ITC 2/53

Information Technology and Control
Vol. 53 / No. 2 / 2024
pp. 584-600
DOI 10.5755/j01.itc.53.2.32897

A Construction Optimization for Laser SLAM Based on Odometer Constraint Fusion

Received 2022/12/02 | Accepted after revision 2024/01/01

**HOW TO CITE:** Huang, H., Yang, P., Cai, S., Li, J., Zheng, Y., Zou, T. (2024). A Construction Optimization for Laser SLAM Based on Odometer Constraint Fusion. *Information Technology and Control*, 53(2), 584-600. https://doi.org/10.5755/j01.itc.53.2.32897

# A Construction Optimization for Laser SLAM Based on Odometer Constraint Fusion

**Haojun Huang, Puxian Yang, Shengqing Cai**
College of Mechanical and Electrical Engineering, Fujian Agriculture and Forestry University, Fuzhou, Fujian 350002, China
Key Laboratory of Agricultural Information Sensoring Technology, Fujian Agriculture and Forestry University, Fuzhou, Fujian 350002, China

**Jixiang Li**
Jinshan College, Fujian Agriculture and Forestry, China

**Yuda Zheng, Tengyue Zou**
College of Mechanical and Electrical Engineering, Fujian Agriculture and Forestry University, Fuzhou, Fujian 350002, China
Key Laboratory of Agricultural Information Sensoring Technology, Fujian Agriculture and Forestry University, Fuzhou, Fujian 350002, China

**Corresponding author:** zouty@fafu.edu.cn

The traditional laser SLAM (Simultaneous Localization and Mapping) algorithm uses the global relative poses and local ones to form residual blocks. Its constructed map is not smooth enough and the constraint construction is too simplex under some special scenarios. Thus, this paper proposes an odometer constraint fusion method called FOSLAM (Fusion Odometer SLAM) to construct residual blocks between constrains and solve the nonlinear least squares by Ceres. The effectiveness and accuracy of this method have been verified through comparative experiments. Experimental results showed that without increasing the time and space complexity, by involving the odometer constraint into the SLAM optimization process, the convergence of scan matching scores can be improved and the constructed grid map edges are smoother and the jagged phenomenon can be reduced. Under sophisticated scene, FOSLAM is able to acquire more accurate maps and laser odometer trajectory than Cartographer method. Therefore, it is suitable to be used on indoor robot for cleaning and inspection and can be further deployed on autonomous unmanned vehicles involving spatial visualization and neuro-heuristic guidance.

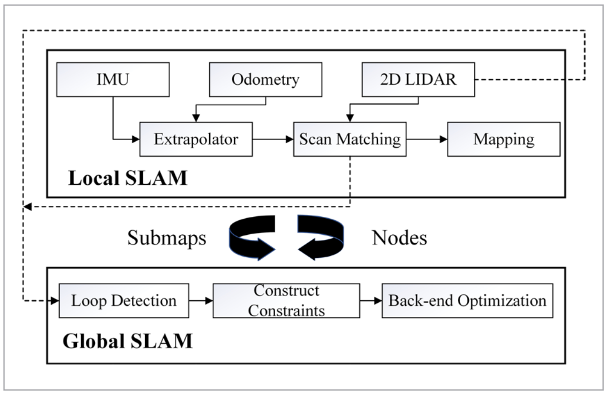
KEYWORDS: Laser SLAM, Redisual blocks, Back-end optimization, Odometer constraint fusion, Ceres.

## 1. Introduction

The SLAM method is mainly used to solve the localization, navigation and map construction problems for mobile robot under unknown environment [1] . Therefore, it is suitable to be used on indoor robot for cleaning and inspection [29] and can be further deployed on autonomous unmanned vehicals involving spatial visualization [4] and neuro-heuristic guidance [24]. In recent years, improvements in SLAM have mainly focused on sensor data fusion and artificial intelligence (AI) heuristic guidance. Kachurka et al. [16] fused indoor wearable GPS information into dataset to improve SLAM performance, and flying lidar data are also taken to rise location accuracy for unmanned aerial robot [25]. Jamaludin et al. [12] used Rao-Blackwellized particle filter (RBPF) integrated with artificial neural networks (ANN) to assist in SLAM modeling, and convolutional neural networks (CNN) [13] can also be used to improve the fault detection of scan matching in SLAM.

Robots with specific sensors are able to acquire their poses by constructing environmental models without any prior conditions, which can greatly improve the autonomous navigation ability and their adaptability to different application environments [30]. The method is also helpful to dynamic path planning, real-time obstacle avoidance and multi-robot cooperation. Modern SLAM framework can be divided into three steps: front-end scan matching, back-end pose graph optimization and loopback detection for positioning. Scan matching builds map and stores pose data, pose graph optimization fine-tunes the map by constructing constraints. In scan matching step, the prior poses are generally provided by IMU (Inertial Measurement Unit) and odometer, then conducting scan matching by combining laser data and prior poses. Finally, output the grid map frame by frame. The pose nodes obtained by scan matching at the front-end are added to the pose graph at the back-end. Simultaneously, all the poses in the graph are globally optimized and the corresponding map points for each pose are corrected accordingly. The framework of 2D SLAM is shown in Figure 1. On Google's Cartographer [11], in order to ensure the convergence of closed-loop constraints and global optimization, loopback detection is usually needed to calculate the constraints of intra- and inter-submap. Finally, the information is added to the optimization problem for nonlinear least squares solution.

**Figure 1**
A framework of laser SLAM



The use of pose graph simplifies the structure of back-end optimization process and it can greatly improve the real-time performance. Therefore, the pose graph optimization (PGO) has occupied a dominant position in the current SLAM researches. The optimization method does not directly solve the cost function but through iteration. During the iterative process, the independent variable is constantly adjusted and the cost function is gradually reduced. When the cost function cannot be decreased further or the change is smaller than the threshold value, the iteration is completed.

With the development of the accuracy requirements of SLAM, researchers have proposed some back-end optimization methods. Konolige et al. [19] introduced a nonlinear optimization system called sparse pose adjustment (SPA) using efficient linear matrix construction and sparse non-iterative Cholesky decomposition to solve large sparse pose graphs. In addition, mainstream SLAM back-end optimization libraries, such as General Graphic Optimization (G2O) [20] and Georgia Tech Smoothing and Mapping (GTSAM) [21] are proposed based on SPA approach. Stochastic gradient descent (GD) [8, 31] is developed to help solving this probelm, but the descent efficiency is unstable and it is easy to fall into local minimum. For high dimensional data, the second derivatives may be hard to be calculated. Gauss-Newton (GN) method [3, 28] is able to improve the efficiency of solving it by fitting Jacobian and Hessian matrixes. However, the results acquired by GN cannot guarantee the gradient descent. Thus, Levenbergt-Marquardt (LM) algorithm [2, 27] is introduced to add a diagonal matrix to the fitting results to ensure the gradient descent. Furthermore, the Dogleg algorithm [10] uses the slope changes in the optimization process to adjust the trust region and guide the LM solution process.

In the optimization strategy of SLAM, the sparsity of the network must be guaranteed to speed up the computation of the optimal solution [9]. Eckenhoff et al. [6] introduced a decoupled, consistent marginalization and sparsification (DMS) approach for reducing the computational cost of graph-based SLAM to enable long-term operation. Recently, Gao et al. [7] have combined Cholesky decomposition with back-end optimization to improve the computational speed of the algorithm. In the simulation, the efficiency of optimized method was improved by 24% compared with the traditional ones.

However, there are common problems in these methods, they only focus on the efficiency of the algorithm but ignore the complex styles and high hardware cost. Furthermore, the selection of back-end constraints is also relatively simple, which does not make full use of front-end data. A suitable loss function is also needed to penalize those terms with too large error [5]. In this paper, a new constraint construction method for SLAM back-end is introduced, named Fusion Odometer SLAM (FOSLAM). The odometer data is fused into the SLAM back-end and the nonlinear least square method is used to optimize the poses. Moreover, FOSLAM is compared with other laser SLAM methods in terms of algorithmic complexity. The possibility of deploying FOSLAM on embedded devices is also verified.

This paper introduces the general laser SLAM framework including the front-end scan matching in Section 2. In Section 3, the new FOSLAM method is introduced which imports the odometer data into the constraint construction at the back-end optimization and uses LMF procedure to guide the optimization iteration process. Finally, in Section 4, experimental results show that FOSLAM can improve the convergence of scan matching without increasing the redundancy and the its mapping results are smoother. Moreover, in high-complexity scenarios, FOSLAM is able to acquire better achievement than Cartographer method.

