J. Shanghai Jiao Tong Univ. (Sci.), 2020, 25(5): 650-658
https://doi.org/10.1007/s12204-020-2201-4

Low Data Overlap Rate Graph-Based SLAM with Distributed Submap Strategy

XIANG Jiawei[1*] (相家炜), *ZHANG Jinyi*[1] (张金艺), *WANG Bin*[1] (王　彬), *MA Yongbin*[2] (马永滨)
(1. Key laboratory of Specialty Fiber Optics and Optical Access Networks, Shanghai University, Shanghai 200444, China;
2. Institute of System Integration, Shanghai Sansi Electronic Engineering Co., Ltd., Shanghai 201100, China)

Abstract: Simultaneous localization and mapping (SLAM) is widely used in many robot applications to acquire the unknown environment's map and the robot's location. Graph-based SLAM is demonstrated to be effective in large-scale scenarios, and it intuitively performs the SLAM as a pose graph. But because of the high data overlap rate, traditional graph-based SLAM is not efficient in some respects, such as real time performance and memory usage. To reduce data overlap rate, a graph-based SLAM with distributed submap strategy (DSS) is presented. In its front-end, submap based scan matching is processed and loop closing detection is conducted. Moreover in its back-end, pose graph is updated for global optimization and submap merging. From a series of experiments, it is demonstrated that graph-based SLAM with DSS reduces 51.79% data overlap rate, decreases 39.70% runtime and 24.60% memory usage. The advantages over other low overlap rate method is also proved in runtime, memory usage, accuracy and robustness performance.

Key words: graph-based SLAM, distributed submap strategy, data overlap rate
CLC number: TP 242　　**Document code:** A

0 Introduction

Simultaneous localization and mapping (SLAM) is the key algorithm for robot to localize itself and map in an unknown environment. The last several decades have seen extensive work in SLAM research. Many surveys have proved that graph-based SLAM[1-2] is much more effective than probabilistic SLAM[3], as graph-based SLAM can revisit previous data while probabilistic SLAM just processes and discards the data[4]. Advanced graph-based SLAM consists of front-end[5] and back-end[6]. The front-end processes the new scan by scan matching for localization and loop closing detection for data association. The back-end updates the pose graph by the result of front-end. After that, global optimization will be executed to decrease accumulated error.

However, traditional graph-based SLAM has a common weakness that scans contain too much overlapped data. Thus, the pose graph grows rapidly and too much memory usage is wasted. And loop closing detection must match the new scan to detected scans, thus the real time performance is not well. To reduce data overlap rate, Mazuran et al.[7] presented an algorithm with the generic linear constraint (GLC) factors and the nonlinear graph sparsification (NGS) method to delete those less informative nodes. The algorithm operates on the Markov blanket of marginalized node. Dube et al.[8] used a sliding window to limit the optimization area within the pose graph. This algorithm reduces the number of parameters to be optimized. Zhao et al.[9] split the pose graph into many subgraphs of different levels, and processed those subgraphs parallel by their levels. Although these algorithms can effectively reduce data overlap rate, there are some drawbacks. For example, they are easy to lead a failure optimization because of information loss in loop closing detection or global optimization, and they show less effect when robot is circling.

To decrease data overlap rate efficiently, a graph-based SLAM with distributed submap strategy (DSS) is presented. Its front-end processes submap based scan matching by using Levenberg-Marquardt (LM) method. If the new scan is too far to the last submap, a new submap will be built according to the scan. Therefore, submaps are distributed. Then, loop closing detection which uses a rough matching by lookup table and a precise matching by LM method is also applied. The back-end updates the pose graph by data extracted from the front-end, and it uses edge quality

Received date: 2018-05-23
Foundation item: the Project Fund for Key Discipline of the Shanghai Municipal Education Commission (No. J50104), and the Major State Basic Research Development Program of China (No. 2017YFB0403500)
***E-mail:** xiang_jiawei@163.com

check to delete low quality edges. According to pose graph, global optimization and submap merging are conducted. Experimental results show 51.79% decrease in data overlap rate. Runtime is reduced 39.70% and memory usage is reduced 31.15%. To prove the advantage over other low overlap rate methods, the comparison among DSS, linear SLAM and cartographer is also conducted. Moreover, the ability of DSS in accuracy and robustness performance is also showed.

This paper is organized as follows. Section 1 anatomizes graph-based SLAM. Section 2 describes graph-based SLAM with DSS in detail. In Section 3, experimental results are illustrated to prove the ability of DSS. Section 4 concludes this paper.

1 Anatomy of Graph-Based SLAM

To anatomize traditional graph-based SLAM, the algorithm architecture analysis which concludes the weakness of traditional graph-based SLAM is needed. For pre-processing, laser model is also evaluated. It explains how a laser scan is converted from LIDAR frame coordinate to the submap frame coordinate. Moreover, signed distance function (SDF) is applied to improve the mapping precision and the localization accuracy.

1.1 Algorithm Architecture Analysis

The architecture of a graph-based SLAM contains two main parts: the front-end and the back-end[10]. The architecture is shown in Fig. 1. The front-end uses scan matching for short term data association and loop closing detection for long term data association. Both of them are the processes of aligning the new laser scan with previous scans or the global map by iterative calculation[11-12]. The back-end processes maximum a posteriori (MAP) estimation by extracting the result of the front-end to update a pose graph which consists of nodes and edges[13]. Each node represents a robot's pose or an observation's pose, and each edge represents the constraint generated from observation or translation. According to pose graph, global optimization is conducted to reduce accumulated error.

