989999532699585 0.0
5 48.07899856567383 23.020999908447266 2.01200008392334 0.0
6 48.21099853515625 23.270999908447266 2.0190000534057617 0.0
7 46.676998138427734 22.709999084472656 1.9639999866485596 0.0
8 46.4370002746582 22.77400016784668 1.9579999446868896 0.11999999731779099
9 46.07500076293945 22.775999069213867 1.9470000267028809 0.18000000715255737
10 45.99399948120117 22.826000213623047 1.9450000524520874 0.18000000715255737
```

Methodology-Preprocessing

❖ Filtering (Range, FOV, Validity)

- **Goal:** Remove invalid or irrelevant points.
- Operations performed:
  - Remove points behind the LiDAR (x < 0)
  - Apply a minimum/maximum range threshold (e.g., 1m < range < 80m)
  - Restrict points to a forward-looking field-of-view (typically horizontal ±60° or ±90°)



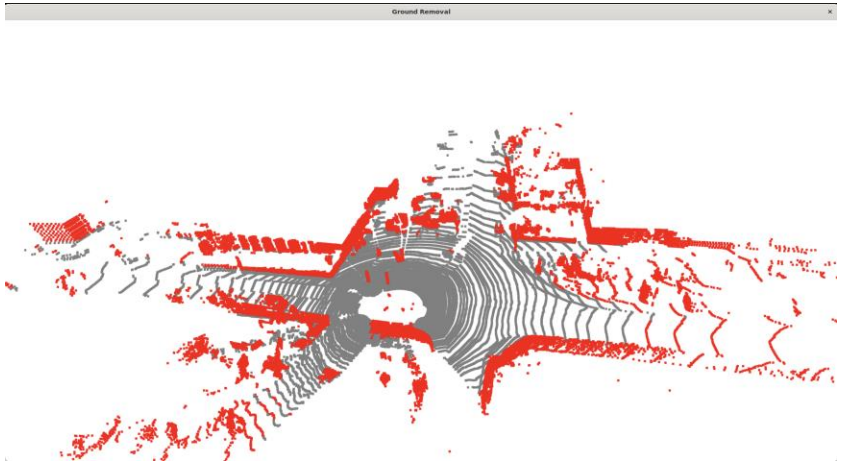
Methodology-Preprocessing

❖ Downsampling (Voxel Grid Filter)

- **Goal:** Reduce the number of points to speed up processing and reduce memory usage.
- Technique used: Voxel Grid Filtering: Divides space into 3D grid cells (voxels), and replaces all points i n a voxel with their centroid or representative.

❖ Outlier Removal

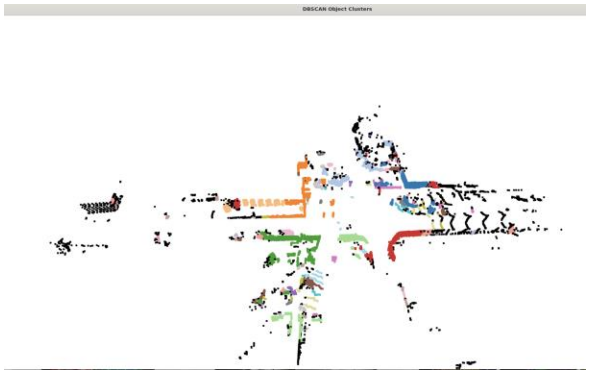
- Goal: Remove noisy or isolated points that don't represent structure.
- Technique used: Statistical Outlier Removal (SOR):For each point, compute the mean distance to its k nearest neighbors, Remove the point if this mean distance is above a threshold.

