Proceeding of the IEEE
International Conference on Robotics and Biomimetics
Dali, China, December 2019

A Triangle Feature Based Map-to-map Matching and Loop Closure for 2D Graph SLAM

Binqian Jiang[1], Yilong Zhu[1], Ming Liu[1], *Senior Member, IEEE*

***Abstract*—The loop closure problem in 2D LiDAR simultaneous localization and mapping (SLAM) is interesting yet to be solved efficiently, as it suffers from a lack of information in 2D laser rangefinder readings. For this reason, we compare two grid submaps, where features and constraints are enriched, to find loops. We propose a geometric environment descriptor called a Triangle Feature (TF). It exploits the Euclidean distance constraint any three feature points in a submap can form. A 2D graph SLAM system using TF to close loops is also developed. The global maps we build outperform some popular open-source SLAM solutions, and the system can run up to 2.4 times of real-time in our experiments.**

***Index Terms*—graph SLAM, loop closure, grid map matching, LiDAR**

I. Introduction

Simultaneous localization and mapping (SLAM) is fundamental to autonomous mobile robots. A key feature distinguishes SLAM from odometry is detecting previously visited areas to reduce mapping errors, a process known as loop closure detection. If we sacrifice loop closures, SLAM reduces to odometry [1]. A 2D LiDAR is suitable for mapping and localization in structured environments, like indoor scenarios, and the sensor's cost and measurement accuracy reach a good balance compared to vision sensors or 3D LiDARs. However, due to the sparsity of 2D LiDAR beams, loop closure purely using range measurements becomes a problem to be solved efficiently.

To overcome the above problem, we propose a triangle feature (TF) based submap-to-submap matching and loop closure detection method. We treat local probability grid maps [2] as grayscale images, and corner points are extracted to build TFs. A set of TFs describes the geometry character of the environment, and matching two sets of TFs from two submaps can be used to detect loop closure. The main idea of the TF matching is that, by constructing triangles, we obtain more descriptors than the plain corner points. If a loop exists, there is a rigid transform between the two submaps. Every pair of congruent triangles from the two TF sets will give an estimate of the transform parameter, while non-congruent pairs will be trimmed. And finally, we use transform parameters from congruent pairs to vote to see if a salient transform exists. In short, this method uses a combination number to proliferate a manageable number of features and uses simple trimming and voting to reject wrong feature pairs while keeping possible correct ones intact.

In our experiments, the quality of the maps we build outperforms some popular open-source solutions using the same input data, and the system can still run at 2.4 times of real-time with global loop detection and optimization enabled.

II. Related work

While the problem of 3D LiDAR SLAM has been widely researched in the past few years, the low-cost 2D LiDAR makes 2D SLAM a good choice in scenarios like indoor environments. Representative works include gmapping [3], hector SLAM [4], cartographer [5]. And we will focus on the problem of loop closure in these 2D SLAMs.

Many 2D SLAM methods don't use loop closure at all, like the famous open-source hector-SLAM [4] and gmapping [3], as a result, drift is significant when building large maps. Still, we find two main categories of methods in existing works concerning 2D loop closure. The first uses an indirect approach, representative researches include vision-based loop closure solution [6]. It avoids the intrinsic limitation of scan range data (too little information per frame), by using extra visual information to decide if the agent is revisiting a place.

The second category uses 2D scan data or map grid data directly. Representative works include Google's cartographer [5], which uses correlative scan matching (CSM) and relies on branch-and-bound to accelerate scan-to-submap matching search speed. However, its Ceres scan matcher performs poorly given a bad initial value, and a coarse result from CSM when building submaps in local SLAM may be troublesome. Plus, the computationally expensive direct scan-to-map matching is only partly avoided.

There are scan feature-based approaches. Fast Laser Interest Region Transform (FLIRT) [7], which uses a curvature-based scan detector and β-grid descriptor to find similarity between scans. This method requires a high overlap ratio of 2 scans, and sparsity will cause a sharply decreased accuracy. Another similarity-based scan-scan graph SLAM [8] uses neighboring scan endpoints to compute scan descriptors.