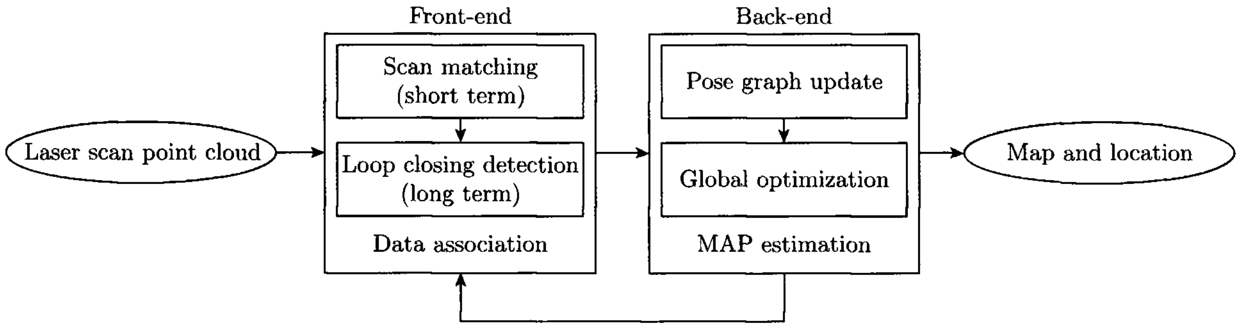


Fig. 1 Graph-based SLAM architecture[10]

Though graph-based SLAM algorithms have been successfully demonstrated in short term applications, they must operate for an extended period of time in many applications. These applications include environmental monitoring, non-stop cleaning robots, or Mars exploration. For such applications, with the movement of the robot, the size of the pose graph can grow unbounded, due to the high data overlap rate between adjacent scans. In practice, memory usage grows linearly as the pose graph has to store observed scans for loop closing detection. Too much memory usage is wasted to store those overlapped data. And overlapped data are computed too many times during loop closing detection which makes real time performance not well. The situation is worse when re-visiting a place multiple times such as circling. Pose graph becomes less efficient as nodes and edges are continuously added with same data, compromising the sparsity structure of the graph.

1.2 Laser Model Evaluation

In front-end, the laser scan has to be converted to submap frame coordinate. Assuming that the current position of the robot relative to the start of the submap is noted as a 1×3 matrix $\boldsymbol{P}_t = [x_t \quad y_t \quad \theta_t]$ at time t. Given a laser scan $\boldsymbol{D}' = [\boldsymbol{d}'_1 \quad \boldsymbol{d}'_2 \quad \cdots \quad \boldsymbol{d}'_I]$ that provides with I scan endpoints $\boldsymbol{d}'_i = (d_i^{x'}, d_i^{y'})$ in LIDAR frame coordinate. The endpoint $\boldsymbol{d}'_i$ can be converted from LIDAR frame coordinate to the submap frame coordinate:

$$\boldsymbol{d}_i = \boldsymbol{d}'_i \otimes \boldsymbol{P}_t = \begin{bmatrix} \cos\theta_t & -\sin\theta_t \\ \sin\theta_t & \cos\theta_t \end{bmatrix} \begin{bmatrix} d_i^{x'} \\ d_i^{y'} \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \end{bmatrix}. \quad (1)$$

The resulting matrix of scan endpoints is denoted as a point cloud $\boldsymbol{D} = [\boldsymbol{d}_1 \quad \boldsymbol{d}_2 \quad \cdots \boldsymbol{d}_I]$ that provides with I scan endpoints $\boldsymbol{d}_i = (d_i^x, d_i^y)$ in submap frame coordinate.

1.3 Mapping by SDF

The point cloud $\boldsymbol{D}$ is used to update the last submap or build a new submap. Different from traditional graph-based SLAM using probability grid map[14], SDF[15] is applied here for mapping. SDF values are positive in-front of the surface, and negative behind. The surface interface is defined by the zero-crossing where the values change sign. Unlike common method stores SDF value and its weight in cell, the graph-based SLAM with DSS stores them in vertex.

Whenever a point cloud $\boldsymbol{D}$ is to be inserted into the map, the vertexes of the ray traced or extended-ray traced cells which are within R distance of the endpoint $\boldsymbol{d}_i$ are to be updated, illustrated in Fig. 2. The vertexes of the ray traced cells are updated with the positive SDF, and the vertexes of the extended-ray traced cells are updated with the negative SDF. Thus, the surface where the SDF value equals to 0 can be found by linear interpolation. Euclidean distance between vertex $\boldsymbol{m}_j$ and the endpoint $\boldsymbol{d}_i$ is used here, and the SDF is normalized by R:

$$\mathrm{sdf}_j^i = \begin{cases} + \left|\boldsymbol{m}_j\boldsymbol{d}_i\right|/R, & \left|\boldsymbol{m}_j\boldsymbol{P}_t\right| < \left|\boldsymbol{d}_i\boldsymbol{P}_t\right| \\ - \left|\boldsymbol{m}_j\boldsymbol{d}_i\right|/R, & \text{else} \end{cases}. \tag{2}$$

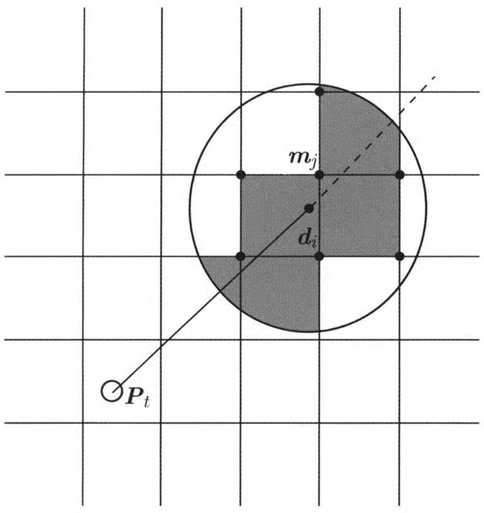


Fig. 2 Vertex to be updated

After all endpoints are processed, the average SDF value $\mathrm{sdf}_j^{\mathrm{avg}}$ is calculated by all SDF values of the vertex $\boldsymbol{m}_j$. The count of the SDF calculations of the vertexes $\boldsymbol{m}_j$ is denoted as n. Then, the weight of the sdf value is calculated as

$$w_j^t = \min\{\text{maximum weight}, w_j^{t-1} + n\}. \tag{3}$$

Finally, the SDF of $\boldsymbol{m}_j$ is computed as

$$\mathrm{sdf}_j^t = \frac{\mathrm{sdf}_j^{t-1} w_j^{t-1} + \mathrm{sdf}_j^{\mathrm{avg}} w_j^t}{w_j^{t-1} + w_j^t}. \tag{4}$$