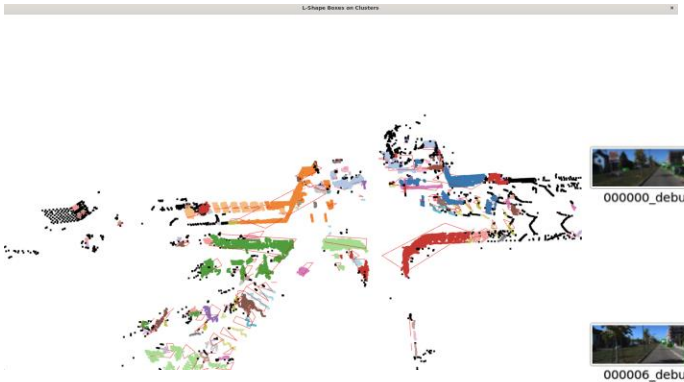


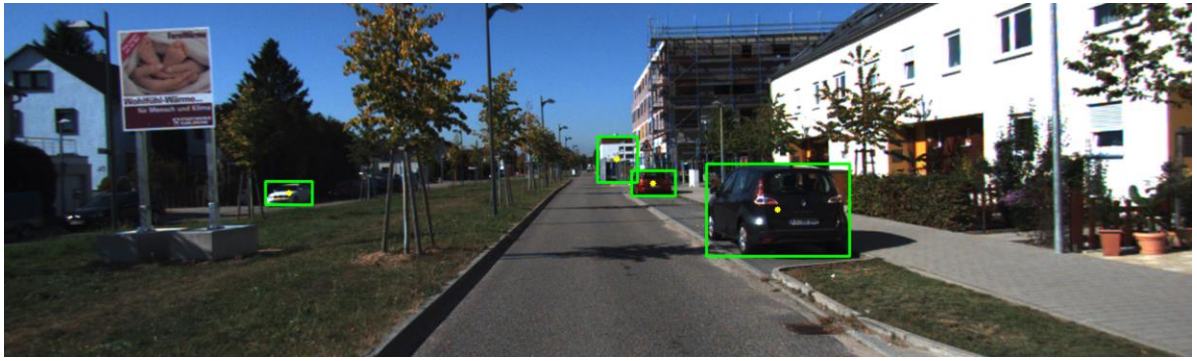
Methodology-Pose Estimation

❖ Motion Estimation

- Goal : To detect vehicles in each frame, determine their 3D positions using LiDAR, and log the results frame-by-frame.
- a 3D visualization of LiDAR point cloud data with clustered objects and L-shape bounding boxes overl aid on them



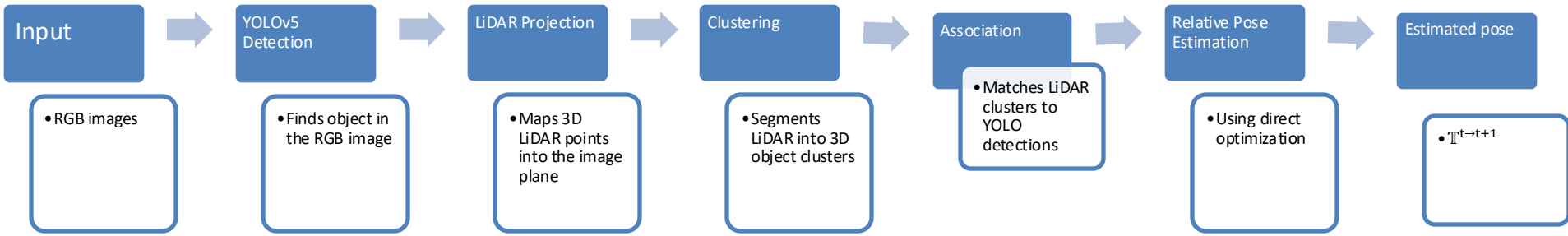




Methodology-Pose Estimation

❖ Motion Estimation

- **Goal:** Once feature matches are found between the current frame and previous frame, the system estimates the relative camera pose (rotation R and translation t) between them.
- Direct Pose Optimization:
  - Minimizing the feature reprojection error between frames:
    - ✓ $\min_{R,t} \sum_i \| f_i^{curr} - f(\pi(RX_i + t)) \|^2$, where X_i is a 3D point from map, f_i^{curr} is the feature from the current frame.
