A mapping method integrating enhanced multi-line laser radar and IMU

Caixiang Zhang
School of Computer Science & Engineering
Xi'an Technological University
Xi'an, China
E-mail: zhangcaixiang@st.xatu.edu.cn

Liming Yao
School of Computer Science & Engineering
Xi'an Technological University
Xi'an, China
E-mail: yaolliming@st.xatu.edu.cn

Xiaodong Wu
School of Computer Science & Engineering
Xi'an Technological University
Xi'an, China
E-mail: 2276047884@qq.com

Shuping Xu
School of Computer Science & Engineering
Xi'an Technological University
Xi'an, China
E-mail: 563937848@qq.com

***Abstract*—Most traditional SLAM algorithms suffer severe performance degradation in complex outdoor scenes with pedestrians and moving vehicles, leading to blurred and deformed environmental maps. This paper proposes FE-LIM, a mapping algorithm fusing multi-line LiDAR and IMU. The algorithm judges dynamic points via inter-frame point offset differential analysis, clusters dynamic outliers by DBSCAN and removes them. LiDAR and IMU timestamps are strictly aligned, and EKF is used for sensor fusion to reduce cumulative drift. An adaptive parameter adjustment mechanism is introduced for point cloud feature extraction according to the previous frame information. Tested on three groups of real urban scenes from the KITTI dataset, the algorithm thoroughly filters out dynamic point clutter, the reconstructed trajectory presents small three-axis and angular offset, and the generated 3D point cloud clearly restores complete outlines of roads, buildings and parking lots. Experimental results verify that the proposed method can effectively eliminate dynamic points and produce high-quality static environment maps.**

Keywords-Simultaneous Positioning and Mapping; Complex Outdoor Environment; Differential Analysis; Dynamic Point Cloud Removal

I. INTRODUCTION

SLAM, or simultaneous localization and mapping, first popped up back in 1988 thanks to Smith and his team. Basically, it's a way for robots to figure out where they are and what's around them by using sensors like cameras, lidar, IMUs, or even simple odometers. The robot takes in all this data, then builds a map and keeps track of its own position on the fly, often using filtering or data fusion tricks. People usually break SLAM into two main types: laser SLAM and visual SLAM. And if you zoom in on laser SLAM, you'll see there's 2D and 3D. For 2D laser SLAM, you're mostly looking at single-line lidar—good enough for mapping smaller, indoor spaces because there's only so much data you can grab at once. But when you want to tackle bigger or more complicated places, like city streets, you need 3D laser SLAM, which relies on multi-line lidar. Those sensors pick up way more detail and handle the chaos of the real world much better.

Over the years, plenty of 3D lidar mapping algorithms have come along. Take LOAM [7], for example—it started out as a way to map big urban spaces. It works by estimating movement in real time and uses IMU data to fix up distorted point cloud scans, so you get a clearer picture of the environment. Later on, researchers at Hong Kong University of Science and Technology tweaked A-LOAM. They brought in the ceres library to overhaul LOAM's optimization, which helped speed up map building by cutting down on heavy calculations. Then there's LeGo-LOAM [8], which added a ground optimization feature on top of the original LOAM. This change helps deal with bumpy or uneven ground, making maps more reliable. LIO-SAM [9] goes even further by mixing LOAM's laser feature extraction with VINS-

Mono's [10] IMU pre-integration and factor graph optimization. Thanks to that combo, it delivers much better positioning accuracy and can handle dynamic environments without breaking a sweat.

Most SLAM algorithms work fine when everything around them stays still. But throw in moving objects, and things get messy—detection gets worse, the 3D point cloud starts to warp, and the path can drift off track. All that really messes with mapping and positioning. That's why new mapping methods for dynamic environments have popped up. Take LIOM [11], for example. It builds on LOAM and LINS, but adds some smart tricks—like tightly coupling with the IMU, using CNNs to filter out moving objects, and running an error state Kalman filter (ESKF). All this boosts accuracy and makes mapping a lot more reliable, even when things are in motion. Still, if you want it to work over the long haul, you need closed-loop detection to keep errors from piling up. There's also Deepvcp [12], created by Baidu's self-driving team. They rebuilt the point cloud registration process using deep learning, and it really paid off—accuracy and robustness in dynamic scenes jumped up a notch.

Some algorithms just can't handle dynamic point clouds, and people usually turn to deep learning to deal with them. But this paper takes a different route. It introduces Fusion-Enhanced Lidar-Inertial Mapping (FE-LIM), which combines enhanced multi-line lidar with an IMU. No deep learning involved. Unlike traditional 3D laser SLAM, FE-LIM uses differential detection to spot moving objects in street scenes—perfect for robots on the go. Once it finds those dynamic objects, it removes them. The method also adds a better time synchronization and error correction module, so it tackles sensor drift and boosts data fusion accuracy. As for the point cloud data picked up by the lidar, the algorithm uses adaptive feature extraction. It screens and collects information to smooth out data density and quality, and at the same time, makes feature extraction more robust.