## 2. Local 2D Laser SLAM

The laser SLAM is generally classified as local SLAM and global SLAM according to the different reference coordinates of robot poses. Local mapping is a process to build a local map using sensor scan data, also called local SLAM. The relationship among robot pose points, observation data and map are established by constraint quantity. There is error between the robot prior pose predicted by IMU and odometer data and the actual one. Thus, it is necessary to update the prior pose by the observation data further [33].

### 2.1. Submap Construction

Laser data, submaps and global map are linked by robot pose. When each frame of laser data is acquired, the laser data are inserted into submap using the scan matching method of Scan-to-Submap. Therefore, a certain amount of laser data forms a submap, all submaps eventually form a global map.

Laser data are usually denoted by vector $h_k$ ($k$=1,2,...). It represents the coordinates of a set of laser points with the LiDAR rotation center as the origin of the coordinate system. In constructing procedure of submap, the transformation matrix $T_\zeta=(R_\zeta,t_\zeta)$ is used to represent the rotation and translation relationship between the two neighbouring frames of laser data. Then, transform the laser data $h_k$ into the submap coordinate system to complete the construction of submap by Equation (1).

$$T_\zeta \cdot h_k = \begin{bmatrix} \cos\zeta_\theta & -\sin\zeta_\theta \\ \sin\zeta_\theta & \cos\zeta_\theta \end{bmatrix} h_k + \begin{bmatrix} \zeta_x \\ \zeta_y \end{bmatrix} \quad (1)$$

where the pose transformation can be expressed as $\zeta$= ($\zeta_x$, $\zeta_y$, $\zeta_\theta$) with a [$\zeta_x$, $\zeta_y$] translation and a $\zeta_\theta$ rotation. In this coordinate transformation formula, the first 2*2 matrix is $R_\zeta$, the second 2*1 matrix is $t_\zeta$.

### 2.2. Smooth Scan Matching

After the result of new scan is inserted into the submap, the state of the 2D grid needs to be updated iteratively. The updating method is as Equations (2)-(3) shown in [11].

$$odds(prob) = \frac{prob}{1-prob} \quad (2)$$

$$M_{\text{new}}(x) = clamp(odds^{-1}(odds(M_{old}(x)) \cdot odds(P_{hit}))), \tag{3}$$

where *prob* is the occupancy probability $P_{hit}$ or non-occupancy probability $P_{miss}$, $x$ represents the grid, and *clamp* denotes the interval limiting function. *odds* is an inverse proportional function that updates the probability of hits and misses, and $odds^{-1}$ is the inverse function of odds. $M_{old}$ is the existing probability grid and $M_{new}$ is the iteratively updated probability grid.

This grid updating mechanism can effectively reduce the interference of dynamic obstacles in the environment. Due to the large error in the predicted pose from the motion model, the Scan-to-Submap matching is used to further update the predicted pose with the observed data. The usual method is to create a nonlinear search window around the prior pose. Search and match within the window to get the closest results to the prior pose, and it is actually solving a nonlinear least square problem as Equation (4) shown.

$$\arg\min \sum_{k=1}^{K} (1 - M_{smooth}(T_{\xi} \cdot h_k))^2 \tag{4}$$

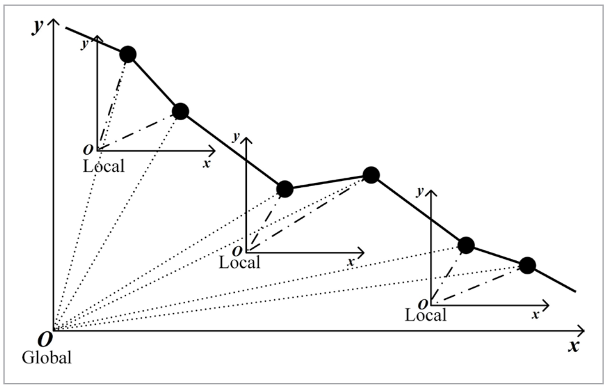
The process of Equation (4) determines the matching degree between laser data and submap. $M_{smooth}$ is a smoothing filter function obtained by bicubic interpolation method. When the difference between interpolated result and the prior pose is smallest, the matching is considered to be successful.

## 3. Pose Graph Optimization

In the graph structure of SLAM back end, each scan frame corresponds to a pair of global coordinates and a pair of local coordinates in global and local submap coordinate system, respectively. The front-end scan matching continuously generates these coordinates and saves them to the back end. The back end of SLAM algorithm combines these pose information into various constraints that actually form the pose graph [32, 14]. When the closed loop is detected, global optimization is carried out on all poses in the whole pose graph. When all poses will be corrected and the corresponding map points on each pose will be corrected accordingly. This is the pose graph optimization process of the global mapping.

The pose graph optimization is based on nodes and edges. A node is a global or local pose of a robot. The edge represents the constraint relationship between two poses. Thus, an edge is a coordinate transformation. In general, there is a set of constraint relations between two global poses. The structure of the pose graph is shown in Figure 2.

**Figure 2**
The structure of pose graph



### 3.1. Nonlinear Least Squares Modeling

In the algorithm, Ceres library is used to model the optimization problem. Ceres solves robust bounds constrained non-linear least squares problem by the formula as Equation (5).

$$\min_{x} \frac{1}{2} \sum_{i} \rho_i(\|f_i(x_{i1}, ..., x_{ik})\|^2) \tag{5}$$

where $\rho_i$ is a loss function. Loss function is a function with scalar value which reduces the impact of outliers on the solution of nonlinear least square problem [26]. $f_i$ represents a cost function that depends on the parameter block. $x_{i1}$, ...., $x_{ik}$ are optimization variables and *min* denotes the function to get the minimum value.

### 3.2. Construction of Residual Terms

In general, the construction of residual terms needs two kinds of constraints. Under global coordinates, the coordinate transformation between two nodes forms the first constraints. The traditional back-end optimization methods extract the transformation between the corresponding nodes from the submap to form the second constraints. However, these methods all ignore the useful data collected by the front-end sensors. Thus, we designed a Fusion Odometer SLAM (FOSLAM) method to adopt relative pose data obtained from odometer to constitute another set of constraints. FOSLAM uses the global relative pose and odometer relative pose to make a set of residual terms and constructing the cost function. Then adopt Ceres to calculate the nonlinear least squares problem in the iterative process. Because both these two sets of constraints are in the form of three-dimensional vector $[x,y,\theta]$, the residual term generated is also in the same form, which can be written as Equation (6).

$$\begin{bmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ \theta_2 - \theta_1 \end{bmatrix} \tag{6}$$

In FOSLAM, Levenbergt-Marquardt-Fletcher (LMF) algorithm is used for iterative updating and Ceres method is adopt to help the calculation automatically. LMF gets rid of the restriction that it needs to iterate near the expansion point and uses Lagrange multiplier as a correction. The Lagrange multiplier is able to transform the least squares with point position constraints into least squares without constraints [15]. The essence of LMF is shown in Equation (7).

$$\Delta x = -(J_e^T J_e + \mu_k I)^{-1} J_e^T e \tag{7}$$

$\Delta x$ is the amount of iteration updates. $e$ is the error function. $-J_e^T e$ is the negative gradient of the target function. And the denominator term $J_e^T J_e + \mu_k I$ is the Hessian matrix of the objective function with modification. The error function Jacobian $J_e$ reflects the differential relationship between variables.

In the algorithm, the value of the modified weight $\mu_k$ is constantly adjusted, so that the ratio of the cost function $\Psi(x)$ and the approximate cost function $\varphi(x)$ approaches 1, indicating that the better the linearity is. The specific process is as Equation (8-10).

$$\Delta\psi = \psi(x^{(k)} + \Delta x) - \psi(x^{(k)}) = \psi(x^{(k+1)}) - \psi(x^{(k)}) \tag{8}$$

$$\Delta\varphi = \varphi(\Delta x) - \varphi(0) = 2\Delta x^T J_e^T e + \Delta x^T J_e^T J_e \Delta x \tag{9}$$

$$\gamma_k = \frac{\Delta\psi}{\Delta\varphi} = \frac{\psi(x^{(k+1)}) - \psi(x^{(k)})}{2\Delta x^T J_e^T e + \Delta x^T J_e^T J_e \Delta x} \tag{10}$$

Therefore, the goal of the algorithm to is to use the modified parameter $\mu_k$ to ensure that each update of the iterative update quantity $\Delta x$ is gradient descent.