- Optimizer: Gauss-Newton optimizer



Methodology-Pose Estimation

AUTONOMOUS NAVIGATION LAB

❖ Neural Feature Descriptor

Input
- Sparse voxelized LiDAR point cloud (x, y, z)

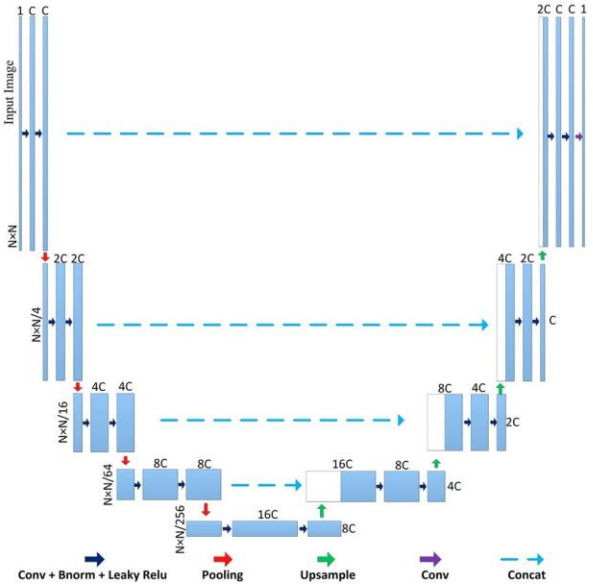
Sparse Conv + BN + ReLU

Downsample (stride 2)
- Encoder

Middle Bottleneck
- Residual Blocks

Upsample (stride 2)
- Decoder

Final 1x1 sparse conv
- 32D feature per point



Methodology- Neural Point Map

❖ Motion Estimation

➢ Initialization

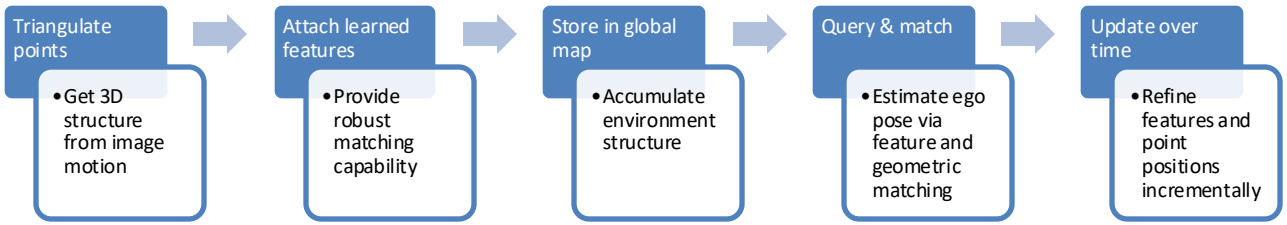
- When a frame is processed and enough parallax is detected, a new keyframe is selected.
- Keypoints are triangulated from stereo/motion, forming new 3D points.
- At each triangulated 3D location, a feature is extracted from the image.
- These (xyz, feature) pairs are inserted into the map.

➢ Update During SLAM

- As new frames arrive, The current frame queries the neural map to find matching points using feature similarity.
- Matching points are used for Pose estimation
- After pose optimization, newly seen areas are generated more map points.
- Old points are removed and their feature vectors updated via temporal averaging.

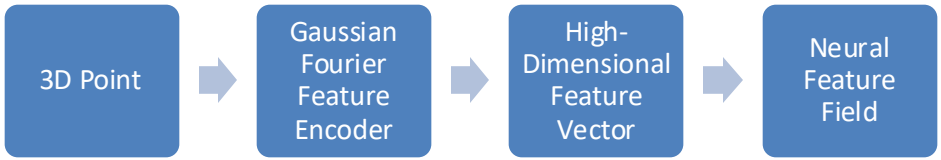
➢ map points carry learned features

- used in query radius search + feature similarity filtering



❖ Gaussian Fourier Features

- a type of positional encoding that maps low-dimensional coordinates (like 3D points) into a higher-di mensional, frequency-rich space using random Fourier basis functions. This helps MLPs (Multi-Layer P erceptrons) learn high-frequency variations more easily.
- Given a 3D point $x \in \mathbb{R}^3$, it computes its Fourier embedding:$\gamma(x) = [sin(2\pi Bx), cos(2\pi Bx)]$, wher e $B \in \mathbb{R}^{D\times 3}$ is a random Gaussian matrix with entries sampled from $\mathcal{N}(0, \sigma^2)$, D is the embedding di mension (32), σ controls the frequency scale.