II. Overall Structure of FE-LIM Algorithm

Here's how the FE-LIM algorithm works, step by step. You start with raw data from multi-line lidar and IMU sensors. Before doing anything else, you clean it up thoroughly — get rid of noise and make sure the data's solid and reliable. After that, instead of diving into complicated deep learning models, the algorithm uses a dynamic point cloud removal method that keeps things simple and efficient. It relies on differential detection, setting a reasonable motion threshold for each point in the cloud. If a point moves more than this threshold between two consecutive frames, it's marked as dynamic.

But it doesn't stop there. The algorithm takes these scattered dynamic points and uses the DBSCAN algorithm to group them into complete dynamic object blocks. This way, it can fully remove any moving objects that might mess up the mapping and affect localization accuracy. Next, the lidar and IMU data need to be strictly in sync, so the algorithm aligns their timestamps precisely to make sure everything lines up correctly in time.

Once the data is synchronized, the algorithm uses the extended Kalman filter (EKF) to fuse complementary information from both sensors. This helps correct various errors that pop up during measurement. After filtering, it extracts clear edge and plane features from the point cloud data to build a complete and refined local point cloud for each frame.

Feature extraction gets another boost. The algorithm introduces adaptive adjustment variables, which let it flexibly tweak the extraction strategy based on how the previous frame's point cloud looked. This makes the process more stable and reliable, even if the environment is constantly changing. In the end, all these steps work together to sharpen the accuracy and boost the robustness of the final 3D map, making the system handle complex, dynamic outdoor environments much better.

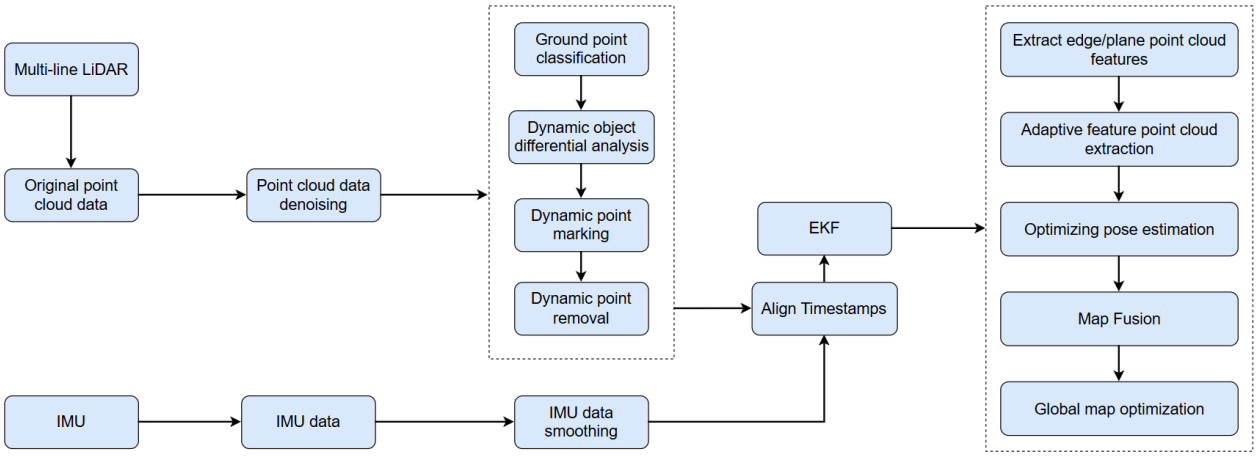


Figure 1. FE-LIM algorithm structure diagram

III. Dynamic Point Cloud Elimination

A. Data preprocessing

Initially, the multi-line laser radar's collected point cloud data undergoes denoising. A radius filter is employed, measuring the distance from each point to its neighbors. It then tallies the number of points within a preset distance threshold. If a point has enough close neighbors, surpassing the denoising threshold, it's kept, otherwise, it's discarded as an isolated point. This process yields a refined point cloud set.

$$P_{\text{filtered}} = \{p \in P | \ \text{count}(q \in P, \| p - q \| < r) > N\} \quad (1)$$

Where p and q are points in the point cloud, P is the original point cloud set, r is the specified denoising radius, and N is the set denoising threshold.