2 Graph-Based SLAM with DSS

The graph-based SLAM with DSS uses submap-based scan matching in the front-end, a new submap will be built only when the result is too far, and otherwise the scan is used to update the last submap. After that, loop closing detection which uses a rough matching by lookup table and a precise matching by LM method is also applied. The back-end processes the data produced from the front-end to update a pose graph. The pose graph is a graphical model that encodes the dependence between the relative pose and the absolute pose. According to the pose graph, global optimization and submap merging are conducted.

2.1 Submap Based Scan Matching

Scan matching is the process of finding the robot pose $\boldsymbol{P}_t$ by matching the new laser scan. To combine scans into submaps, the submap based scan matching is presented here. It matches the laser scan to the last submap to calculate the robot pose $\boldsymbol{P}_t$ which equals to relative pose between the laser scan and the submap.

$\boldsymbol{P}_t$ is calculated by the non-linear least squares error problem which minimizes

$$\boldsymbol{P}_t = \arg\min_{\boldsymbol{P}} f(\boldsymbol{P}) = \arg\min_{\boldsymbol{P}} \sum_{i=1}^{I} \left[S\left(\boldsymbol{d}'_i \otimes \boldsymbol{P}\right)\right]^2 = \arg\min_{\boldsymbol{P}} \sum_{i=1}^{I} \left[S(\boldsymbol{d}_i)\right]^2, \tag{5}$$

where, $S(\boldsymbol{d}_i)$ returns the SDF value at the submap frame coordinate $\boldsymbol{d}_i$; $\boldsymbol{d}_i$ is calculated by endpoint $\boldsymbol{d}'_i$ and robot pose $\boldsymbol{P}$ as Eq. (1). As SDF value always models continuously, it can be calculated as Eq. (6) by bilinear interpolation. The equation uses the four vertexes of the cell where the endpoint $\boldsymbol{d}_i$ is at, depicted in Fig. 3.

$$S(\boldsymbol{d}_i) = \frac{y - y_0}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}\mathrm{sdf}_{11}^t + \frac{x_1 - x}{x_1 - x_0}\mathrm{sdf}_{01}^t\right) + \frac{y_1 - y}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}\mathrm{sdf}_{10}^t + \frac{x_1 - x}{x_1 - x_0}\mathrm{sdf}_{00}^t\right). \tag{6}$$

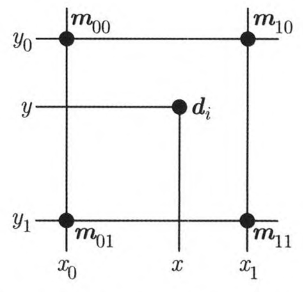


Fig. 3 Bilinear interpolation

According to the LM method, the parameter $\boldsymbol{P}$ is replaced by a new estimate $\boldsymbol{P} + \Delta\boldsymbol{P}$ in each iteration step:

$$f(\boldsymbol{P} + \Delta\boldsymbol{P}) = \sum_{i=1}^{I} \left[S(\boldsymbol{d}'_i \otimes (\boldsymbol{P} + \Delta\boldsymbol{P}))\right]^2. \tag{7}$$

To determine $\Delta\boldsymbol{P}$, the function $S(\boldsymbol{d}_i'\otimes(\boldsymbol{P}+\Delta\boldsymbol{P}))$ is approximated by its linearization according to its first order Taylor expansion:

$$S(\boldsymbol{d}_i'\otimes\boldsymbol{P}) - \nabla S(\boldsymbol{d}_i'\otimes\boldsymbol{P})\frac{\partial(\boldsymbol{d}_i'\otimes\boldsymbol{P})}{\partial\boldsymbol{P}}\Delta\boldsymbol{P} = S(\boldsymbol{d}_i) - \nabla S(\boldsymbol{d}_i)\frac{\partial\boldsymbol{d}_i}{\partial\boldsymbol{P}}\Delta\boldsymbol{P} = S(\boldsymbol{d}_i) - \boldsymbol{J}_{\boldsymbol{d}_i}\Delta\boldsymbol{P}, \quad (8)$$

where, $\boldsymbol{J}_{\boldsymbol{d}_i}$ is the Jacobian of the function $S(\boldsymbol{d}_i)$; $\frac{\partial\boldsymbol{d}_i}{\partial\boldsymbol{P}}$ is evaluated by Eq. (1):

$$\frac{\partial\boldsymbol{d}_i}{\partial\boldsymbol{P}} = \frac{\partial(\boldsymbol{d}_i'\otimes\boldsymbol{P})}{\partial\boldsymbol{P}} = \begin{bmatrix} 1 & 0 & -\sin\theta d_i^{x'} - \cos\theta d_i^{y'} \\ 0 & 1 & \cos\theta d_i^{x'} - \sin\theta d_i^{y'} \end{bmatrix}, \quad (9)$$

and the gradient of the SDF value can be approximated by

$$\nabla S(\boldsymbol{d}_i) = \frac{\partial S(\boldsymbol{d}_i)}{\partial x} + \frac{\partial S(\boldsymbol{d}_i)}{\partial y}, \quad (10)$$

$$\frac{\partial S(\boldsymbol{d}_i)}{\partial x} = \frac{y-y_0}{y_1-y_0}(\mathrm{sdf}_{11}^t - \mathrm{sdf}_{01}^t) + \frac{y_1-y}{y_1-y_0}(\mathrm{sdf}_{10}^t - \mathrm{sdf}_{00}^t), \quad (11)$$

$$\frac{\partial S(\boldsymbol{d}_i)}{\partial y} = \frac{x-x_0}{x_1-x_0}(\mathrm{sdf}_{11}^t - \mathrm{sdf}_{10}^t) + \frac{x_1-x}{x_1-x_0}(\mathrm{sdf}_{01}^t - \mathrm{sdf}_{00}^t). \quad (12)$$