In FOSLAM, Cauchy kernel function is selected to reduce the negative impact of incorrect closed-loop detection on the final optimization result and enhance the algorithm robustness to outliers [22]. FOSLAM also uses the Ceres nonlinear optimization library to construct the residual block between the global relative pose and the odometer pose. The algorithm iterate the pose residual to obtain the optimal solution by creating the sparse cost function of automatic differentiation and import the front-end calculated odometer pose.

## 4. Experiments and Results

The constraint construction strategy provided in this paper is implemented based on Google's Cartographer front-end construction idea, which is a real-time solution for building indoor grid maps. This section verifies the advantages of FOSLAM through comparative experiments. In the experiments, in order to control the system's cost, the robot is equipped with a single line laser LiDAR A1M8 (SLAMTEC, China), a nine-axis IMU JY901 (Wit-Motion, China) and a two-wheel differential chassis. The software framework is built based on ROS (Robot Operating System) and a mainstream SLAM algorithm deployed on it. The host Raspberry Pi 4B is mounted on the mobile robot and the secondary PC virtual machine uses Ubuntu 18.04 and ROS melodic version. The results of SLAM construction are observed through the Rviz 3D visualization plug-in that comes with ROS.

Fusion Odometer SLAM (FOSLAM) uses the relative pose calculated from the odometer data as a set of constraints and constructs a residual term by the global constraints between poses. Furthermore, FOSLAM uses LMF to guide the iterative process and enhances its robustness through Cauchy kernel function. Utilize Google's Ceres nonlinear optimization C++ package to resolve the nonlinear least square problem among constraints. The procedure of calculating constraints for FOSLAM is illustrated in Algorithm 1.

In FOSLAM, the Ceres method with numerical automatic derivation is used to avoid complex Jacobian calculation. Thus, the Ceres' function with the optimization variable and residual term passed in is the only requirement for solving the nonlinear least squares problem when constructing the cost function. In Algorithm 1, global_submap is the global coordinate of the origin node in the submap and global_node is the global coordinate of the pose node. The first constraint is the relative transformation between the pose node and the origin of the submap on the global coordinates, which is needed to be optimized in Ceres solution. The second constraint, which is marked as loro (local_observed_relative_ odometer), is the local relative coordinate transformation between their interpolation results. otw is odometer translation weight and orw is odometer rotation weight of the odometer. The main implementation steps are shown as follows.

**Algorithm 1.** The procedure of calculating constraints for FOSLAM

```
problem.AddResidualBlock (CostFunction, Cauchy
            kernel,
                        global_submap, global_node);
transform::Rigid3d begin_odometer =
            OdometerInterpolate(global_submap.time);
transform::Rigid3d end_odometer =
            OdometerInterpolate(global_node.time);
transform::Rigid3d loro =
            begin_odometer ->inverse() * end_odometer;
ceres::AutoDiffCostFunction CostFunction(loro, otw,
orw)
{  template <typename T>
     bool operator()(start_pose, end_pose, T* residual)
     const
     {ComputeUnscaledResidual(loro, start_pose,
     end_pose);
   residual[0] * otw;
   residual[1] * otw;
   residual[2] * orw;
   return true;}}
Ceres::Solver::Summary summary;
Ceres::Solve(options, &problem, &summary);
```

**Step 1:** Create the cost function and set the residual block. According to the time, the odometer data are interpolated to obtain the local poses at the beginning and the end of a period of time, which are ***begin_odometer*** and ***end_odometer*** respectively. The relative change between these two coordinates is ***loro***, which is a constraint relation. Then, the automatic differential cost function in Ceres is created and the variables need to be optimized are passed in. In that procedure, the ***global_submap*** corresponds to ***start_pose*** and the ***global_node*** corresponds to ***end_pose***, which is another constraint relation. Thus, the residual term between these two constraints is able to be calculated. The residual built has the same structure as the three-dimensional pose vector *T[x-,y,θ]*. There is a transformation between two different poses in global coordinate system. Then, by using the odometer data, another transformation between the same two poses can be discovered. As a result, a set of residuals is created based on these two coordinate transformations. In the algorithm, transformation is calculated as Equation (11), where $R$ is the rotation matrix and $t$ is the translation matrix.

$$T_{12} = T_1^{-1} \cdot T_2 = \begin{bmatrix} R_1^{-1} \cdot R_2 & R_1^{-1} \cdot (t_2 - t_1) \\ 0 & 1 \end{bmatrix} \tag{11}$$

**Step 2:** The LMF optimization approach is used to construct the optimization issue that needs to be solved. Use the automatic differential cost function to determine the Jacobi.

**Step 3:** Create a residual block and add a loss function. The Cauchy kernel function is used to reduce the negative impact of mismatching between global constraints and odometer constraints on the final optimization result.

Cauchy loss function is as Equation (12) shown, where $s$ is the square of the residual.

$$\rho(s) = \log(1+s) \tag{12}$$

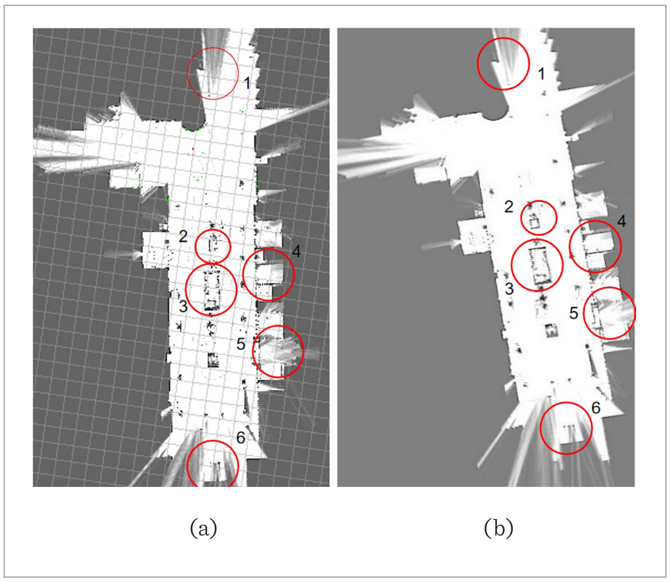
**Step 4:** The final residual block is added to Ceres to solve the least square problem with variable constraints.

The mapping results of a 30 m*100 m area from the conventional Cartographer algorithm and FOSLAM are shown in Figure 3. It can be found that FOSLAM performed better than Cartographer in terms of expressiveness. The items were deformed and their outlines were not discernible at positions 2 and 3 of Figure 3(a). However, in Figure 3(b), the objects' forms were more regular and FOSLAM lessened the effect of external noise during the mapping process. Furthermore, at the 1, 4, 5 and 6 places and edges in Figure 3, the FOSLAM mapping is far smoother and less jagged. The importation of odometer data, which results in more precise poses of scan matching in the front-end, is the primary cause of this improvement.

The front-end scan matching analysis was also done to confirm the accuracy of the results. The robot was set

**Figure 3**

(a) Cartographer mapping result on ROS in a 30 m*100 m area; (b) FOSLAM mapping result on ROS in a 30 m*100 m area



up in the same location. The global map was created by generating a total of 10 submaps. The scan matching of FOSLAM saves the matching score of the node pose to back end. The average scan matching score for each submap during the experiment was obtained from the algorithm's back-end.

For comparison, the two conventional laser SLAM algorithms Karto [18] and Hector [17] were also involved in the experiments. The results are shown in Figure 4, Tables 1 and 2. Based on the information in Figure 4(a) and Table 1, Cartographer has the maximum score of 67.5% and the minimum score of 52.3% during the initial mapping phase, which is the construction of the first four submaps, with a difference of 15.2%. While the maximum score of FOSLAM is 65.6% and the minimum score is 56.2%, with a difference of 9.3%. According to the above analysis, the variation of scan matching score of FOSLAM is 9.3%, which is 5.9% less than the one of Cartographer in the initial mapping phase. In order to make the global poses more accurate, the FOSLAM integrates the odometer data into the back end for constraint construction when estimating the poses. Therefore, the prior poses used by FOSLAM scan matching is closer to the real value. That results in a smoother transition proceduire between the neighboring two frames of laser data, and avoiding the distortion of the mapping caused by the large fluctuation in the scan matching. Thus, the performance of FOSLAMin the initial mapping stage is more stable.