Next, the IMU-gathered linear and angular velocities undergo smoothing. By applying low-pass filtering to data from both previous and current moments, refined, smoothed velocity readings are obtained.

$$a_{\text{smooth}} = \alpha \cdot a_{\text{current}} + (1-\alpha) \cdot a_{\text{previous}} \quad (2)$$

$$\omega_{\text{smooth}} = \alpha \cdot \omega_{\text{current}} + (1-\alpha) \cdot \omega_{\text{previous}} \quad (3)$$

Where a_{smooth} is the smoothed acceleration, ω_{smooth} is the smoothed angular velocity, $a_{\text{current}}, \omega_{\text{current}}$ is the acceleration and angular velocity at the current moment, $a_{\text{previous}}, \omega_{\text{previous}}$ is the acceleration and angular velocity at the previous moment, and α is the smoothing coefficient.

Smoothing cuts down random data fluctuations and noise, boosting stability and reliability, and paving the way for attitude estimation and sensor fusion.

B. Dynamic point marking

Detect points with shifting positions in consecutive LiDAR scans, flagging them as potential dynamic objects. Here's how dynamic points are determined.

Step 1. Collect and process LiDAR and IMU data.

First, collect the data processed by the laser radar to obtain a set of specific data:

$$P(t) = \{p_1(t), p_2(t), \ldots, p_n(t) \quad (4)$$

Where $p_i(t) = (x_i, y_i, z_i, t)$ is the coordinates and timestamp of the i-th point at time t. n is the total number of points collected in this scan.

Next, gather IMU data, derive its speed and position, and bundle smoothed acceleration, angular velocity, etc. recording them as specified.

$$A(t) = (a_x, a_y, a_z, t) \quad (5)$$

$$\Omega(t) = (\omega_x, \omega_y, \omega_z, t) \quad (6)$$

$$Q(t) = \left(q_w,\ q_x,\ q_y,\ q_z,\ t\right) \quad (7)$$

Among them, $a_x,\ a_y,\ a_z$ are the acceleration along the three axes of the IMU device, $\omega_x, \omega_y, \omega_z$ are the angular velocity around each axis, $q_w,\ q_x,\ q_y,\ q_z$ are the direction represented by the quaternion, and t is the timestamp when these data are collected.

Step 2. Constructing a time series.

LiDAR data comprises 3D spatial point coordinates. Observe and track a dynamic object's position over time to form a position time series. (x, y, z)

$$P_{\text{series}} = \left\{\left(x(t_1), y(t_1), z(t_1)\right), \ldots, \left(x(t_n), y(t_n), z(t_n)\right)\right\} \quad (8)$$

The IMU data contains accelerations $a_x,\ a_y,\ a_z$ and angular velocities $\omega_x, \omega_y, \omega_z$. Similarly, the time series D of acceleration and angular velocity in the IMU data is constructed.

Acceleration time series:

$$A_{\text{series}} = \left\{\left(a_x(t_1), a_y(t_1), a_z(t_1)\right), \ldots, \left(a_x(t_n), a_y(t_n), a_z(t_n)\right)\right\} \quad (9)$$

Angular velocity time series:

$$\Omega_{\text{series}} = \left\{\left(\omega_x(t_1), \omega_y(t_1), \omega_z(t_1)\right), \ldots, \left(\omega_x(t_n), \omega_y(t_n), \omega_z(t_n)\right)\right\} \quad (10)$$

Step 3. Dynamic object differential analysis.

For the point cloud data, calculate the changes in its time series and compare the position changes of corresponding points in two consecutive frames P_{t-1} and P_t. Given a point p_i in P_{t-1}, find the corresponding point p'_j in P_t. Calculate the Euclidean distance between two points:

$$d(p_i, p'_j) = \sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2 + (z_i - z'_j)^2} \quad (11)$$

Where $d(p_i, p'_j)$ is the distance between points p_i and p'_j, that is, the distance the same point cloud moves between two consecutive frames.

For the IMU data, the difference between consecutive data points in the time series is calculated to identify changes in velocity and acceleration. For each dimension of the position sequence (x, y, z), the difference is expressed as:

$$\Delta x_t = x_t - x_{t-1}, \Delta y_t = y_t - y_{t-1}, \Delta z_t = z_t - z_{t-1} \quad (12)$$

Further calculation of the difference in velocity and acceleration:

$$v_t = \sqrt{\Delta x_t^2 + \Delta y_t^2 + \Delta z_t^2} / \Delta t \quad (13)$$

$$a_t = (v_t - v_{t-1}) / \Delta t \quad (14)$$

Step 4. Threshold judgment.

For the point cloud data judgment threshold, if the distance $d(p_i, p'_j)$ exceeds the predetermined threshold d_{th}, point p_i is considered to be in dynamic change and is marked.

The IMU data is divided into acceleration threshold a_{thresh} and angular velocity threshold ω_{thresh}.