Thus, Eq. (7) can be approximated in vector notation as

$$f(\boldsymbol{P}+\Delta\boldsymbol{P}) \approx \|\boldsymbol{V}(\boldsymbol{P}) - \boldsymbol{J}\Delta\boldsymbol{P}\|^2 = \boldsymbol{V}^{\mathrm{T}}(\boldsymbol{P})\boldsymbol{V}(\boldsymbol{P}) - 2\boldsymbol{V}(\boldsymbol{P})\boldsymbol{J}\Delta\boldsymbol{P} + \Delta\boldsymbol{P}^{\mathrm{T}}\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}\Delta\boldsymbol{P}, \quad (13)$$

where, $\boldsymbol{J}$ is the Jacobian matrix, whose i-th row equals $\boldsymbol{J}_{\boldsymbol{d}_i}$; $\boldsymbol{V}(\boldsymbol{P})$ is the vector whose i-th component is $S(\boldsymbol{d}_i)$.

Finally, taking Eq. (13) with respect to $\Delta\boldsymbol{P}$ in a damped version and setting the result to zero gives:

$$\Delta\boldsymbol{P} = (\boldsymbol{H} + \mu\boldsymbol{I})^{-1}\boldsymbol{J}^{\mathrm{T}}\boldsymbol{V}(\boldsymbol{P}), \quad (14)$$

where $\boldsymbol{H}$ is the Hessian matrix:

$$\boldsymbol{H} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}. \quad (15)$$

The LM method can now be evaluated, yielding a step $\Delta\boldsymbol{P}$ toward the minimum. The parameter $\mu > 0$ is used for step adjustment to ensure the step is in the descent direction. It starts with a small value (10^{-4}). If the new iteration has a lower error than the last iteration, μ is divided by 10. If the error is same or larger, μ is multiplied by 10.

If the result pose is not too far, which means the distance between the new scan and the last submap is within the threshold K, the new scan is used to update the last submap. Otherwise, a new submap is built by the new scan. Therefore, submap has the complete data of all contained scans but only saves the overlapped data once. And all submaps are distributed, as adjacent submaps are at least K distance apart.

2.2 Loop Closing Detection

After adding the new scan, loop closing detection is conducted by 4 steps. Firstly, remove all the adjacent submaps form the candidate aggregate C which contains previous submaps. Secondly, calculate the Euclidean distance from the new scan to submaps in C, and then remove the submaps whose Euclidean distance is larger than the threshold. After that, a rough matching using lookup table is processed to reduce the candidate number. Finally, a precise matching based on LM method is processed.

The lookup table forms a searching window W around the estimated relative pose $\boldsymbol{P}_0$:

$$\left.\begin{aligned} &W = \{\boldsymbol{P}_0 \pm (i_x\mathrm{step}_x, i_y\mathrm{step}_y, i_\theta\mathrm{step}_\theta)\} \\ &i_x \leqslant \frac{W_x}{\mathrm{step}_x},\ i_y \leqslant \frac{W_y}{\mathrm{step}_y},\ i_\theta \leqslant \frac{W_\theta}{\mathrm{step}_\theta} \end{aligned}\right\}, \quad (16)$$

where, W_x and W_y are set to 2 m which equals to half of the LIDAR detection range; W_θ is set to 30°. To prevent scan endpoint from moving more than the low resolution cell width, step_x is set to the low resolution cell width r as well as step_y, and step_θ is calculated as

$$\mathrm{step}_\theta = \arccos\left(1 - \frac{r^2}{2d^2}\right), \quad (17)$$

where d equals to the largest range within all scan endpoints. Then, match a specific submap, and a score of every pose is calculated as

$$\mathrm{Score}(\boldsymbol{P}_i) = \sum_{i=1}^{I}\{1 - [S(\boldsymbol{d}_i\otimes\boldsymbol{P}_i)]^2\}, \quad \boldsymbol{P}_i \in W. \quad (18)$$

If the score is smaller than the threshold which is set as 0.8 times of maximum Score($\boldsymbol{P}_i$), the submap will be deleted from the candidate aggregate C. The maximum Score($\boldsymbol{P}_i$) equals to the count of scan endpoints I.