```python
class GaussianFourierFeatures(nn.Module):
    def __init__(self, config: Config) -> None:
        super().__init__()

        self.freq = torch.tensor(config.pos_encoding_freq)
        self.num_bands = config.pos_encoding_band
        self.dimensionality = config.pos_input_dim

        self.register_buffer(
            "B", torch.randn([self.dimensionality, self.num_bands]) * self.freq
        )

        self.out_dim = self.num_bands * 2 + self.dimensionality

    def forward(self, x):
        x_proj = (2.0 * torch.pi * x) @ self.B
        return torch.cat([x, torch.sin(x_proj), torch.cos(x_proj)], axis=-1)

    def featureSize(self):
        return self.out_dim
```

Methodology-Local Map Extraction

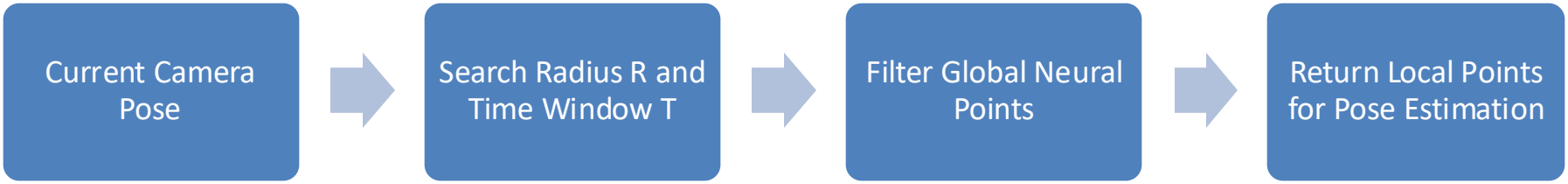
AUTONOMOUS NAVIGATION LAB

❖ Radius-Based Selection

- Select all map points within a fixed Euclidean distance (radius) from the current estimated camera pose.

❖ Time-Based Selection

- Keep only the most recent N keyframes within a fixed time window.



❖ Neighbor Search

- Finds the k closest points to a given 3D query location in the global map.

❖ Inverse-Distance Weighting

- Once neighbors are found, each neighbor contributes a weighted amount to the aggregated feature, with closer points contributing more.

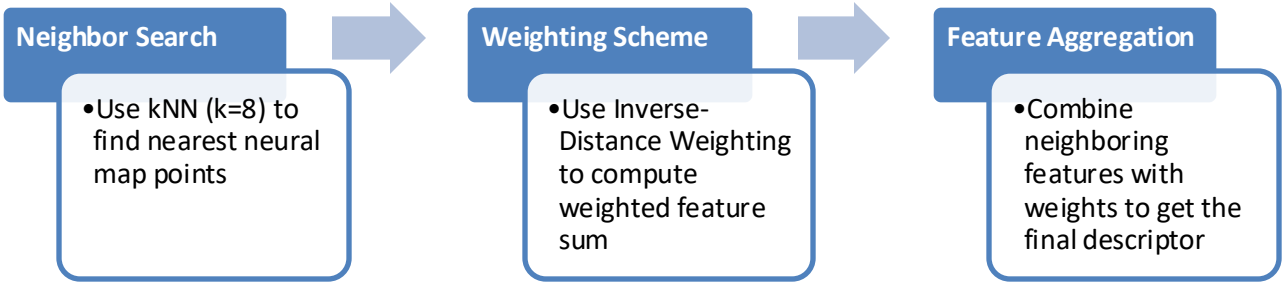
Methodology-Feature Aggregation

❖ Neighbor Search

- Finds the k closest points (k=8) to a given 3D query location in the global map.