Moreover, Figure 4(a) and Table 1 show the changing trend of the scan matching scores during the entire process. After the fourth submap, FOSLAM had a score range between 57.7% and 61.1% with a difference of 3.4%. The Car-

**Figure 4**

(a) The average score of scan matching in each submap; (b) The maximum score, minimum score and average score of four algorithms in front-end scan matching

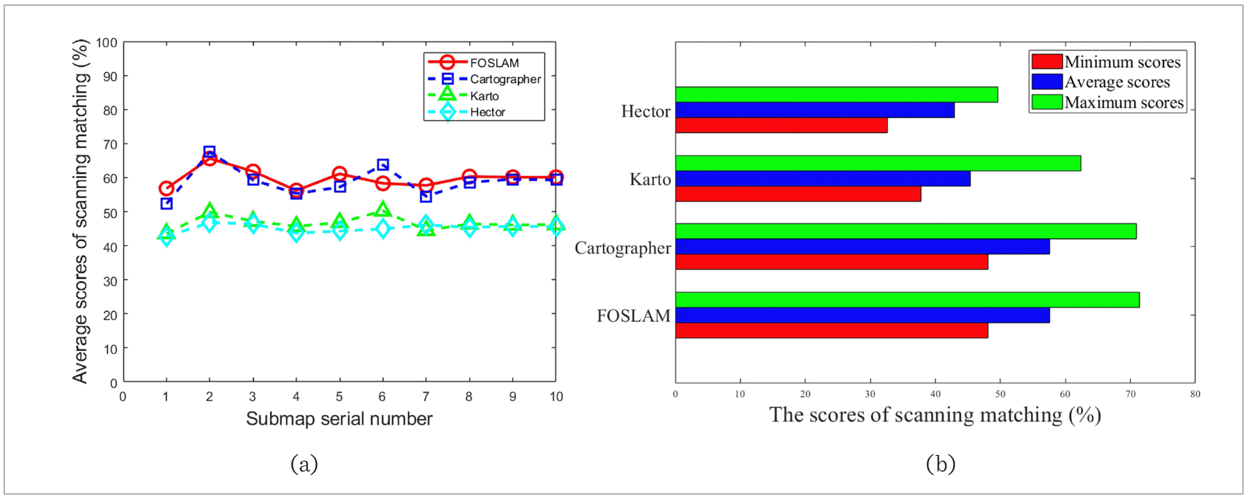


**Table 1**
The average score of scan matching for each submap

| Submap number | The average score of FOSLAM for scan matching | The average score of Cartographer for scan matching | The average score of Karto for scan matching | The average score of Hector for scan matching |
|---|---|---|---|---|
| 1 | 56.8% | 52.3% | 43.6% | 42.7% |
| 2 | 65.6% | 67.5% | 49.8% | 46.8% |
| 3 | 61.8% | 59.3% | 47.2% | 46.4% |
| 4 | 56.2% | 55.3% | 45.7% | 43.7% |
| 5 | 61.1% | 57.2% | 46.8% | 44.3% |
| 6 | 58.3% | 63.7% | 50.2% | 44.9% |
| 7 | 57.7% | 54.5% | 44.5% | 46.1% |
| 8 | 60.3% | 58.6% | 46.4% | 45.4% |
| 9 | 60.1% | 59.5% | 46.1% | 45.7% |
| 10 | 60.1% | 59.3% | 46.2% | 45.6% |

**Table 2**
The maximum score, minimum score and average score of four algorithms

| | maximum score | minimum score | average score |
|---|---|---|---|
| FOSLAM | 71.37% | 48.11% | 57.56% |
| Cartographer | 70.91% | 48.11% | 57.59% |
| Karto | 62.35% | 37.72% | 45.34% |
| Hector | 49.62% | 32.65% | 42.91% |

tographer received a score range from 54.5% to 63.7% with a 9.2% difference. These data clearly showed that FOSLAM scan matching has stronger convergence ability than Cartographer. Furthermore, it can be seen from Figure 4(a) that the scan matching curve of FOSLAM at the later stage of mapping has a more gentle variation. In particular, between submap 6 and submap 7, the score of FOSLAM changed by only 0.6%, while that of Cartographer changed by 9.2%. This also resulted in distorted and jagged map on Cartographer, as evidenced by position 2 and position 3 in Figure 3. The two-dimensional map created by FOSLAM was smoother in Figure 3 due to the significant convergence. It can also be clearly seen in Figure 4(a) and Table 1 that the scan matching results of FOSLAM outperforms those of conventional Karto and Hector.

The average matching score for FOSLAM is 57.56%, nearly the same as Cartographer's, but 12.22% and 14.65% higher than that for Karto and Hector, respectively. The scan matching process of FOSLAM was made easier to be convergent by including odometer data in the constraint building, resulting in a smoother map and less jagged edges.

The space and time complexity of the algorithms were further checked to confirm whether FOSLAM can be implemented on lightweight embedded devices. The experiments primarily compared the mapping performance and speed of FOSLAM with Cartographer, Karto and Hector. The four methods unified the timestamp in ROS environment and output the log. Under the same experimental scenario, FOSLAM and Cartographer needed five stages to complete the mapping, meanwhile Karto and Hector needed six stages. Table 3 displays the mapping degree and time consumption for entire mapping process.

Under the identical circumstances, FOSLAM took 48 s on average for mapping, compared to 46 s on aver-

**Table 3**
Mapping degree and time consumption for entire mapping process

| Stage | FOSLAM's mapping degree | Cartographer's mapping degree | Karto's mapping degree | Hector's mapping degree |
|---|---|---|---|---|
| 1 | 13.42% | 14.71% | 10.27% | 9.56% |
| 2 | 34.66% | 35.10% | 28.09% | 26.31% |
| 3 | 57.81% | 57.32% | 50.75% | 49.72% |
| 4 | 81.97% | 83.02% | 69.39% | 69.14% |
| 5 | 100.00% (48s) | 100.00% (46s) | 84.71% | 83.62% |
| 6 | Completed | Completed | 100.00% (54s) | 100.00% (58s) |

**Table 4**
Mapping degree for each stage

| Stage | FOSLAM's incremental mapping degree for each stage | Cartographer's incremental mapping degree for each stage | Karto's incremental mapping egree for each stage | Hector's incremental mapping degree for each stage |
|---|---|---|---|---|
| 1 | 13.42% | 14.71% | 10.27% | 9.56% |
| 2 | 21.24% | 20.39% | 17.82% | 16.75% |
| 3 | 23.15% | 22.22% | 22.66% | 23.41% |
| 4 | 24.16% | 25.70% | 18.64% | 19.42% |
| 5 | 18.03% | 16.98% | 15.32% | 14.48% |
| 6 | Completed | Completed | 15.29% | 16.38% |

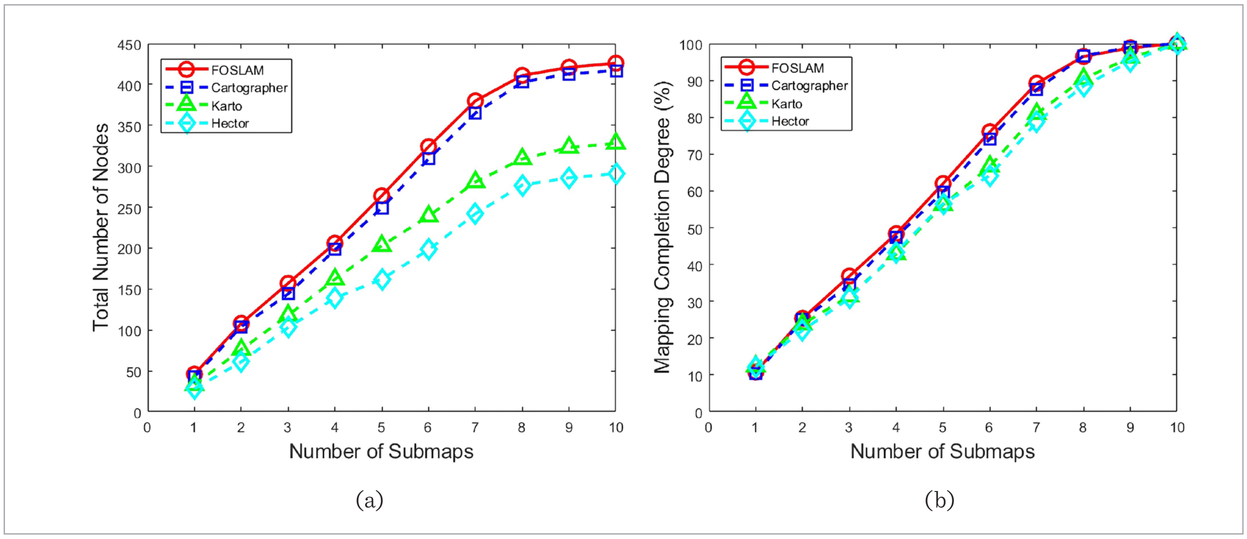
age for Cartographer. Table 3 shows the total mapping degree at the end of each phase. For example, at the end of Stage 3, the FOSLAM mapping degree is 57.81%, while Hector had mapped only 49.72%. The incremental mapping degrees for each stage were calculated and shown in Table 4, FOSLAM was mapped at 13.42%, 21.24%, 23.15%, 24.16% and 18.03%, respectively. The mapping degrees of Cartographer for each of the five phases were 14.71%, 20.39%, 22.22%, 25.7% and 16.98%, respectively. Thus, FOSLAM and Cartographer had similar mapping degree at each stage. The efficiency of the mapping completion at each stage is also comparable. Karto and Hector took 54 s and 58 s on average to map the same area, respectively. Due to the fact that the integration of the odometer data does not increase the computing load, the mapping time consumption of FOSLAM was still comparable to that of Cartographer.