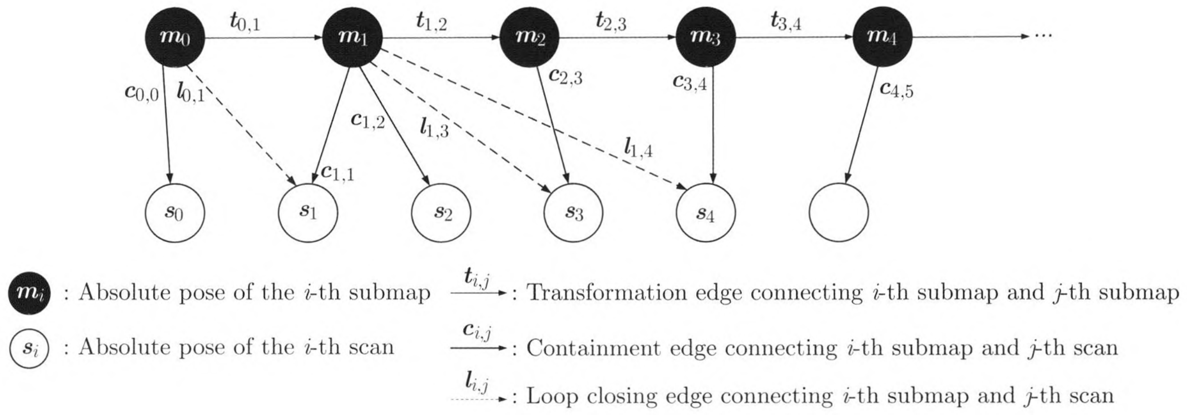
Next, a precise matching using LM method as Eq. (14) is used to discover the only loop closing which has the lowest error function as Eq. (5) and the highest matching percentage. The matching percentage P_{m} indicates how much the submap corresponds:

$$P_{\mathrm{m}} = \frac{n_0}{I}, \quad (19)$$

where n_0 is the count of endpoints whose SDF value is 0 after convertion by $\boldsymbol{P}_i$. Finally, a loop closing edge is added to pose graph. Because overlapped data of scans within the submap will be saved only once, those data will also be calculated only once during loop closing detection. Real time performance is improved obviously.

2.3 Pose Graph Update

Pose graph is a collection of nodes and edges. Nodes represent the absolute pose of submaps or scans, and edges which are also called constraints represent the relative pose between submaps and scans. Figure 4 shows an example of pose graph. There are three main edges: transformation edge which represents the relative pose between two adjacent submap nodes shows the movement of the robot; containment edge which represents the relative pose between one submap and one scan means that the scan is contained by the submap; loop closing edge shows that the loop closing detection succeeds between the two nodes.



m_0
$t_{0,1}$
m_1
$t_{1,2}$
m_2
$t_{2,3}$
m_3
$t_{3,4}$
m_4
...
$c_{0,0}$
$l_{0,1}$
$c_{1,2}$
$l_{1,3}$
$c_{1,1}$
$c_{2,3}$
$l_{1,4}$
$c_{3,4}$
$c_{4,5}$
s_0
s_1
s_2
s_3
s_4
m_i : Absolute pose of the i-th submap
s_i : Absolute pose of the i-th scan
$t_{i,j}$: Transformation edge connecting i-th submap and j-th submap
$c_{i,j}$: Containment edge connecting i-th submap and j-th scan
$l_{i,j}$: Loop closing edge connecting i-th submap and j-th scan

Fig. 4 Pose graph

After the scan matching and the loop closing detection for the new scan, the back-end gets the relative pose between the scan and the last submap. A scan node and a containment edge will be added. If a new submap is built, a submap node and a transformation edge will be added. And if the loop closing detection succeeds, a loop closing edge will also be added.

Because of the accumulated error or false loop closing detection, an edge quality check is needed. Edge quality check will remove those low quality edges, and it can reduce time consumption remarkably. Edge quality check $Q_{i,j}$ is computed as Eq. (20). It shows the relative error of the edge and the two nodes. Those edges will be removed if their quality $Q_{i,j}$ is under the threshold.

$$Q_{i,j} = \frac{\|\boldsymbol{r}_{i,j}\|_2 - \|\boldsymbol{p}_i - \boldsymbol{p}_j\|_2}{\|\boldsymbol{r}_{i,j}\|_2}, \tag{20}$$

where $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are the poses of the nodes linked by the edge $\boldsymbol{r}_{i,j}$.

2.4 Global Optimization and Submap Merging

Once a loop closing edge is added, the global optimization based on the pose graph will be conducted. It forms as finding the minimum of the error function:

$$\boldsymbol{p}^* = \arg\min_{\boldsymbol{p}} \sum_{\langle \boldsymbol{p}_i, \boldsymbol{p}_j \rangle \in \boldsymbol{p}} \boldsymbol{e}^{\mathrm{T}}(\boldsymbol{p}_i, \boldsymbol{p}_j, \boldsymbol{r}_{i,j}) \times Q_{i,j} \boldsymbol{e}(\boldsymbol{p}_i, \boldsymbol{p}_j, \boldsymbol{r}_{i,j}), \tag{21}$$

$$\boldsymbol{e}(\boldsymbol{p}_i, \boldsymbol{p}_j, \boldsymbol{r}_{i,j}) = \boldsymbol{r}_{i,j} - \begin{bmatrix} \boldsymbol{R}_i^{-1}(\boldsymbol{T}_i - \boldsymbol{T}_j) \\ \theta_i - \theta_j \end{bmatrix}, \tag{22}$$

where $\boldsymbol{p} = [\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_N]$ is the matrix of poses of all nodes. The global optimization is solved by general graph optimization (g2o)[16].

The pose graph can not only be used for global optimization, but also helps to merge submaps into the global map. The global map is updated by sequential submap merging.

The process of submap merging is shown in Fig. 5, and the submap is merged by its location and the loop closing edges which link the scans to previous submaps. Since loop closing edges' values are relative variables, the first submap's location has to be fixed as the base point of the global map.

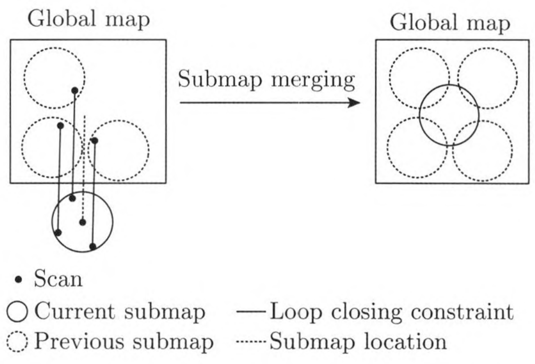


Fig. 5 An illustration of submap merging

3 Experimental Result

In order to verify the ability of graph-based SLAM with DSS, some experimental results are presented.

