Supplementary Materials

Author Names Omitted for Anonymous Review. Paper-ID [add your ID here]

A. Voxel Gaussian and Incremental Update

To manage the computational complexity of a large-scale point cloud map, we divide the local map into voxels of fixed size (e.g., 2.0 meters) and employ incremental updates for voxels affected by the new point cloud frame P. Each voxel is assumed to follow a Gaussian distribution, and the voxels are assumed to be independent of each other. Thus, the map distribution can be approximated as a GMM with each voxel being a Gaussian component.

Let V_i denote the i-th voxel, and $\mathcal{N}_i(\mu_i, \Sigma_i)$ be its corresponding Gaussian distribution. The mean μ_i and covariance Σ_i can be computed as:

$$\mu_i = \frac{1}{|V_i|} \sum_{x \in V_i} x, \tag{1}$$

$$\Sigma_i = \frac{1}{|V_i| - 1} \sum_{x \in V_i} (x - \mu_i)(x - \mu_i)^T, \tag{2}$$

where $|V_i|$ denotes the number of points in the i-th voxel.

When a new point cloud frame P is added to the map, we first transform it into the map coordinate system using the estimated pose T. Then, for each point $p \in P_M$, we compute its corresponding voxel index (i, j, k) using:

$$i = \left\lfloor \frac{p_x}{l} \right\rfloor, \quad j = \left\lfloor \frac{p_y}{l} \right\rfloor, \quad k = \left\lfloor \frac{p_z}{l} \right\rfloor, \tag{3}$$

where l is the voxel size. The Gaussian distribution of the corresponding voxel can be updated incrementally:

$$\mu_i' = \frac{|V_i|\mu_i + \sum_{p \in V_i'} p}{|V_i| + |V_i'|}, \tag{4}$$

$$\Sigma_i' = \frac{|V_i|\Sigma_i + \sum_{p \in V_i'} (p - \mu_i')(p - \mu_i')^T}{|V_i| + |V_i'|}. \tag{5}$$

where V_i' denotes the set of new points falling into the i-th voxel. To efficiently retrieve the voxels, we use a hash table with the voxel index (i, j, k) as the key and the voxel information as the value. This allows for fast updates and queries of the voxel-based map distribution.

B. Wasserstein-based Keyframe Selection

Algorithm 1 presents the Wasserstein distance-based keyframe selection process. The algorithm takes a point cloud frame P, its estimated pose T, a predefined Wasserstein distance threshold τ, and the voxel size l as inputs. It first initializes a voxel map M_1 and computes the Gaussian parameters and point count for each voxel. When a new frame arrives, it is transformed into the map coordinate system and used to update the voxel map incrementally. The average Wasserstein distance between the corresponding voxels in the previous and updated maps is computed using Equation **??**. If the average distance exceeds the threshold τ, the frame is marked as a keyframe. The algorithm continues to process incoming point cloud frames in real-time, effectively identifying informative keyframes for large-scale LiDAR mapping.

Algorithm 1 Wasserstein-based Keyframe Selection

Require: Point cloud frame P, pose T, Wasserstein distance threshold τ, voxel size l
Ensure: Keyframe decision $b \in 0, 1$ for frame P

```
Initialize voxel map M_1
for each voxel V_i in M_1 do
    Compute and store μ_i, Σ_i, and points count n_i using (1), (2), and (3)
end for
while frame P is available do
    Transform P into the map coordinate P_M
    Initialize voxel map M_2
    for each point p_M ∈ P_M do
        Incremental update M_2 using (3), (4), and (5)
    end for
    D_W ← ComputeWassersteinDistance(M_1, M_2)
    if D_W > τ then
        b ← 1                          ▷ Mark frame P as a keyframe
    else
        b ← 0                          ▷ Mark frame P as a non-keyframe
    end if
    Remove outside voxels and update map M_1 and M_2
end while
```