The scan matching process continuously saves the successfully matched node data to the back end during SLAM mapping. Thus, besides comparing the time complexity, the experiments also tested their memory consumption of nodes and submaps. The testing processes were excuted in the same 30 m*100 m area. In the test, the total number of submaps and nodes consumed were extracted from the back end of the algorithm. Ten submaps were constructed for each algorithm to facilitate analysis. All data generated during the mapping process were saved to the dataset corresponding to the 10 submaps. The results were shown in Figure 5. The global map was made up of these 10 submaps and Tables 5 and 6 display the data.

As shown in Figure 5(a), during the process of creating the first 7 submaps, the total number of pose nodes produced by FOSLAM and Cartographer gradually increased with comparatively constant growth rate. During the process of creating the last 3 submaps, the node growth rate is relatively slow and the number of nodes tends to be stable. This is mainly due to the fact

**Figure 5**

(a) The total number of nodes generated with the creation of the submaps; (b) The mapping completion degree with the creation of submaps



that the driving route of robot in the experiment is a loop. The nodes generated overlapped with some of the ones at the beginning when the robot travelled to the end of the loop. To reduce computation, duplicate nodes are no longer saved to the back end. According to the Table 5, FOSLAM generated a total of 426 pose nodes during the entire mapping process compared with 417 nodes made by Cartographer.

From the perspective of memory consumption, the considerable improvement of FOSLAM in the mapping performance is only at the cost of the more memory consumption of 9 nodes. According to Table 7, the incremental mapping degree for each submap of FOSLAM is 10.79%, 14.56%, 11.5%, 11.5%, 13.62%, 14.08%, 13.15%, 7.27%, 2.35% and 1.18%, respectively, which are nearly same to Cartographer. The most difference occurs in the third submap, FOSLAM generated only 1.91% of the additional cost. This results showed that involving of odometer constraint in the back end does not increase the space complexity of the submap observably, but improves the accuracy of mapping.

The number of nodes generated by Karto and Hector are fewer, but they cannot complete the high-precision mapping tasks. The space complexity of FOSLAM is also reflected in the number of pose nodes contained in each submap. The number of nodes in each submap is a factor in the space complexity of algorithms in addition to the changing trend of the total number of nodes shown in Figure 5. Only FOSLAM and Cartographer are taken into account here because Karto and Hector are rarely used in practical engineering. As shown in Figure 6.

**Table 5**

The total number of nodes generated by the creation of the submaps

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| FOSLAM | 46 | 108 | 157 | 206 | 264 | 324 | 380 | 411 | 421 | 426 |
| Cartographer | 43 | 104 | 144 | 198 | 249 | 309 | 365 | 403 | 413 | 417 |
| Karto | 33 | 76 | 118 | 162 | 203 | 239 | 281 | 309 | 323 | 328 |
| Hector | 28 | 61 | 103 | 139 | 162 | 198 | 242 | 277 | 286 | 291 |

**Table 6**
The mapping degree generated by the creation of submaps for entire mapping process

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| FOSLAM | 10.79% | 25.35% | 36.85% | 48.35% | 61.97% | 76.05% | 89.20% | 96.47% | 98.82% | 100% |
| Cartographer | 10.31% | 24.94% | 34.53% | 47.48% | 59.71% | 74.10% | 87.53% | 96.64% | 99.04% | 100% |
| Karto | 12.38% | 23.80% | 31.42% | 42.85% | 56.19% | 66.66% | 80.95% | 90.47% | 96.19% | 100% |
| Hector | 12.27% | 22.11% | 30.82% | 43.38% | 56.50% | 64.17% | 78.69% | 88.47% | 95.13% | 100% |

**Table 7**
The incremental mapping degree for each submap

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| FOSLAM | 10.79% | 14.56% | 11.50% | 11.50% | 13.62% | 14.08% | 13.15% | 7.27% | 2.35% | 1.18% |
| Cartographer | 10.31% | 14.63% | 9.59% | 12.95% | 12.23% | 14.39% | 13.43% | 9.11% | 2.40% | 0.96% |
| Karto | 12.38% | 11.42% | 7.62% | 11.43% | 13.34% | 10.47% | 14.29% | 9.52% | 5.72% | 3.81% |
| Hector | 12.27% | 9.84% | 8.71% | 12.56% | 13.12% | 7.67% | 14.52% | 9.78% | 6.66% | 4.87% |

**Figure 6**
Number of nodes contained in each submap

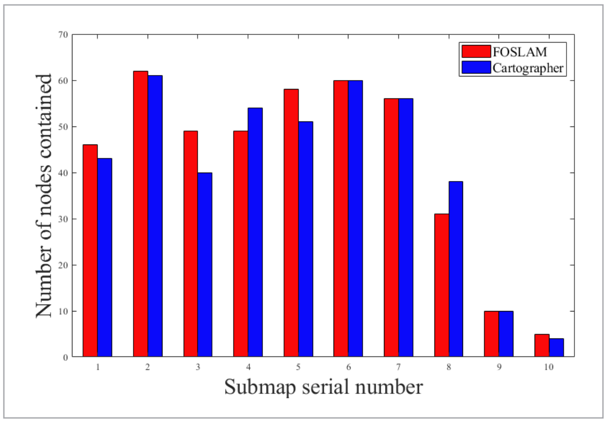


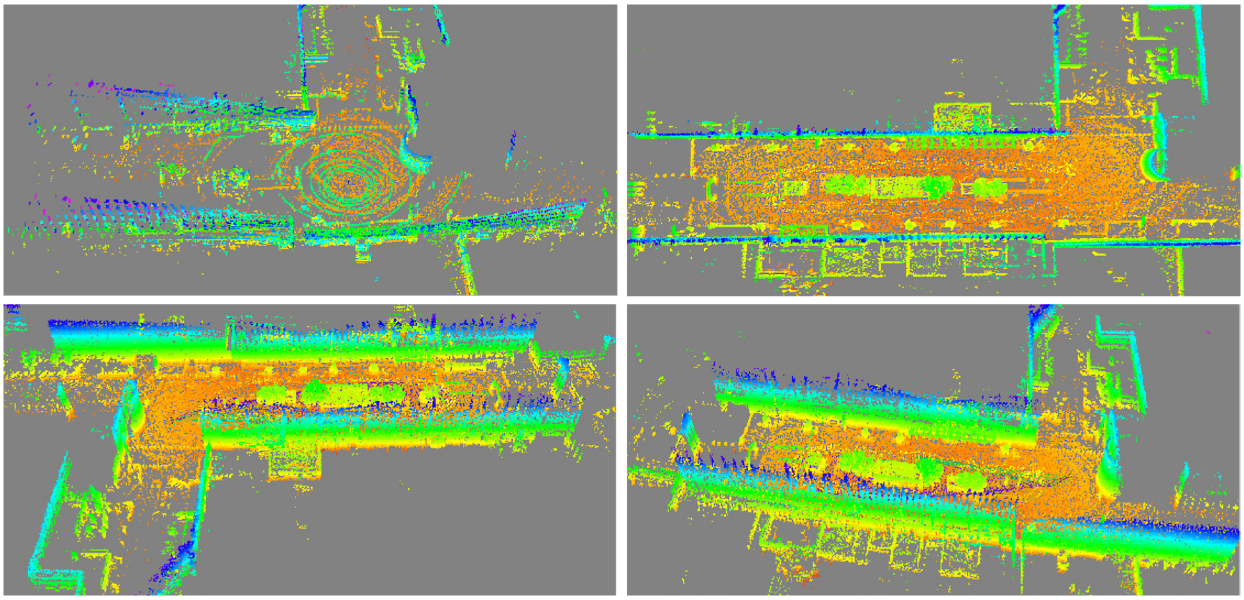
Figure 6 shows the number of nodes for each submap of FOSLAM and Cartographer. The third submap of FOSLAM contains 9 more nodes than the ones of Cartographer. Furthermore, from the whole process, the number of nodes contained in each submap are also consistent with the data in Table 4. Thus, it also shows that FOSLAM does not increase memory usage too much. The above analysis results show that FOSLAM does not occupy much computing resources and cost less memory. Thus, it is suitable to be run on lightweight embedded.