The experiment dataset is collected from a physical robot teleoperated by the PC in a real world scenario. The robot shown in Fig. 6 carries a RobotPeak LIDAR RPLIDAR A2, a Uranus MEMS magnetometer HI216M and a Raspberry Pi 3B. The specification of sensors is provided in Table 1. To evaluate the improvement of graph-based SLAM with DSS, the dataset was designed to include many different environmental challenges. It was captured by driving the robot around an office environment totalling 60 m of distance travelled, and arriving precisely back in the starting position. The dataset contains several large loops and several areas with few obstacles.

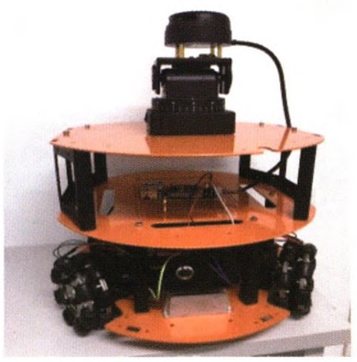


Fig. 6 Physical robot

Table 1 Specification of LIDAR and MEMS magnetometer

Sensor	Detection range	Sample rate	Distance resolution/mm	Angle resolution/(°)	Field of view/(°)
LIDAR	4 m	4 000 Sa/s	0.5	0.45	360
Magnetometer	±4 800 μT	100 Hz	–	0.01	–

According to the dataset, three experiments are conducted. The first experiment shows the reduction in data overlap rate which is the main contribution of DSS. Benefited from it, the improvement in real time performance and memory usage is also performed. The final experiment evaluates the accuracy and robustness of DSS.

3.1 Comparison of Data Overlap Rate

In this section, the data overlap rate of adjacent submap pairs with DSS and adjacent scan pairs without DSS is calculated. It is noted that different threshold K which is proposed in Section 2.1 is used. The data overlap rate is calculated same as the matching percentage P_m which is proposed as Eq. (19). The result of the comparison of data overlap rate is shown in Table 2 ($\bar{x}$ is the mean value, $\tilde{x}$ is the median value, σ is the variance).

Table 2 Comparison of data overlap rate

Algorithm	$\bar{x}$	$\tilde{x}$	σ
Without DSS	0.842 545	0.855 556	0.022 423
With DSS ($K = 4$ m)	0.406 155	0.453 370	0.045 069
With DSS ($K = 8$ m)	0.311 361	0.387 097	0.047 921

The maximum data overlap rate is set to 1, which means the two adjacent scans or submaps are absolutely same. The minimum is set to 0, which means there is no overlapped data. It is shown that graph-based SLAM with DSS performs a huge improvement in reducing data overlap rate. DSS with $K = 4$ m reduces 51.79% data overlap rate in average, and DSS with $K = 8$ m reduces 63.05% data overlap rate in average. The result also reveals that larger K leads to a lower data overlap rate, as larger K illustrates that the distance between two adjacent submaps is larger, thus the overlap area occupies a smaller proportion. Moreover, graph-based SLAM with DSS shows a larger variance, as the number of scans used to update submap is random which means the coverage area of every submap is random. Thus, the variance is influenced by robot route, and straight route leads to a small variance, while irregular route such as circling leads to a large variance. This is also the reason why larger threshold K causes a larger variance.

3.2 Analysis of Real Time Performance and Memory Usage

Beyond the comparison of data overlap rate, the analysis of real time performance and memory usage is needed as it proves the efficiency of DSS. Thus, real time performance and memory usage experiments are carried out. To prove the advantage of DSS over other existing low data overlap rate methods, linear SLAM[9] and cartographer[13] are also compared in this experiment.

The run time experiment result is shown in Fig. 7 and Table 3. With the increment of the laser scan count which is called iteration index, all three algorithms spend longer time in average. And those jitters

Table 3 Comparison of real time performance

Algorithm	Run time/ms	Reduction/%
With DSS ($K = 4$ m)	32 984.494	–
With DSS ($K = 8$ m)	23 468.353	−40.55
Without DSS	54 697.936	39.70
Linear SLAM	38 077.327	13.37
Cartographer	41 997.064	21.46