In a map-to-map based method [9], the author treats the probability grid map as an image and borrows detectors from computer vision to find feature points and match with his descriptor matrix. Different image descriptors are compared in this article, but the misclassification rate of the descriptors is high and a modified RANSAC is needed to obtain satisfactory results.

[1]All authors are with the Robotics and Multi-Perception Laboratory in Robotics Institute at the Hong Kong University of Science and Technology, Hong Kong, China. {bjiangah, yzhubr}@connect.ust.hk eelium@ust.hk

Other related methods include using geometric landmarks of the environment for loop closure detection [10], which transforms 2D laser scans into pose invariant histogram representations, using Euclidean distance and relative orientation as parameters, which is similar to our method. The submap-level loop closure problem also has some connection with the field of grid map registration and merging as well, which studies the similar map-to-map matching problem [11].

III. System overview

Our 2D SLAM solution uses LiDAR data as a primary source of input, and an inertial measurement unit (IMU) is used to provide pose estimation prior. No odometry is used in our current design. And the whole SLAM system consists of a frontend and a backend. We choose the occupancy probability grid map, similar to that in Google's cartographer [5], to represent the environment.

The frontend mainly performs scan matching and scan registering in the local grid map, or *submap*, to distinguish from the overall grid map, or *global map*. The goal of our frontend is to build a submap with enough features and the error accumulated in it is below the grid resolution.

In the backend, we use pose graph optimization to deal with the problem of error accumulation when the number of submaps grows larger. The vertices of the pose graph are the estimated global poses of the submaps (estimated every time a new submap is generated), also known as pose node, and the edges are the measurement of the two vertices it links (which is measured using our map-to-map matching method right after the submap finishes updating). When a submap is finished, no new scans will be inserted, and the TFs (explained in section V-B) will be extracted immediately. Then, the newly extracted features will be compared with all previous submaps' features (explained in Section V), and a confirmed transform between two submaps will cause an edge to form between the two vertices, namely a closed loop.

With enough submaps being recorded, we will periodically perform a graph optimization for all the vertices and edges available. The optimization results will be used to update the measurement and estimation of the pose nodes. A global map can be generated any time by registering the submaps with their optimized poses into one large grid map.

IV. Local 2D SLAM

Local 2D SLAM aims at finding the best 3 degree-of-freedom (DOF) pose estimate $\xi = (\xi_x, \xi_y, \xi_\theta)$, which consists of a translation component (x, y) and a rotation componentξ_θ, of LiDAR observations, or scans, and insert the scans into a local probability grid map. We define the endpoint as a LiDAR beam hitting obstacles as a *scan endpoint*. As is mentioned in section III, our map representation is similar to that of cartographer's [5], but the major difference lies in that we record the discrete subpixel-scaled scan endpoint location in a grid cell if any endpoints ever fall into that cell. We define the scan endpoint in a grid map cell as a *nucleus*.

During the local SLAM, every input scan is matched against the local map M using correlative scan matching (CSM) [12] and point-to-line iterative closest point (PL-ICP) [13]. A good enough scan match in recent several scans will be inserted into the submap, updating the probability value and, if a scan endpoint hits the cell, the estimate of the location of the nucleus.

A. Scans

Scan data in 2D space is substantially a series of points, we write them as $H = \{h_k\}_{k=1,\dots,K}, h_k \in \mathbb{R}^2$, with the origin of the scan at $0 \in \mathbb{R}^2$. Given the estimate of the pose of the scan frame in the submap frame ξ, and the transform matrix T_ξ transforming scan endpoints from scan frame to submap frame can be denoted as

$$T_\xi = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \tag{1}$$

where $\boldsymbol{R} = \begin{bmatrix} \cos\xi_\theta & -\sin\xi_\theta \\ \sin\xi_\theta & \cos\xi_\theta \end{bmatrix}$, and $\boldsymbol{t} = \begin{bmatrix} \xi_x \\ \xi_y \end{bmatrix}$.