At a certain moment t_i, the acceleration $a(t_i)$ exceeds this threshold, that is:

$$\| a(t_i)\| = \sqrt{a_x(t_i)^2 + a_y(t_i)^2 + a_z(t_i)^2} > a_{\text{thresh}} \quad (15)$$

This indicates that a significant acceleration change occurred at that time.

At a certain moment t_i, the angular velocity $\omega(t_i)$ exceeds this threshold, that is:

$$\| \omega(t_i)\| = \sqrt{\omega_x(t_i)^2 + \omega_y(t_i)^2 + \omega_z(t_i)^2} > \omega_{\text{thresh}} \quad (16)$$

The marker then undergoes a significant angular velocity change at that time.

C. Dynamic point cluster removal

The DBSCAN algorithm clusters marked point cloud data post-differential planning, eliminating identified dynamic objects. The process is detailed in Fig.2.

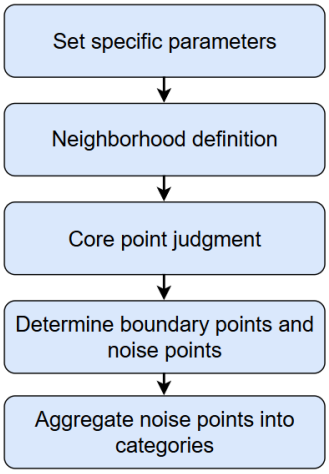


Figure 2. Dynamic point cluster removal process

Step 1. Set specific parameters.

Set r as the neighborhood radius for identification and set $MinPts$ as the minimum number of points required to form a cluster area.

Step 2. Field definition.

For each identified out-of-domain point cloud p, find all points whose distance to p is less than or equal to r, and construct these special points as the r-neighborhood of p. It can be expressed as:

$$N_r(p) = \{q \in D | \ dist(p,q) \le r\} \quad (17)$$

Where D is the point cloud data range, and $dist(p,q)$ is the distance between point p and point q.

$$dist(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2} \quad (18)$$

Step 3. Judgment core points.

If there are at least $MinPts$ points in a point cloud area r, including the point itself, it is judged as a core point, that is:

$$| N_r(p) | \ge MinPts \quad (19)$$

Step 4. Determine boundary points and noise points

If point p is not a core point but is within the r-neighborhood of a core point, then p is considered a boundary point. If point p is neither a core point nor a boundary point, then p is classified as a noise point.

Step 5. Aggregate all noise points into categories.

For unassigned core or non-isolated boundary points, recursively add all dense points in their r-neighborhood to the same cluster. These clusters are then identified as dynamic objects.

Step 6. Remove the point clouds of these dynamic objects from the current frame.

Clustering is used from the current point cloud data to remove the marked point cloud and obtain the processed point cloud data.

IV. EKF Filters the Aligned Sensor Information

We synchronize LiDAR and IMU timestamps, employing interpolation to match specific scan times. The Extended Kalman Filter (EKF) then refines this aligned sensor data, minimizing errors from noise or transmission delays.

Fig3 illustrates using interpolation to align LiDAR and IMU data, followed by EKF filtering to refine sensor information.

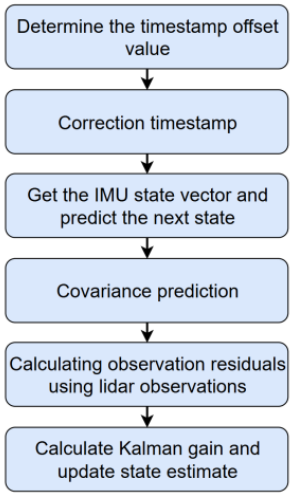


Figure 3. EKF fusion correction of IMU and odometer error process

The specific implementation is as follows:

Step 1. Timestamp offset value determination.

Gather timestamp data from incoming LiDAR and IMU streams. If timestamps are unsynchronized, compute the average time offset

by comparing timestamps recorded by both systems for the same event.

$$\Delta t_{avg} = \frac{1}{N}\sum_{i=1}^{N}(t_{L,i} - t_{I,i}) \tag{20}$$

Where $t_{L,i}$ and $t_{I,i}$ represent the timestamps recorded by the lidar and IMU, and Δt_{avg} is estimated by taking the average of the time deviations of all events.

Step 2. Timestamp correction.

For ongoing data streams like LiDAR and IMU, timestamp deviations vary. To maintain temporal consistency, timestamps are adjusted using interpolation.

$$D_t = D_{t_1} + \frac{(t - t_1)}{(t_2 - t_1)} \cdot (D_{t_2} - D_{t_1}) \tag{21}$$

Where t_1 and t_2 are the time data between the timestamps of the deviation, and the data values corresponding to these two points are D_{t_1} and D_{t_2}. The data value D_t at time t is estimated by linear interpolation.