❖ Inverse-Distance Weighting

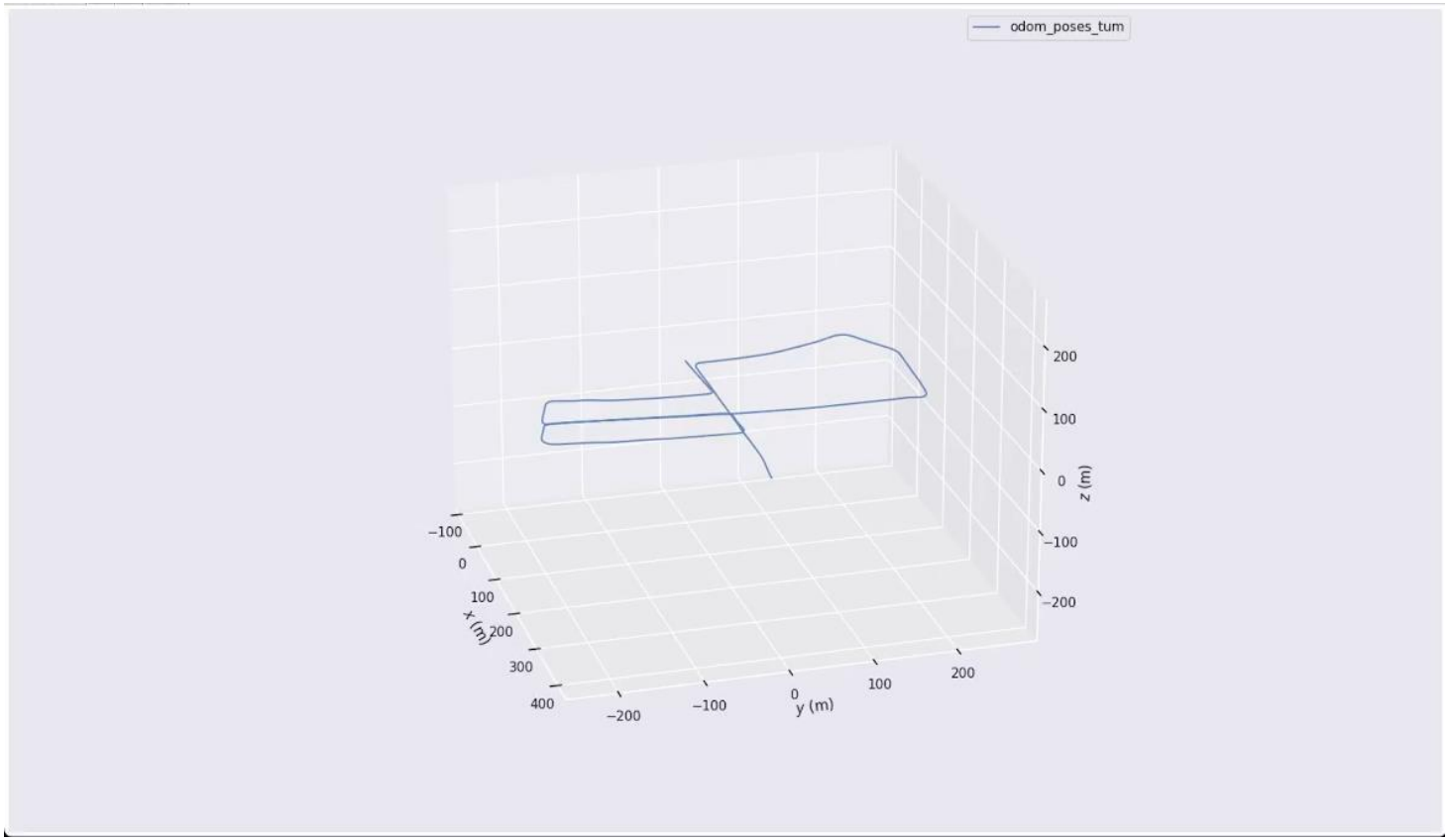
- Once neighbors are found, each neighbor contributes a weighted amount to the aggregated feature, with closer points contributing more.
- The weights are inversely proportional to the distance: $w_i = \frac{1}{\|x - x_i\| + \epsilon}$