B. Submaps

Submaps are discretized representation of the planar environment. Given the grid resolution r and the coordinates of a point (x, y) in the submap frame, its discrete coordinates can be denoted as $\left(\lfloor \frac{x}{r} + 0.5 \rfloor, \lfloor \frac{y}{r} + 0.5 \rfloor\right)$, where $\lfloor x \rfloor$ means round x to the greatest integer less or equal than x.

Every grid cell has 3 components, namely

- Probability value, which represents the probability of the grid being obstructed.
- Bool value, indicates if the grid has a nucleus.
- Discrete subpixel-scaled coordinates of the nucleus in the cell frame, which record the smoother location of the nucleus. We add this field for better PL-ICP and global map registration performance.

A submap is updated when a new scan is inserted. Scan measurements have three possible outcomes. The first is returning a valid scan range, which indicates the obstacle's position and free space between the scan origin and the endpoint. The second is returning maximum scan range, which means no obstacles from the origin all the way to the maximum range. And the third is returning a minimum range, which means the data should be discarded. We call the scan endpoints *hits* and the maximum range *misses*, and hits and misses are mapped to submap cells together with those intersect with the laser rays, which will be updated in this insertion.

We use the sequential updating formulation of Bayes' theorem to incrementally update the probability values of grid cells [2]. A newly observed grid will be assigned the value of the observation directly, otherwise, the new probability will be calculated using Bayes' theorem. We use $odd = p_{hit}/p_{miss}$ to represent the probabilities. For an observed grid, with the new observation as *hit*, the probability of the grid in state s and after the observation $z \in \{0, 1\}$ can be derived as

$$\text{odd}(s|z) = \frac{p(z|s=1)}{p(z|s=0)} \text{odd}(s) \tag{2}$$

where $\mathrm{odd}(s)$ is the prior estimation of the occupancy probability value, and $\frac{p(z|s=1)}{p(z|s=0)}$ is the sensor model, which is considered a constant in our case.

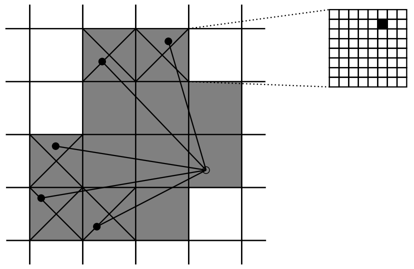


Fig. 1: Demonstration of a scan and pixels with hits (shaded and crossed out) and misses (shaded only). It also shows a cell with a nucleus on the right.

To update the position of the nucleus in a probability grid map cell using a new scan endpoint, we use a Kalman filter [14] to update the subpixel nucleus' position. We assume that the filter has already converged the moment we begin to update. Then the Kalman filter is reduced to a simple weighted average of measurement and prediction. We note that this is suboptimal, but the error will only be in the subpixel level, and the cost of storing and updating the covariance matrix for every "hit" cell is relieved.

C. Scan matching

Every laser scan will go through a scan matching process, either for robot pose estimation or for updating the submap. Scan-to-scan matching is conventionally used for relative pose estimation [15], [16], but it accumulates error quickly, thus we will use scan-to-map matching for robustness. Besides, using solely the optimization-based scan matching methods will easily stick in local minima with a poor initial guess. For this reason, we use a two-stage scan matching method. In the first stage, we use correlative scan matching, which coarsely searches in (x, y, θ) windows to provide an initial guess for the next stage. In the second stage, we use PL-ICP [13] for a delicate scan matching.

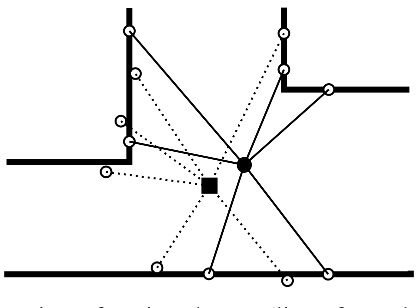


Fig. 2: Illustration of a virtual scan (lines from the black dot) and real scan (dotted lines from the black square). The bold lines represent obstacles in our submap, and hollow circles are scan endpoints.