Step 3. Get the state vector of the IMU.

$$x_k = [p_k, v_k, q_k]^T \tag{22}$$

Where x_k is the state vector, including position p_k, velocity v_k, and posture q_k.

Get its state vector through acceleration and angular velocity:

$$x_k = f\left(x_{k-1},\ u_{k-1}\right) + w_{k-1} \tag{23}$$

Among them, u_{k-1} is the control input, the acceleration a_{k-1} and angular velocity ω_{k-1} measured by the IMU, and w_{k-1} is the process noise.

Step 4. Predict the next state of the IMU.

The acceleration and angular velocity of the IMU are used to predict the state at the next moment. Assume that the acceleration a_(k-1) directly affects the velocity and the angular velocity ω_(k-1) affects the posture q_k.

$$p_k = p_{k-1} + v_{k-1}\Delta t + \frac{1}{2}a_{k-1}\Delta t^2 \tag{24}$$

$$v_k = v_{k-1} + a_{k-1}\Delta t \tag{25}$$

$$q_k = q_{k-1} \otimes Exp\left(\omega_{k-1}\Delta t\right) \tag{26}$$

Where Δt is the time interval, $Exp()$ is the operation of converting angular velocity into quaternion change, and $\otimes$ represents quaternion multiplication.

Step 5. Covariance prediction.

$$P_k = F_{k-1}P_{k-1}F_{k-1}^T + Q_{k-1} \tag{27}$$

Where F_{k-1} is the Jacobian of the state transfer matrix and Q_{k-1} is the noise covariance matrix of the IMU.

Step 6. Calculating observation residuals using lidar observations.

$$y_k = z_k - h\left(x_{k|k-1}\right) \tag{28}$$

where z_k is the position observation provided by the lidar and $h\left(x_{k|k-1}\right)$ converts the predicted state into the observation space.

Step 7. Calculate Kalman gain and update state estimate.

$$K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1} \tag{29}$$

H_k is the Jacobian matrix of the observation model, and R_k is the observation noise covariance matrix.

Update the state estimate:

$$x_{k|k} = x_{k|k-1} + K_k y_k \tag{30}$$

Update the covariance gain:

$$P_{k|k} = \left(I - K_kH_k\right)P_{k|k-1} \tag{31}$$

V. MAP BUILDING

A. Adaptive feature extraction

Feature extraction employs an adaptive approach. Local curvature calculates edge eigenvalues from refined point cloud data. The RANSAC algorithm extracts plane features, with parameters adaptively optimized for feature framework extraction.

The specific method for adaptive extraction of point cloud features is as follows:

Step 1. Calculate point cloud density.

The local density of each point cloud is calculated using the K nearest neighbor (KNN) algorithm:

$$\rho(p) = \frac{1}{|N_k(p)|} \sum_{q \in N_k(p)} \| p - q \| \tag{32}$$

Where $N_k(p)$ is the set of k nearest neighbors of point p, and $\| p - q \|$ is the distance between point p and point q.

Step 2. Feature Threshold Adjustment.

$$k_{th} = f(\rho) \tag{33}$$

Among them, $f(\rho)$ is a function that adjusts the curvature threshold k_{th} according to the point cloud density ρ.

Step 3. Feature extraction.

Edge points are pinpointed by computing the point cloud's local curvature, gauged by alterations in the normal vector of the least-squares fitted plane.

$$\kappa(p) = \frac{1}{\| N(p) \|} \sum_{q \in N(p)} \| n_p - n_q \| \tag{34}$$

Where $N(p)$ is the neighborhood of point p, n_p and n_q are the normal vectors of the fitted plane at point p and its neighborhood point q, respectively.

The features of the plane are extracted through the RANSAC algorithm, and the point clouds are randomly assigned into groups of three to construct the plane equation:

$$Ax + By + Cz + D = 0 \tag{35}$$

Among them, A, B, C are the normal vectors of the plane, and D is the displacement term.

Then, for each point in the point cloud, calculate its distance to the plane model and find the inner points in the plane point cloud:

$$d = \frac{| Ax + By + Cz + D |}{\sqrt{A^2 + B^2 + C^2}} \tag{36}$$

Step 4. Add adaptive distance tolerance function.