- then normalized : $\widehat{w_i} = \frac{w_i}{\sum_j w_j}$



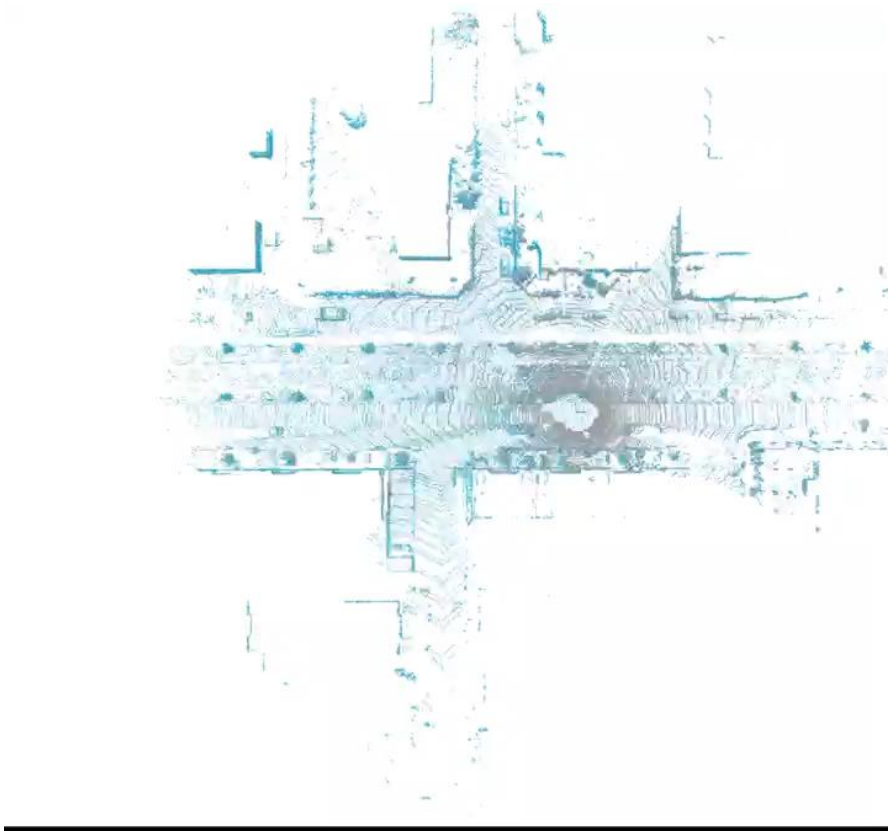
Methodology-Loop closure Detection

❖ **Integrated with the keyframe management and pose graph optimization system.**

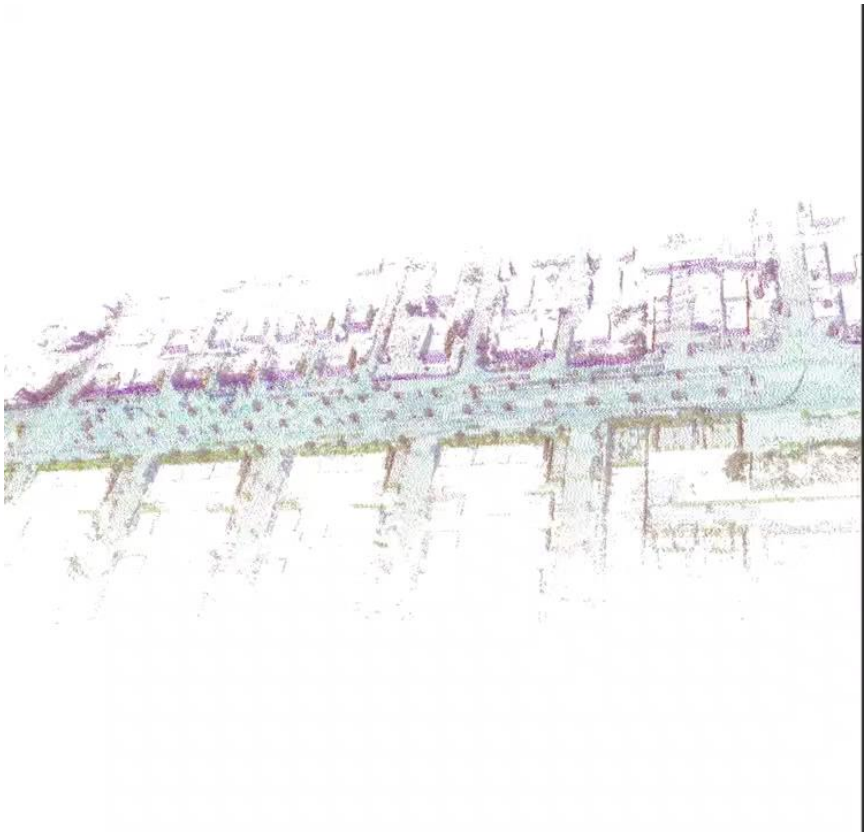