Maybe it is worth pointing out that PL-ICP is, intrinsically, a scan-to-scan matching method. In our case, we tailored it to meet our need for scan-to-map matching. The specific method is as follows. When a good enough scan is inserted into the submap, we will pick out all the cells updated as *hit*. Then, the nuclei of the hit cells are recorded as virtual scan endpoints (virtual because they come from map instead of a real laser scan), with the next pose estimation as its origin, a virtual laser scan can be encapsulated to match with the real scan data. Fig.2 can be taken for reference.

V. Global SLAM

Since the scan-to-map matching only uses a local map, the error can be significant when the submap grows too large. Thus a larger environment requires many small submaps. In our solution, a pose graph optimization [16] is run at submap level to reduce the accumulated error. And we use a triangle feature-based loop closure detection method to detect loops between two submaps. Any detected loop closures will be added to the pose graph optimization process. Note that due to the sparsity of features, a single scan cannot bring us much information, for this reason, the poses of all the inserted scans are not added to the pose graph optimization.

A. Pose graph optimization

Pose graph optimization can be expressed as a nonlinear least squares problem

$$\begin{aligned} \boldsymbol{x}^* &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \sum_{i,j} \|f(\boldsymbol{x}_{i,j}, \boldsymbol{z}_{i,j})\|^2 \\ f(\boldsymbol{x}_{i,j}, \boldsymbol{z}_{i,j}) &= \boldsymbol{x}_i - \boldsymbol{T}_{i,j}\boldsymbol{x}_j \end{aligned} \tag{3}$$

where $\boldsymbol{x}_{i,j}$ is the paired poses of submaps, $\boldsymbol{x}^*$ is the optimized poses, and $\boldsymbol{T}_{i,j}$ is the transform from $\boldsymbol{x}_j$ to $\boldsymbol{x}_i$, corresponding to a measurement $\boldsymbol{z}_{i,j}$. $f(\boldsymbol{x}_{i,j}, \boldsymbol{z}_{i,j})$ is the error function.

Considering different measurements with different uncertainty estimation, the target function can further be written as

$$\boldsymbol{x}^* = \min_{\boldsymbol{x}} \sum_{k=1}^{n} e_k(x_k, z_k)^T \Omega_k e_k(x_k, z_k) \tag{4}$$

where e_k is the error function, Ω_k is the information matrix, x_k and z_k are the paired estimate and measurement. We use the off-the-shelf g2o [17] to solve the graph optimization problem.

B. Deriving of the triangle feature

The current CSM with the branch-and-bound loop detection method used by cartographer is an optimized exhaustive search and pruning method in nature. The computation cost is still expensive. Besides, a single frame of 2D laser scan provides too little information, so scan-to-scan and scan-to-map methods are not very effective in case of partial overlapping. To overcome the problem of lack of information in laser scans and to boost matching speed, we propose a triangle feature (TF) based submap-to-submap loop detection method.

Our method treats submaps as images. Image registration is extensively studied in the field of computer vision, and mature descriptors like SIFT [18], SURF [19], and ORB [20] have good performance in image matching and registration. However, the grid map differs from images in three major aspects. First, an occupancy grid map is a relatively precise reflection of the real world, all its data comes from range measurement, so scaling is unnecessary in grid map matching. Second, an accurate grid map distinguishes free space and obstacles precisely, which means an abrupt change of value among adjacent pixels (”hard”). Blob image feature descriptors (SIFT, SURF) have better performance with ”softened” images [9] but are outperformed by corner features in ”hard” images. Third, restricted by the size of a submap, the information a grid map can provide is still limited compared to images.

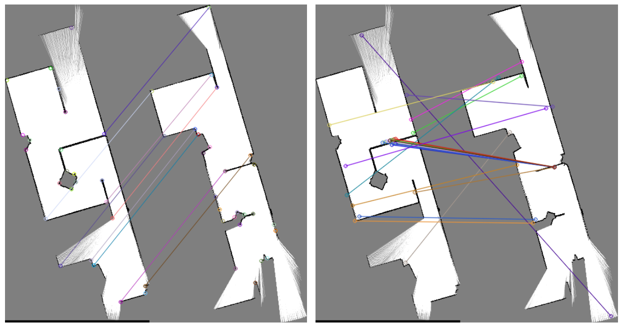

(a) Our method used to match two partially overlapped grid maps. (b) SURF+FLANN, showing 23 most prominent matched pairs.