When you're deciding if a feature point counts as an inner point, you need to look at how far it is from the plane and compare that distance to a set threshold. The problem is, if you just pick one fixed threshold, things can go sideways—especially when the environment changes or you have a lot more (or fewer) feature points than before. So, to really fine-tune the optimization of these feature points as conditions shift, we came up with an adaptive threshold adjustment function. This function watches the relative error between the inner point matches you get with your current threshold and the target matches you want. Then, it automatically tweaks the distance tolerance for the next round, keeping the optimization on track, even as things change.

$$C_{current} = \frac{Num^{in}}{Num^{all}} \tag{37}$$

Among them, Num^{in} represents the number of inliers judged in the current iteration, Num^{all} represents the total number of feature points in the current iteration, and $C_{current}$ represents the correspondence degree of the current feature extraction, that is, the ratio of the number of inliers currently extracted to the total number of points.

$$Param_{new} = Param_{old} * \left(1 + \alpha * \frac{C_{target} - C_{current}}{C_{target}} \right) \tag{38}$$

Among them, C_{target} represents the target correspondence. $Param_{old}$ is the distance tolerance of the current iteration, $Param_{new}$ is the distance tolerance of the next iteration after adaptive update, and α is a proportional adjustment coefficient, which is used to control the amplitude of the distance tolerance adjustment, make adaptive judgments on the feature extraction process, and dynamically adjust the point cloud feature extraction parameters at the next moment according to the current state to ensure the accuracy and integrity of the 3D point cloud information. Through this dynamic adjustment mechanism, useful information can be extracted from complex urban street environments more effectively, improving the accuracy of map construction and navigation.

B. Pose optimization and map optimization

The multi-line lidar and IMU data are fused, the fused pose is estimated, and the pose is optimized using the nonlinear optimization technology Levenberg-Marquardt algorithm.

Step 1. The multi-line lidar and IMU data are fused and the IMU data is used for initial pose estimation.

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \omega \Delta t \tag{39}$$

Step 2. The pose is optimized using the Levenberg-Marquardt algorithm.

$$\hat{\theta}^* = \arg\min_{\theta} \sum_i w_i \| f_i(\theta) - \mathbf{y}_i \|^2 \tag{40}$$

Among them, $f_i(\theta)$ is the prediction model, y_i is the observation data, and w_i is the weight.

Block point cloud maps are merged via global map fusion to derive rotation and translation matrices. The map is then refined to create a comprehensive 3D point cloud map.

$$M_{global} = M_{global} \bigcup \{ R(\hat{\theta}^*) p + T(\hat{\theta}^*) \mid p \in P_{current} \} \tag{41}$$

Where M_{global} is the global map, $P_{current}$ is the point cloud information of the current lidar frame, $R(\hat{\theta}^*)$ and $T(\hat{\theta}^*)$ are the rotation and translation matrices obtained based on pose estimation, respectively.

Finally, the global map is optimized regularly, redundant data is removed, and similar feature points are merged to improve the accuracy of the map and reduce storage requirements.

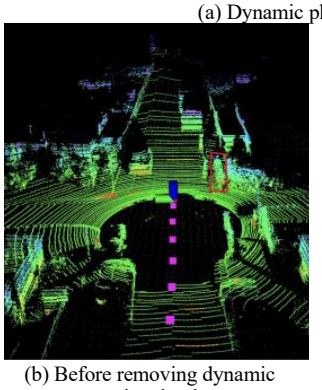
VI. Experimental Verification

A. Dynamic point cloud removal test

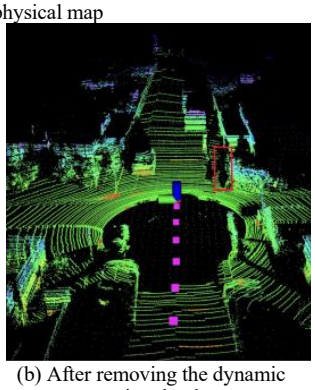
Eliminating dynamic point clouds in multi-line LiDAR mapping enhances scene positioning and improves mapping accuracy. Three real-world tests using the KITTI dataset demonstrate dynamic point cloud removal.

Fig4 illustrates that after processing, dynamic interfering point clouds are effectively removed, revealing a clearer static background.Fig.5 confirms the method's accuracy in distinguishing dynamic objects from static scenes.Fig.6 demonstrates that, while eliminating dynamic points, the method preserves the stability and integrity of the static point cloud, ensuring reliable mapping results.