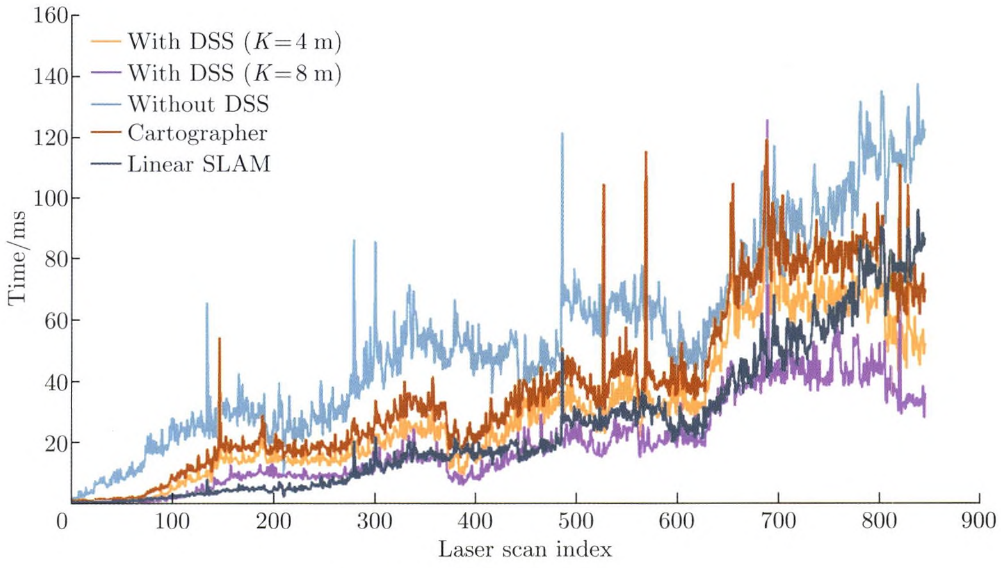


Fig. 7 Real time performance

are caused by bad initial pose matrix. Compared to graph-based SLAM without DSS, DSS with $K = 4\,\text{m}$ decreases 39.70% runtime. Because graph-based SLAM without DSS stores scans with too much overlapped data. However, graph-based SLAM with DSS saves those scans into a small amount of submaps, saving time by computing those overlapped data only once. It is showed that larger threshold K leads to a better real time performance, as larger K makes less submaps built. However, it is bad for global optimization and construction of global map, as those submaps are not accurate enough. And as can be seen, DSS has a 21.46% lower than cartographer in run time, and a 13.37% lower than linear SLAM. Because DSS uses edge quality check to remove those low quality edges, the time consumption of global optimization is effectively decreased. In the linear SLAM, it shows the best real time performance at the beginning of the experiment, as it processes those subgraphs parallel. But with the moving of the robot, time consumption grows rapidly due to larger subgraph size and bigger subgraph count.

Similar to the real time performance, graph-based SLAM with DSS is proved to be efficient for decreasing memory usage. Notably, there is 22.13 MB memory used for the initialization, and the result is shown in Fig. 8 and Table 4.

DSS with $K = 4\,\text{m}$ decreases 24.60% memory usage. Because graph-based SLAM without DSS stores scans which contain too much overlapped data. What's more, its speed of growth is linear, as of course scans share similar memory usage. However, graph-based SLAM with DSS's memory usage only significantly grows when a new submap is built. It also explains why larger threshold K leads to a smaller memory usage, as a new submap will only be built when the distance is larger than K. And DSS also has a 26.50% less than linear SLAM in memory usage, and a 15.45% more than cartographer. It is clear that linear SLAM costs the most memory as it still saves all scan data. Thus, its memory usage performance is similar to graph-based SLAM without DSS or even worse, because it also has to save subgraph structure information. By combining scans into submaps, both DSS and cartographer perform well in memory usage. DSS costs more memory as SDF map has to save two values in every vertex: SDF value and its weight, while cartographer only needs to save occupancy probability in every cell.

Table 4 Comparison of memory usage

Algorithm	Memory usage/MB	Reduction/%
With DSS ($K = 4\,\text{m}$)	28.238	–
With DSS ($K = 8\,\text{m}$)	25.453	−10.94
Without DSS	37.453	24.60
Linear SLAM	38.422	26.50
Cartographer	24.459	−15.45

3.3 Evaluation of Accuracy and Robustness

In order to evaluate accuracy and robustness, the final maps obtained by traditional graph-based SLAM, DSS, linear SLAM and cartographer are showed in Fig. 9.

As can be seen, traditional graph-based SLAM has the greatest performance in accuracy and robustness. In exchange, it costs huge run time and memory usage. It is noted that linear SLAM obtained the lowest quality of final map. Because subgraph loses too much information in large scale environment, it is easy to lead a failure optimization. The same weakness of DSS and cartographer is that they have no method to