Experiments-Motion Estimation



Experiments-Motion Estimation

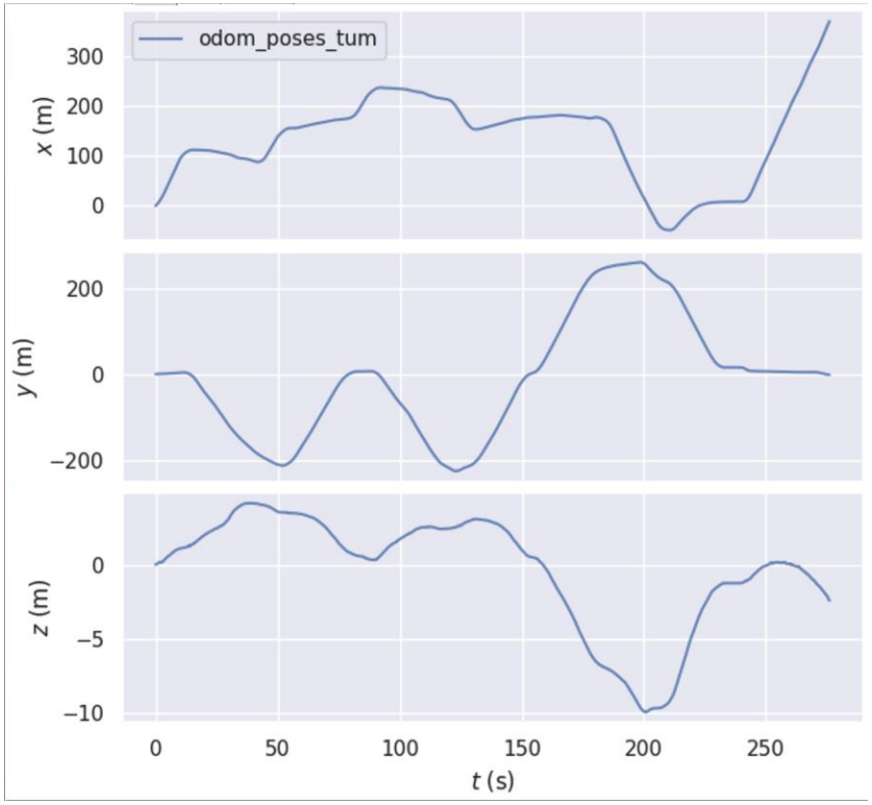


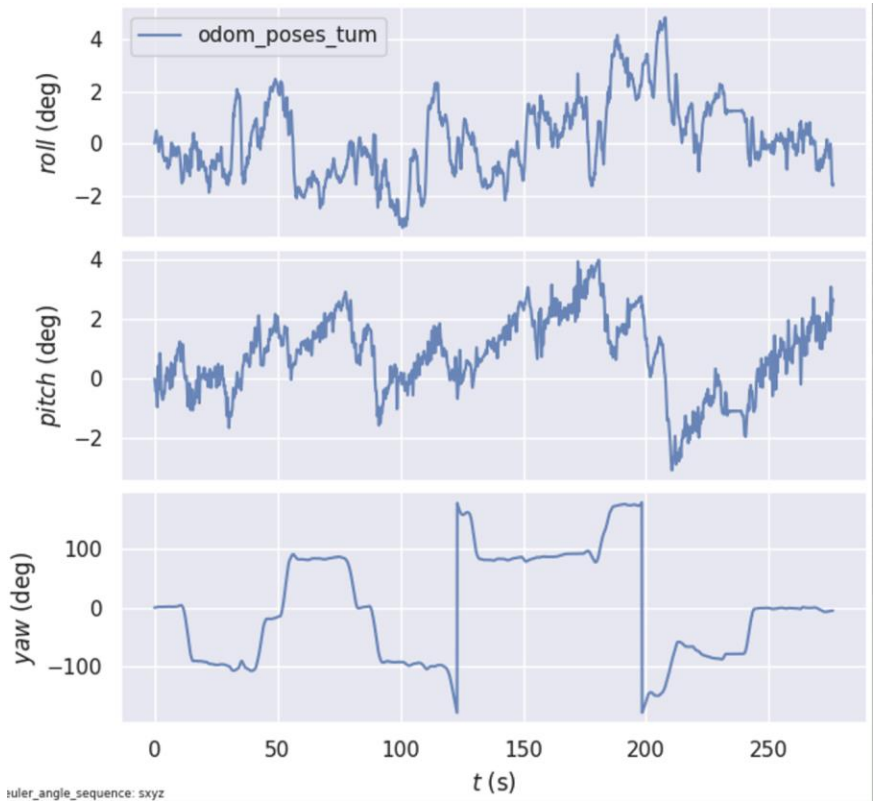
Experiments-Motion Estimation



Experiments-Motion Estimation

Experiments-Motion Estimation





Thank you