(a) Dynamic physical map

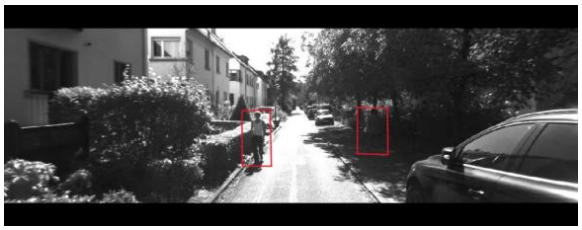


(b) Before removing dynamic point cloud

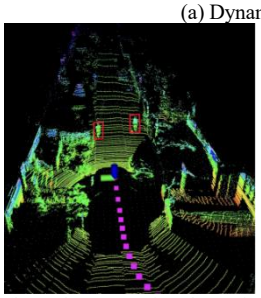


(b) After removing the dynamic point cloud

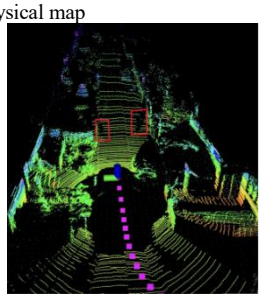
Figure 4. The first group of dynamic point cloud removal effect comparison chart



(a) Dynamic physical map



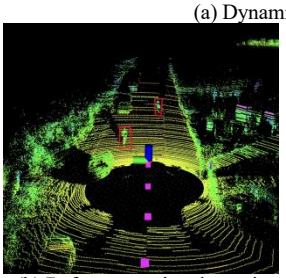
(b) Before removing dynamic point cloud



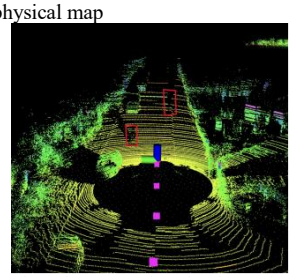
(b) After removing the dynamic point cloud

Figure 5. The second group of dynamic point cloud removal effect comparison chart

(a) Dynamic physical map



(b) Before removing dynamic point cloud



(b) After removing the dynamic point cloud

Figure 6. The third group of dynamic point cloud removal effect comparison chart

B. Complete map test

For the purpose of verifying the system's mapping performance in actual environments, the publicly available KITTI dataset is adopted as the test benchmark. The KITTI dataset contains high-quality information collected simultaneously by multiple sensors, such as lidar, IMU, GPS, and binocular cameras, and is suitable for positioning and map construction research in the field of autonomous driving.

During the test, typical urban roads and parking areas were selected as mapping scenes. The system collected environmental point cloud information through multi-line laser radar and combined it with IMU data for sensor fusion processing. Following the timestamp alignment and Extended Kalman Filter (EKF)-based correction of the sensor data, the system carries out continuous splicing and registration of the point cloud data from each frame, and improves the mapping precision by means of a pose optimization algorithm.

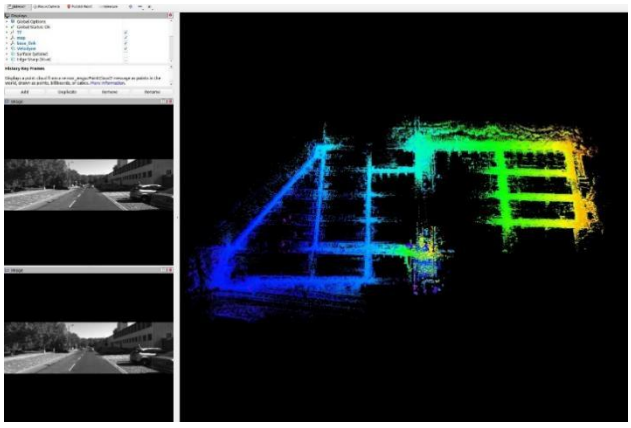
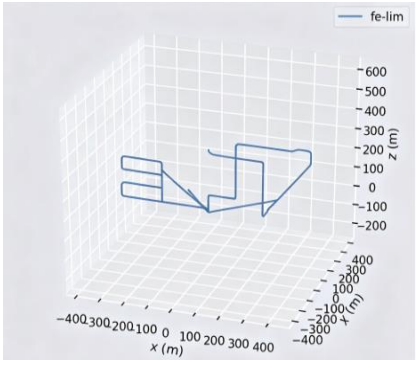
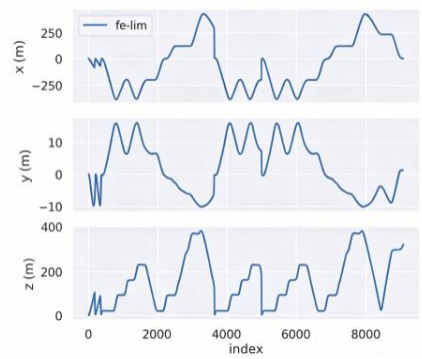


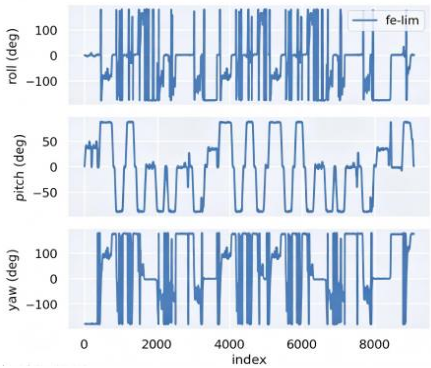
Figure 7. Complete mapping effect of FE-LIM algorithm