devices with high accuracy and smoother map. Finally, do supplemental verification of the 3D point cloud mapping produced by FOSLAM, as illustrated in Figure 7. The findings demonstrate that FOSLAM also has strong stability when it comes to 3D SLAM and also can achieve high accuracy when building dense point clouds of nearby objects.

According to the performance under the above experimental scenarios, the advantages of FOSLAM can be found. In the 2DSLAM field nowadays, the popular methods are Hector, Karto and Cartographer. Cartographer is considered to be the most commonly implemented method among these three methods in engineering. For the new FOSLAM method proposed in this paper, it involves real-time local pose relationship obtained by front-end odometer to construct constraint terms. The constraint relationship between poses is strengthened and the convergence of scan matching score is improved. Therefore, compared with the state-of-the-art Cartographer algorithm, FOSLAM performed smoother and less jaggedly due to strong constraints between neighbouring poses. In terms of the time consumption under the same scene, FOSLAM has the advantage of time consumption compared with Hector and Karto, and its mapping efficiency is higher. Moreover, through algorithm complexity analysis, all these improvements will not

**Figure 7**

3D point cloud mapping



increase time and space consumption much and the method is able to be lightweight implementation.

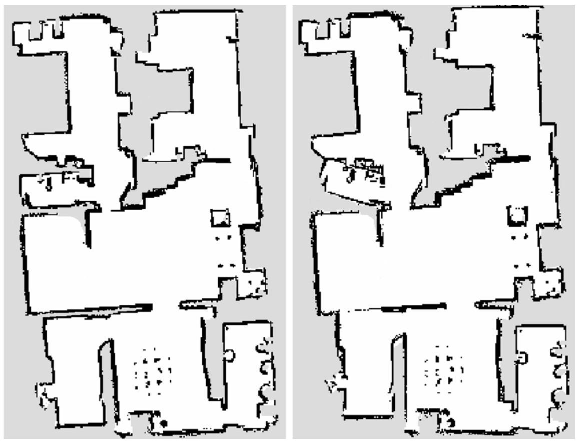
In order to verify the performance of FOSLAM under real environment, a teaching building with high complexity and diverse details for global mapping is selected as the testing scene. The original laboratory equipment and objects are kept without moving. In this experiment, FOSLAM is performed compared with Cartographer, involving mapping, scan matching and accuracy. The global mapping effect of FOSLAM and Cartographer is shown in Figure 8.

As shown in Figure 8, it can be found that the mapping result of Cartographer showed map drift and overlap. Ten mapping tests were carried out by Cartographer under the same scenario, the robot path and speed were kept unchanged. On average, eight out of ten mapping tests resulted in drift and overlap as shown in Figure 8(b). However, the mapping result of FOSLAM can always reflect the accurate size of the map. This is because FOSLAM introduces the real-time pose data acquired by the front-end odometer to construct residual terms as constraint conditions.

During the experiment, FOSLAM generated 9815 pose nodes in global mapping while Cartographer generated 9748. In order to estimate the influence of the odometer constraint fusion on the front-end scan

**Figure 8**

Mapping results of a floor of the teaching building (a) FOSLAM mapping result; (b) Cartographer mapping result

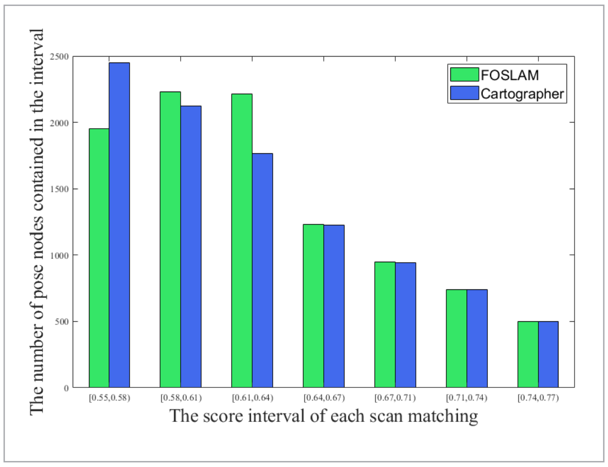


(a) (a)

matching procedure, the number of nodes in each interval of the scan matching scores was counted. Figure 9 shows the number of pose nodes in each score interval acquired by these two methods at the front-end of SLAM. The matching scores from FOSLAM were mostly between [0.58,0.64], while those of Cartographer were mostly between [0.55,0.58]. This indi-

**Figure 9**

The number of pose nodes contained in each score interval of scan matching at the front-end of SLAM

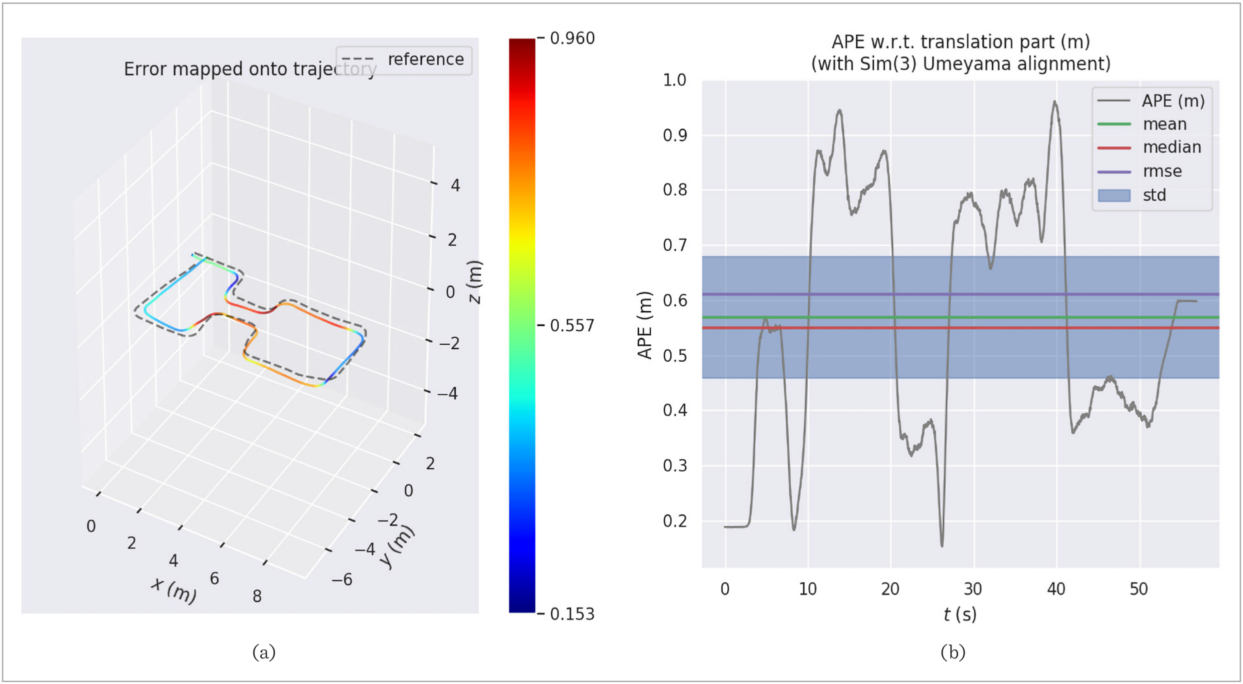


cates that the pose nodes of FOSLAM are more well-matched to the map, and the same conclusion can also be found in Figure 8.