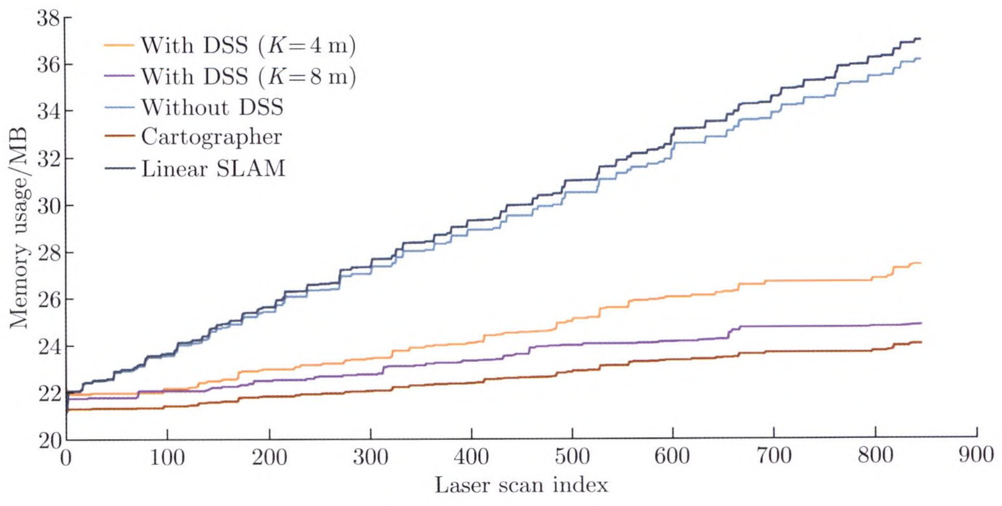


Fig. 8 Memory usage

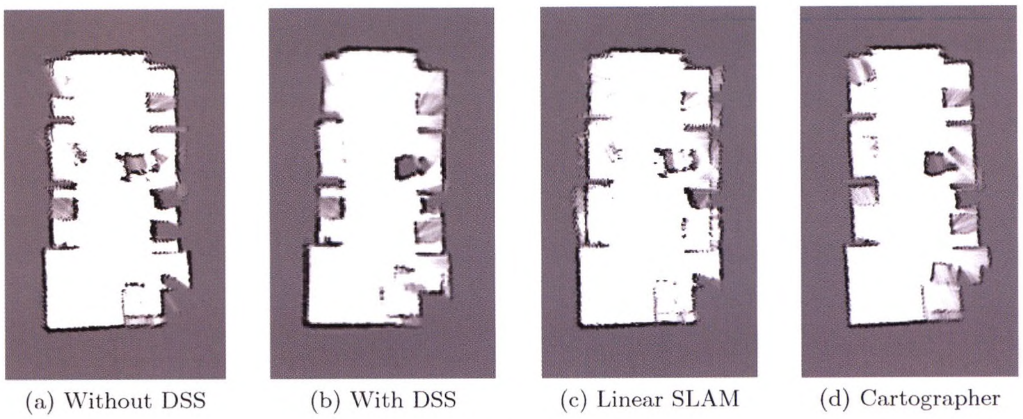


Fig. 9 Final maps

decrease accumulated error of scan's pose in the submap, as global optimization can only correct submap's pose in global map. Besides, DSS is more accurate and robust than cartographer. It is mainly benefited from using SDF map. SDF map has two main advantages over probability grid map: one is that SDF map is possible to recover the surface at a precision above the minimum voxel size as it always models continuously; the other is that SDF map stores two values of each voxel, SDF value and its weight, thus the calculation error is smaller.

4 Conclusion

To decrease data overlap rate, graph-based SLAM with DSS is presented. Firstly, scans are used to build distributed submaps after submap based scan matching in front-end. By building submaps, memory usage is obviously reduced as those overlapped data are merged together. In loop closing detection, overlapped data will not be repeatedly calculated which is helpful to improve real time performance. After that in back-end, the new scan or submap is added to the pose graph. Global optimization and submap merging are conducted according to pose graph. From experimental results, graph-based SLAM with DSS is proved to be effective and practicable for reducing data overlap rate. It is also efficient in runtime, memory usage, accuracy and robustness performance. Further work will explore 3D SLAM with DSS in an even more efficient way.

References

[1] CARLONE L, ARAGUES R, CASTELLANOS J A, et al. A fast and accurate approximation for planar pose graph optimization [J]. *The International Journal of Robotics Research*, 2014, **33**(7): 965-987.

[2] MUTZ F, VERONESE L P, OLIVEIRA-SANTOS T, et al. Large-scale mapping in complex field scenarios using an autonomous car [J]. *Expert Systems With Applications*, 2016, **46**: 439-462.

[3] SAEEDI S, NARDI L, JOHNS E, et al. Application-oriented design space exploration for SLAM algorithms [C]//*IEEE International Conference on Robotics and*

Automation. Singapore, Singapore: IEEE, 2017: 5716-5723.

[4] SANTOS J M, PORTUGAL D, ROCHA R P. An evaluation of 2D SLAM techniques available in Robot Operating System [C]//*IEEE International Symposium on Safety, Security, and Rescue Robotics*. Linkoping, Sweden: IEEE, 2013: 1-6.

[5] NIU X J, YU T, TANG J, et al. An online solution of LiDAR scan matching aided inertial navigation system for indoor mobile mapping [J]. *Mobile Information Systems*, 2017, **2017**: 4802159.

[6] LENAC K, KITANOV A, CUPEC R, et al. Fast planar surface 3D SLAM using LIDAR [J]. *Robotics and Autonomous Systems*, 2017, **92**: 197-220.

[7] MAZURAN M, BURGARD W, TIPALDI G D. Nonlinear factor recovery for long-term SLAM [J]. *The International Journal of Robotics Research*, 2016, **35**(1/2/3): 50-72.

[8] DUBE R, SOMMER H, GAWEL A, et al. Non-uniform sampling strategies for continuous correction based trajectory estimation [C]//*IEEE International Conference on Robotics and Automation*. Stockholm, Sweden: IEEE, 2016: 4792-4798.

[9] ZHAO L, HUANG S D, DISSANAYAKE G. Linear SLAM: A linear solution to the feature-based and pose graph SLAM based on submap joining [C]//*IEEE/RSJ International Conference on Intelligent Robots and Systems*. Tokyo, Japan: IEEE, 2013: 24-30.

[10] CADENA C, CARLONE L, CARRILLO H, et al. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age [J]. *IEEE Transactions on Robotics*, 2016, **32**(6): 1309-1332.

[11] AGARWAL S, SHREE V, CHAKRAVORTY S. RFM-SLAM: Exploiting relative feature measurements to separate orientation and position estimation in SLAM [C]//*IEEE International Conference on Robotics and Automation*. Singapore, Singapore: IEEE, 2017: 6307-6314.

[12] LEHTOLA V V, VIRTANEN J P, VAAJA M T, et al. Localization of a mobile laser scanner via dimensional reduction [J]. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2016, **121**: 48-59.

[13] HESS W, KOHLER D, RAPP H, et al. Real-time loop closure in 2D LIDAR SLAM [C]//*IEEE International Conference on Robotics and Automation*. Stockholm, Sweden: IEEE, 2016: 1271-1278.

[14] KOHLBRECHER S, VON STRYK O, MEYER J, et al. A flexible and scalable SLAM system with full 3D motion estimation [C]//*IEEE International Symposium on Safety, Security, and Rescue Robotics*. Kyoto, Japan: IEEE, 2011: 155-160.

[15] FOSSEL J D, TUYLS K, STURM J. 2D-SDF-SLAM: A signed distance function based SLAM frontend for laser scanners [C]//*IEEE/RSJ International Conference on Intelligent Robots and Systems*. Hamburg, Germany: IEEE, 2015: 1949-1955.

[16] KÜMMERLE R, GRISETTI G, STRASDAT H, et al. G2o: A general framework for graph optimization [C]//*IEEE International Conference on Robotics and Automation*. Shanghai, China: IEEE, 2011: 3607-3613.