(a) Mapping trajectory restoration



(b) Three-axis offset



(c) Angle offset

Figure 8. FE-LIM algorithm mapping related data

Take a look at Fig. 7—you’ll see the 3D point cloud map we built from the test data. The system does a solid job here: you can clearly pick out road boundaries, building outlines, and the layout of the parking areas. Fig. 8 digs in a bit more, showing the mapping trajectory along with the three-axis and angle offset curves. These results really highlight how the FE-LIM algorithm keeps pose estimation steady and cumulative error low, which means mapping stays sharp and accurate. Zoom in on the point cloud and you’ll notice the density stays even, and the outlines don’t break up—they’re smooth and complete. This shows the system really works and holds up when mapping real scenes. The experimental results back this up: with multi-sensor data fusion, motion distortion correction, and dynamic object filtering, the system produces crisp, high-precision 3D maps. It’s reliable, detailed, and definitely ready for real-world use.

VII. Conclusions

This paper introduces an outdoor large-scene SLAM approach that combines multi-line laser radar and IMU for intelligent driving research. It demonstrates excellent adaptability and precise positioning in outdoor settings. By creating high-density 3D point cloud maps with laser radar, the system accurately detects road boundaries, obstacles, and structures. The DBSCAN algorithm clusters and identifies dynamic points, effectively eliminating interference from moving objects or human activity, enhancing map stability and reliability.

Acknowledgment

The authors extend their gratitude to collaborators. This research is partly supported by the Shaanxi Province University Student Innovation and Entrepreneurship Project (S202510702092) and a national-level college student entrepreneurial project (202510702002X).

References

[1] Xu W, Cai Y, He D, et al. FAST-LIO2: Fast direct LiDAR-inertial odometry[J]. IEEE Transactions on Robotics, 2022, 38(4): 2053-2073.

[2] LI Yabin. Research on Indoor SLAM and Path Planning of Omnidirectional Mobile Robot [D]. North University of China, 2023.

[3] Liu Bingqi, Zhang Junshan, Tang Hui, et al. A review of algorithms of laser and visual SLAM [J/OL]. Laser & Optoelectronics Progress, 1-23[2025-07-19].

[4] SHEN Si-jie, TIAN Xin, WEI Guo-liang, et al. Review of SLAM Algorithm Based on 2D Lidar [J]. Computer Technology and Development, 2022, 32 (01): 13-18+46.

[5] TIAN Jianlin, CHEN Xiaobo, ZHANG Wenchang, et al. Research Progress on Autonomous Navigation Technology for Mobile Robot [J]. Techniques of Automation and Applications, 2025,44(07):1-4+55.

[6] LIU Mingzhe, XU Guanghui, TANG Tang, et al. Review of SLAM Based on Lidar [J]. Computer Engineering and Applications, 2024, 60 (01): 1-14.

[7] Cattaneo D, Matteucci M. CLOISTER: LiDAR-inertial odometry with online initialization[J]. IEEE Robotics and Automation Letters, 2022, 7(2): 1772-1779.

[8] Shan T, Englot B. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain[C]//2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018: 4758-4765.

[9] Shan T, Englot B, Meyers D, et al. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping[C]//2020 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2020: 5135-5142.

[10] Lin J, Zhang F. R3LIVE: A robust, real-time, RGB-colored, LiDAR-inertial-visual tightly-coupled state estimation and mapping package[C]//2022 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2022.

[11] Vizzo I, Guadagnino T, Mersch B, et al. KISS-ICP: In defense of point-to-point ICP—simple, accurate, and robust registration if done the right way[J]. IEEE Robotics and Automation Letters, 2023, 8(2): 1029-1036.

[12] Deng H, Birdal T, Ilic S. PointNetLK: Revisiting point cloud registration with pointnet and robust kernel[C]//2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2022: 1-10.

[13] Gong J, Sun Y, Liu X. GriT-DBSCAN: A spatial clustering algorithm for very large databases[J]. Information Systems, 2023, 113: 102163.

[14] Fischler M A, Bolles R C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography[J]. Communications of the ACM, 1981, 24(6): 381-395.

[15] Sun P, Kretzschmar H, Dotiwalla X, et al. Scalability in perception for autonomous driving: Waymo open dataset[C]//2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2020: 2446-2454. (Waymo Open Dataset,2020)

[16] Behley J, Garbade M, Milioto A, et al. SemanticKITTI: A dataset for semantic scene understanding of LiDAR sequences[C]//2019 IEEE/CVF International Conference on Computer Vision (ICCV). IEEE, 2019: 9297-9307.