Fig. 3: Comparison of feature matching result bewteen our overall method and the mature SURF+FLANN [21] matching.

For the reasons above, we extract Shi-Tomasi corner points [22] from grid maps to build triangle features (TF). A *triangle feature* is defined as a sequence of ordered three extracted Shi-Tomasi corner points. When not causing misunderstanding, we also call the triangle constructed from the ordered three points as *triangle feature*. The three points (or vertices of a triangle) $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3} \in \mathbb{R}^2$ are arranged in such an order that

$$\mathbf{L_1} \geq \mathbf{L_2} \geq \mathbf{L_3} \tag{5}$$

where $\mathbf{L_1} = |\boldsymbol{v_1} - \boldsymbol{v_2}|$, $\mathbf{L_2} = |\boldsymbol{v_2} - \boldsymbol{v_3}|$, and $\mathbf{L_3} = |\boldsymbol{v_3} - \boldsymbol{v_1}|$.

By looking at the definition, it's easy to discern that TFs mainly exploit the relative position of the geometric features of the environment. With adequate TFs the environment can be represented precisely, and few local inconsistencies in a submap will not affect its matching with another submap recorded at the same place.

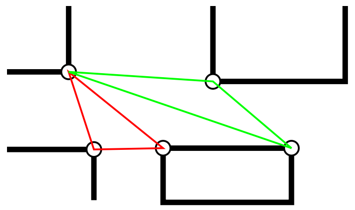


Fig. 4: Demonstration of triangle features (the green and red triangles). The hollow circles are corner points.

C. Loop closure detection

When a submap M^k finishes updating, Shi-Tomasi corner points are extracted. The upper bound of the number of corner points is set according to the size of the submap. To avoid corner points cluttering in a small area, we set an exclusive distance r_{min}. That means, every extracted corner point is the most prominent one within the radius of r_{min}, those corner points in its radius are detected but not extracted. The extracted corner point set is denoted as P^k.

The inter-point distances of P^k will be frequently used, so we can calculate the distance table $dist^k$ in advance, and $dist^k(i,j)$ means the distance between point p_i^k and p_j^k, where $p_j^k, p_j^k \in P^k$.

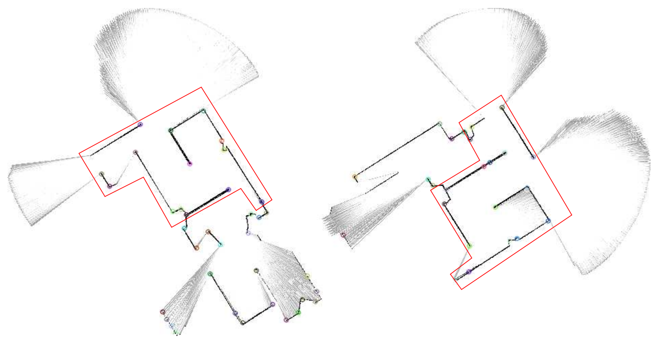


Fig. 5: Demonstration of two submaps going for a match. Overlap regions are bordered with red polygons. The colorful small circles are extracted corner points. Note that even in the overlapped region, not all the corner points have their counterparts in the other grid map.

Then we will generate the triangle features. Theoratically, a point set with $n (n \geq 3)$ elements can generate $\binom{n}{3}$ triangles, we will show how to optimize the generating process later. Let the ith generated triangle be t_i^k (whose ordered vertices are $t_i^k(1), t_i^k(2), t_i^k(3)$ repectively), and a total N^k triangles will be generated, the triangle feature set, or the *signiture* for submap M^k can be denoted as

$$T^k = \{t_i^k \mid i = 1, 2, \ldots, N^k\} \tag{6}$$

And the following process is matching the signitures T^k, T^l of two candidate submaps. For a pair of triangle features t_i^k and t_j^l, we call them *almost congruent* (AC), if and only if