For large-scale mapping, the cumulative errors of the laser odometer at the front-end of SLAM will be constantly superimposed, resulting in reduction of the track accuracy. The odometer constraint construction method continuously obtains the pose of the constructed trajectory for optimization and feeds it back to the front-end of SLAM for subsequent scan matching. For error evaluation, APE (Absolute Pose Error), RPE (Relative Pose Error) and RMSE (Root Mean Square Error) error indexes are the commonly used evaluating indicators [23, 34]. Figure 10 (a) shows the global APE error between the laser odometer of Cartographer after back-end pose optimization and the actual trajectory of the robot in a large scene. Figure 10 (b) shows the change curve of that APE error over time. In the same way, Figure 11 shows the APE situation of FOSLAM method.

**Figure 10**

The APE situation of Cartographer method. (a) Global APE error between the estimated trajectory from the Cartographer laser odometer and the actual trajectory (b) The change curve of the APE error over time



The comparative results for other APE error indicators are shown in Table 8. In the case of the same trajectory, the comparative results of RPE error are shown in Table 9.

**Figure 11**

The APE situation of FOSLAM method. (a) Global APE error between the estimated trajectory from the FOSLAM laser odometer and the actual trajectory (b) The change curve of the APE error over time

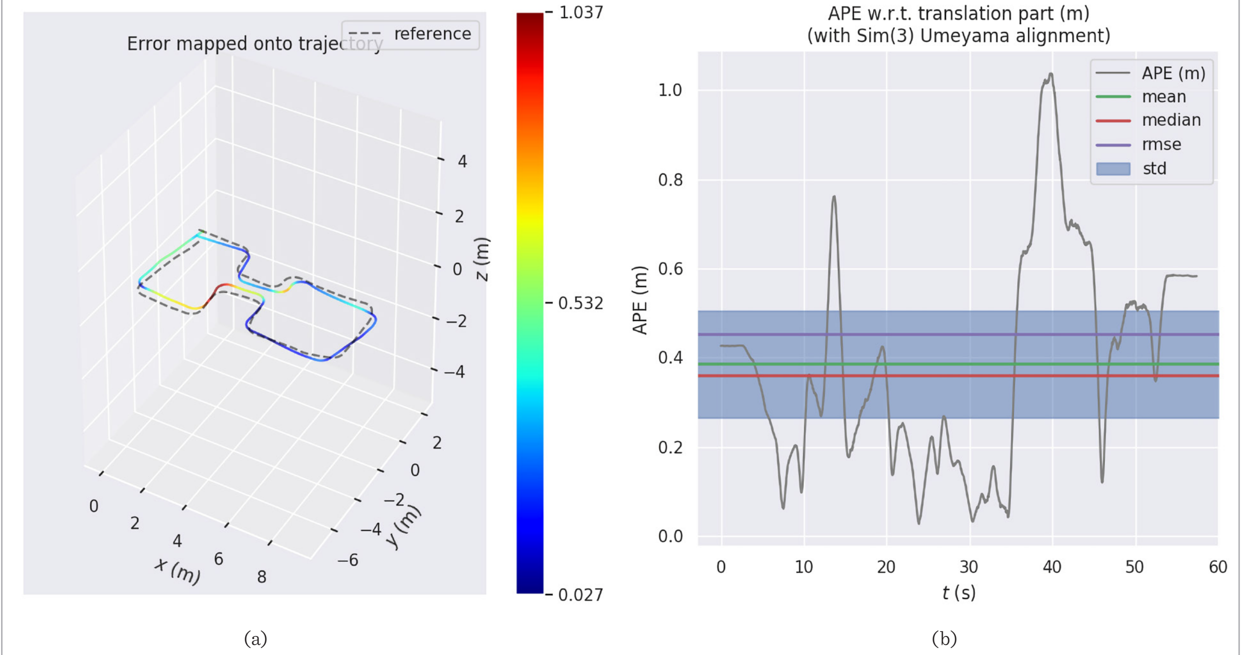


**Table 8**

The MEAN error, RMSE and standard deviation data of APE evaluation criteria from Cartographer and FOSLAM performance under large applicaition scene, respectively

| | MEAN (m) | RMSE (m) | STD (m) |
|---|---|---|---|
| Cartographer | 0.569705 | 0.610834 | 0.220350 |
| FOSLAM | 0.385600 | 0.452806 | 0.217372 |

**Table 9**

The MEAN error, RMSE and standard deviation data of RPE evaluation criteria from Cartographer and FOSLAM performance under large applicaition scene, respectively

| | MEAN (m) | RMSE (m) | STD (m) |
|---|---|---|---|
| Cartographer | 0.002238 | 0.003228 | 0.002376 |
| FOSLAM | 0.002185 | 0.002965 | 0.001945 |

According to the experimental results, the FOSLAM's estimated trajectories are more accurate than Cartographer's. The global evaluation results of APE showed that the trajectory acquired by the FOSLAM is more globally consistent. Its mean error is about 0.18 m lower than Cartographer's and its root mean square error was about 0.16 m lower than Cartographer. However, the traditional back-end optimization algorithms, such as Cartographer, lack the real-time front-end pose information and the constraints between pose nodes are relatively simple. Therefore, their laser odometer estimated results fed back to the front-end function are less accurate than FOSLAM. These results can also be found in the RPE evaluation of the trajectory. Furthermore, FOSLAM can enhance the constraint between nodes without sacrificing too much running time and memory space, improve the convergence of the front-end scan matching score of the SLAM system and then make the mapping performance smoother and more accurate than Cartographer. FOSLAM is also more efficient than Hector and Karto.

## 5. Conclusion

In this paper, we introduced a FOSLAM method which fuse the front-end two-wheel differential odometer data into the back-end pose graph optimization and use the Ceres nonlinear least square library to construct residual. The advantage of the algorithm was verified on a dataset of 30 m*100 m area by experiments. Global optimization of node pose is performed by using Ceres method. The optimized global pose makes the prior data of the front-end more accurate and the scan matching process smoother.

The data analysis results show that FOSLAM does not increase the redundant time and space complexity compared to the traditional method Cartographer. Furthermore, it optimized the convergence of front-end scan matching and improved the robustness of the mapping process. Moreover, FOSLAM achieves smoother in terms of rendering and its match performance is better than some popular laser SLAM algorithms, such as Hector and Karto. In experiments, FOSLAM is evaluated against Cartographer, state-of-the-art 2D laser SLAM algorithm, in a floor scenario with higher environmental complexity. FOSLAM's mapping results are consistently better than Cartographer's and its front-end laser odometer trajectory estimation is also more accurate than Cartographer's.

FOSLAM is suitable to be applied on indoor robot for cleaning and inspection and can be further deployed on autonomous unmanned vehicals. Although the lack of visual information makes its performance slightly insufficient when facing complex environments, its low cost, relatively high reliability and rapid response capabilities make it more suitable for large-scale deployments, especially under economical environments and harsh working conditions. In the future, the addition of neuro-heuristic algorithms may further increase its performance.