C. Sensor Setups

Figure 1 (c) showcases our handheld sensor acquisition platform. The sensor suite also incorporates an SBG INS with a 100 Hz IMU and a 3DM-GQ7 RTK-INS with a 700 Hz IMU. For localization evaluation, we utilize a 30 Hz fused 6-DoF trajectory by 3DM-GQ7. Additionally, we use ground truth point cloud maps scanned by a Leica RTC360 from the FusionPortableV2 dataset, as shown in Figure 1 (b). The accuracy of the entire map is below 1.5 cm for most areas. Figure 1 (a) displays the five collected datasets projected onto Google Map, with the magenta `CP0` dataset path covering a distance exceeding 10.8 km, collected using a mini vehicle. The `CP1`, `CP2`, `RB2`, and `PK01` sequences are collected using a self-balancing vehicle or handheld setup, as shown in Figure 1 (d). On the software side, we develop and implement the entire multi-session mapping system using C++, GTSAM[1], and Open3D[3]. The experiments are conducted on a personal

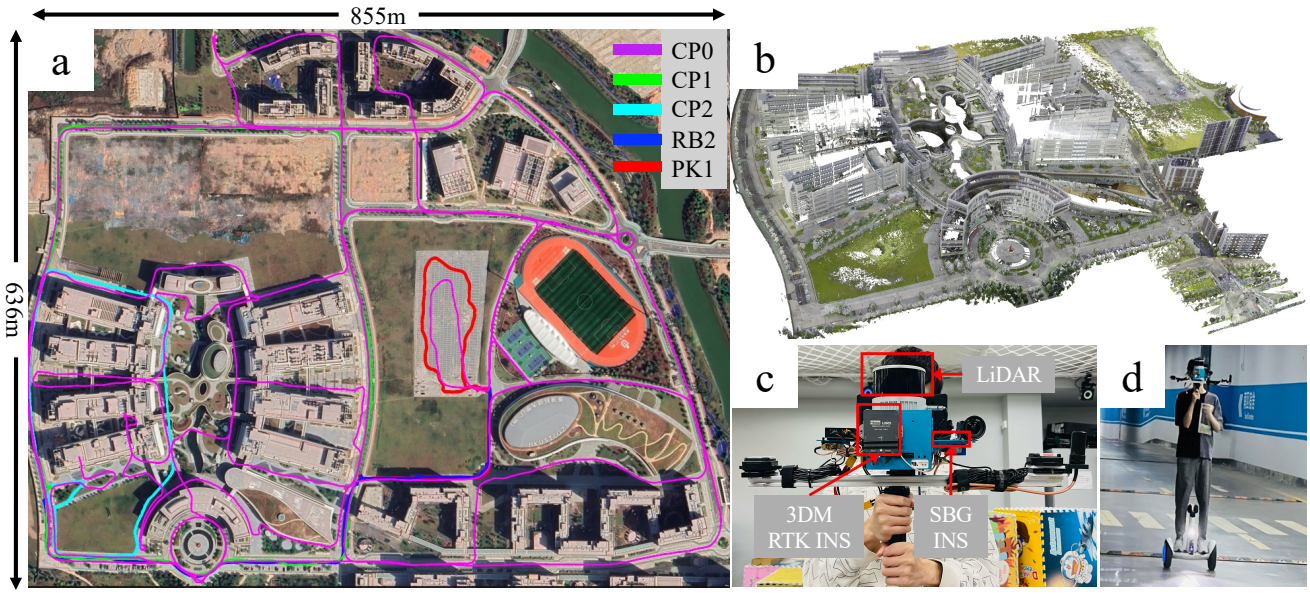

Fig. 1: **Experimental setup and data collection**. (a) Trajectories of the datasets projected onto Google Maps, covering the majority of the campus area. (b) Ground truth RGB point cloud map of the campus used for the experiments[2]. (c) Handheld multi-sensor platform used for data acquisition. (d) Self-balancing vehicle employed for data collection.

desktop computer equipped with an Intel Core i7-12700K CPU, 96 GB of RAM, and a 2 TB hard drive.

References

[1] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. *Georgia Institute of Technology, Tech. Rep*, 2:4, 2012.

[2] Hexiang Wei et al. Fusionportablev2: A unified multi-sensor dataset for generalized slam across diverse platforms and scalable environments, 2024.

[3] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018.