$$\begin{aligned} dist^k(t_i^k(1), t_i^k(2)) &\approx dist^l(t_j^l(1), t_j^l(2)) \\ dist^k(t_i^k(2), t_i^k(3)) &\approx dist^l(t_j^l(2), t_j^l(3)) \\ dist^k(t_i^k(3), t_i^k(1)) &\approx dist^l(t_j^l(3), t_j^l(1)) \end{aligned} \tag{7}$$

note that the criteria of $\approx$ should be customized according to the actual situation. In our case, we define the result of $x_1 \approx x_2 (x_1, x_2 > 0)$ as

$$\frac{|x_1 - x_2|}{\min(x_1, x_2)} < 0.03$$

With a pair of AC triangles, the correspondence of each vertex can be pinned down. Thus, a rigid transform between the two point sets can be estimated using least-squares estimation [23]. The estimation result is a triplet of a translation component and a rotation component in 2D space, we denote the result as a triplet

$$u_{i,j}^{k,l} = (\Delta\theta, \Delta x, \Delta y)$$

i, j are the subscripts of the AC triangles in two feature sets repectively. And the result set of all the estimated transforms between pairs of congruent triangles can be written as

$$U^{k,l} = \{u_{i,j}^{k,l} \mid t_i^k \text{ and } t_j^l \text{ are AC triangle pairs } \} \tag{8}$$

where t_i^k and t_j^l are TF defined in equation (6).

With possible transform set $U^{k,l}$ at hand, we then go through a vote-counting process. First, we divide the angle range (say $[-\pi, \pi)$) into small intervals (say $\pi/180$ per step), and count the number of $u_{i,j}^{k,l}$s whose $\Delta\theta$ fall into every interval (voting for the interval). Then, we pick out the interval(s) getting significantly more votes than the rest intervals and create a new transform set $U_1^{k,l}$ from $u_{i,j}^{k,l}$s which fall into the intervals getting most votes. And the voting process goes on equivalently for Δx, using $U_1^{k,l}$ as input set and producing $U_2^{k,l}$, and the equivalent for Δy to produce $U_3^{k,l}$ at last. If any of the above processes failed to select the interval(s) with a significant number of votes, or the size of a possible transform set $U_i^{k,l}$ $(i = 1, 2, 3)$ is too small, we exit the matching immediately and return a match failure.

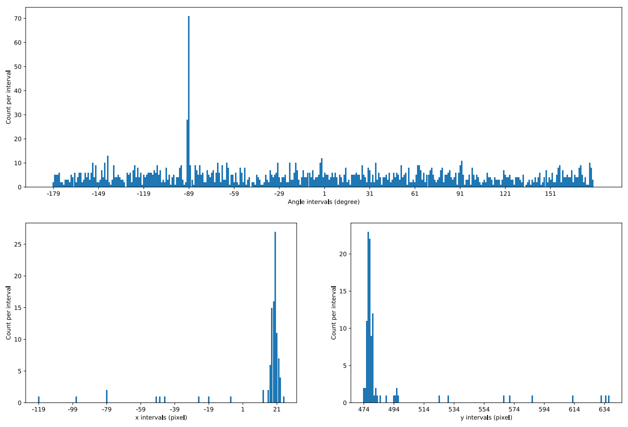


Fig. 6: The voting count of the features of the two submaps in Fig.5 (12 common feature corner points) with a salient potential transform.

Suppose we find a $U_3^{k,l}$ with enough elements, we then use the transform triplets $u_{i,j}^{k,l}$s to find the corresponding corner points in corner point set P^k, P^l of the two submaps. We denote the selected corner point sets as

$$\begin{aligned} P_0^k =& \{p_i^k \mid \exists m, n \in N, \text{s.t. } p_i^k \in t_m^k \text{ and } u_{m,n}^{k,l} \in U_3^{k,l}\} \\ P_0^l =& \{p_i^l \mid \exists m, n \in N, \text{s.t. } p_i^l \in t_n^l \text{ and } u_{m,n}^{k,l} \in U_3^{k,l}\} \end{aligned} \tag{9}$$

One point p_i^k in P_0^k may match with several points in P_0^l during the vote-counting process (this is due to another point p_j^k locates close to p_i^k). A naive way to solve the problem is to use a match count threshold if the two points' matching count is below the threshold, they are decoupled (and they are linked to the points with the maximum matching count). Here we use a graph solution. We first construct a bipartite graph

$$G = (U, V, E)$$

where $U = P_0^k, V = P_0^l$, and E is the matrix recording point match count in the two sets. The more counts a pair of points have, the larger the weight of the edge. So the problem becomes a bipartite graph maximum weight matching and can be solved using the KM algorithm [24]. Up to now, the loop closure detection process is complete.