### Acknowledgement

This project is supported by the Natural Science Foundation of Fujian Province in China (2022J01609), the Outstanding Youth Fund of Fujian Agriculture and Forestry University (XJQ201820) and the Science and Technology Innovation Special Fund of Fujian Agriculture and Forestry University (CXZX2020132B)

## References

1. Ahmed, M. F., Masood, K., Fremont, V., Fantoni, I. Active SLAM: A Review on Last Decade. Sensors, 2023, 23(19). https://doi.org/10.3390/s2319809
2. Aloise, I., Grisetti, G. Chordal Based Error Function for 3-D Pose-Graph Optimization. IEEE Robotics and Automation Letters, 2019, 5(1), 274-281.https://doi.org/10.1109/LRA.2019.2956456
3. Bao, Y. Q., Yang, Z., Pan, Y., Huan, R. H. Semantic-Direct Visual Odometry. IEEE Robotics and Automation Letters, 2022, 7(3), 6718-6725. https://doi.org/10.1109/LRA.2022.3176799
4. Chou, C. C., Chou, C. F. Efficient and Accurate Tightly-Coupled Visual-Lidar SLAM. IEEE Transactions on Intelligent Transportation Systems, 2022, 23(9), 14509-14523. https://doi.org/10.1109/TITS.2021.3130089
5. Dong, N., Qin, M. H., Chang, J. F., Wu, C. H., Ip, W. H., Yung, K. L. Weighted Triplet Loss Based on Deep Neural Networks for Loop Closure Detection in VSLAM. Computer Communications, 2022, 186, 153-165. https://doi.org/10.1016/j.comcom.2022.01.013
6. Eckenhoff, K., Paull, L., Huang, G. Decoupled, Consistent Node Removal and Edge Sparsification for Graph-Based SLAM. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, 3275-3282. https://doi.org/10.1109/IROS.2016.7759505
7. Gao, L., Dong, C., Liu, X., Ye, Q., Zhang, K., Chen, X. Improved 2D Laser SLAM Graph Optimization Based on Cholesky Decomposition. 2022 8th International Conference on Control, Decision and Information Technologies (CoDIT), 2022, 659-662. https://doi.org/10.1109/CoDIT55151.2022.9803938
8. Geng, X. X., Huang, G., Zhao, W. X. Almost Sure Convergence of Randomised-Difference Descent Algorithm for Stochastic Convex Optimisation. IET Control Theory and Applications, 2021, 15(17), 2183-2194. https://doi.org/10.1049/cth2.12184
9. Gong, Z., Li, J., Luo, Z. P., Wen, C. L., Wang, C., Zelek, J. Mapping and Semantic Modeling of Underground Parking Lots Using a Backpack LiDAR System. IEEE Transactions on Intelligent Transportation Systems, 2021, 22(2), 734-746. https://doi.org/10.1109/TITS.2019.2955734

10. He, S. H., Li, Y. Z., Lu, Y. K., Liu, Y. S. Design of Visual Inertial State Estimator for Autonomous Systems via Multi-Sensor Fusion Approach. Mechatronics, 2023, 95. https://doi.org/10.1016/j.mechatronics.2023.103066

11. Hess, W., Kohler, D., Rapp, H., Andor, D. Real-Time Loop Closure in 2D LIDAR SLAM. 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, 1271-1278. https://doi.org/10.1109/ICRA.2016.7487258

12. Jamaludin, A., Yatim, N. M., Noh, Z. M., Buniyamin, N. Rao-Blackwellized Particle Filter Algorithm Integrated with Neural Network Sensor Model Using Laser Distance Sensor. Micromachines, 2023, 14(3). https://doi.org/10.3390/mi14030560

13. Jeong, H., Lee, H. C. CNN-Based Fault Detection of Scan Matching for Accurate SLAM in Dynamic Environments. Sensors, 2023, 23(6). https://doi.org/10.3390/s23062940

14. Jia, Z., Leng, J. Mobile Robot Vision Odometer Based on Point-Line Features and Graph Optimization. 2018 Chinese Control and Decision Conference (CCDC), 2018, 3398-3403. https://doi.org/10.1109/CCDC.2018.8407711

15. Jurić, A., Kendeš, F., Marković, I., Petrović, I. A Comparison of Graph Optimization Approaches for Pose Estimation in SLAM. 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), 2021, 1113-1118. https://doi.org/10.23919/MIPRO52101.2021.9596721

16. Kachurka, V., Rault, B., Muñoz, F. I. I., Roussel, D., Bonardi, F., Didier, J. Y., et al. WeCo-SLAM: Wearable Cooperative SLAM System for Real-Time Indoor Localization Under Challenging Conditions. IEEE Sensors Journal, 2022, 22(6), 5122-5132. https://doi.org/10.1109/JSEN.2021.3101121

17. Kohlbrecher, S., Von Stryk, O., Meyer, J., Klingauf, U. A Flexible and Scalable SLAM System with Full 3D Motion Estimation. 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, 2011, 155-160. https://doi.org/10.1109/SSRR.2011.6106777

18. Konolige, K., Grisetti, G., Kümmerle, R., Burgard, W., Limketkai, B., Vincent, R. Efficient Sparse Pose Adjustment for 2D Mapping. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, 22-29. https://doi.org/10.1109/IROS.2010.5649043

19. Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W. g2o: A General Framework for Graph Optimization. 2011 IEEE International Conference on Robotics and Automation, 2011, 3607-3613. https://doi.org/10.1109/ICRA.2011.5979949

20. Li, L., Bano, S., Deprest, J., David, A. L., Stoyanov, D., Vasconcelos, F. Globally Optimal Fetoscopic Mosaicking Based on Pose Graph Optimisation With Affine Constraints. IEEE Robotics and Automation Letters, 2021, 6(4), 7831-7838. https://doi.org/10.1109/LRA.2021.3100938

21. Pfeifer, T., Lange, S., Protzel, P. Advancing Mixture Models for Least Squares Optimization. IEEE Robotics and Automation Letters, 2021, 6(2), 3941-3948. https://doi.org/10.1109/LRA.2021.3067307

22. Qi, L., Shen, M., Wang, D., Wang, S. Robust Cauchy Kernel Conjugate Gradient Algorithm for Non-Gaussian Noises. IEEE Signal Processing Letters, 2021, 28, 1011-1015. https://doi.org/10.1109/LSP.2021.3081381

23. Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D. A Benchmark for the Evaluation of RGB-D SLAM Systems. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, 573-580. https://doi.org/10.1109/IROS.2012.6385773

24. Su, P., Luo, S. Y., Huang, X. C. Real-Time Dynamic SLAM Algorithm Based on Deep Learning. IEEE Access, 2022, 10, 87754-87766. https://doi.org/10.1109/ACCESS.2022.3199350

25. Tan, C. H., Bin Shaiful, D. S., Tang, E., Soh, G. S., Foong, S. Flydar: A Passive Scanning Flying Lidar Sensing System for SLAM Using a Single Laser. IEEE Sensors Journal, 2022, 22(2), 1746-1755. https://doi.org/10.1109/JSEN.2021.3131209

26. Vallvé, J., Solà, J., Andrade-Cetto, J. Pose-Graph SLAM Sparsification Using Factor Descent. Robotics and Autonomous Systems, 2019, 119, 108-118. https://doi.org/10.1016/j.robot.2019.06.004

27. Von Stumberg, L., Wenzel, P., Yang, N., Cremers, D. LM-Reloc: Levenberg-Marquardt Based Direct Visual Relocalization. 2020 International Conference on 3D Vision, 2020. https://doi.org/10.1109/3DV50981.2020.00107

28. Wang, C., Li, Z. B., Kang, Y. F., Li, Y. Z. Applying SLAM Algorithm Based on Nonlinear Optimized Monocular Vision and IMU in the Positioning Method of Power Inspection Robot in Complex Environment. Mathematical Problems in Engineering, 2022, 2022. https://doi.org/10.1155/2022/3378163

29. Wang, X. H., Ma, X., Li, Z. W. Research on SLAM and Path Planning Method of Inspection Robot in Complex Scenarios. Electronics, 2023, 12(10). https://doi.org/10.3390/electronics12102178

30. Yang, A., Cao, Y., Liu, Y., Zeng, Q. C., Xiu, F. Q. AGV Robot for Laser-SLAM Based Method Testing in Automated Container Terminal. Industrial Robot-the International Journal of Robotics Research and Application, 2023, 50(6), 969-980. https://doi.org/10.1108/IR-04-2023-0063

31. Yang, J. D., Song, H. M., Li, X. X., Hou, D. Block Mirror Stochastic Gradient Method for Stochastic Optimization. Journal of Scientific Computing, 2023, 94(3). https://doi.org/10.1007/s10915-023-02110-y

32. Yoo, W., Kim, H., Hong, H., Lee, B. H. Scan Similarity-Based Pose Graph Construction Method for Graph SLAM. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, 131-136. https://doi.org/10.1109/IROS.2018.8593605

33. Zhang, H., Chen, N., Fan, G., Yang, D. An Improved Scan Matching Algorithm in SLAM. 2019 6th International Conference on Systems and Informatics (ICSAI), 2019, 160-164. https://doi.org/10.1109/ICSAI48974.2019.9010259

34. Zhang, Z., Scaramuzza, D. A Tutorial on Quantitative Trajectory Evaluation for Visual (-Inertial) Odometry. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, 7244-7251. https://doi.org/10.1109/IROS.2018.8593941

This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution 4.0 (CC BY 4.0) License (http://creativecommons.org/licenses/by/4.0/).