Looking for a transform becomes easy when we have two selected point sets P_0^k and P_0^l and the correspondence of their elements is established. We can feed the two selected point sets into least-squares estimation [23] to estimate a more accurate transform. We denote the transform from P_0^k to P_0^l as $\boldsymbol{T}_{l,raw}^k$. And using the transform $\boldsymbol{T}_{l,raw}^k$ as an initial guess, we further refine the matching by feeding all the nuclei in the grid map cells of both submaps into ICP [25] for delicate adjustment, which we denote as $\boldsymbol{T}_{l,fine}^k$, and the final transform estimate is

$$\boldsymbol{T}_l^k = \boldsymbol{T}_{l,fine}^k \boldsymbol{T}_{l,raw}^k \tag{10}$$

D. Speed up loop detection

In subsection V-C, we describe the complete process of our loop closure detection method. Though we typically have less than 40 feature corner points per submap, the times of comparing triangles for AC ones are still enormous. We use 3 ways to speed up the process and the results in section VI turn out satisfying.

The first is to kick out triangles with long edges. It can be expected that partially overlapped submap pairs are more likely to appear than entirely overlapped pairs. For this reason, triangle features with edges spanning across the whole submap can rarely find an AC counterpart even there's a loop between two submaps. So we set a length threshold L_{max}^k for submap M^k, and don't generate triangles which have edges longer than L_{max}^k in the first place.

The second is to sort the TF sets T^k and T^l of submap M^k and M^l, before finding AC triangles from the two sets. This is because failing any one of condition (7) will lead to a match failure. We order the $t_i^k \in T^k$ by $\text{dist}^k\left(t_i^k(1), t_i^k(2)\right)$ (i.e. longest edge of the traingle) in ascending order, and equivalent for $t_j^l \in T^l$. After sorting we define a set $S^{k,l}$ for pairs $s_i^{k,l}$ can be quickly calculated in $O(n)$ time complexity

$$\begin{aligned} S^{k,l} = \{s_i^{k,l} \mid s_i^{k,l} = (i, j), &\text{j is chosen such that} \\ L_j^l \text{ is the smallest satisfying } &L_j^l \approx L_i^k \text{ otherwise } j = 0\} \end{aligned} \tag{11}$$

where $L_i^k = \text{dist}^k\left(t_i^k(1), t_i^k(2)\right)$, $L_j^l = \text{dist}^l\left(t_j^l(1), t_j^l(2)\right)$. In this way, for a given t_i^k, we can start comparing at index in $s_i^{k,l}.second$ the ordered set of T^l, and exit as soon as $L_j^l \approx L_i^k$ is no longer valid, and start with t_{i+1}^k and going on.

The third method is a probabilistic approach. It sets an upper bound for the number of AC triangle pairs. In our experiment, we find that when a loop exists, the propotion of votes for right transform parameters (i.e. $\Delta\theta, \Delta x, \Delta y$) is very high, by randomly reducing voters (AC triangle pairs) to a reasonable number will not affect the vote result. So we randomly choose elements from $S^{k,l}$ mentioned above till we have enough voters engaged. The idea is similar to probabilistic Hough transform [26].

VI. Experimental results

Simulations are conducted to verify our SLAM system proposed in section IV and V.

In a simulation, a kobuki turtlebot is mounted with a noisy laser rangefinder, a noisy IMU, and a perfect odometry. Note that the odometry data is for the use of comparison SLAM methods, and our method will not use it. And an arena with a loop is built, with the turtlebot driving in the arena to collect data for SLAM. We will show that our method is robust and fast compared to existing SLAM solutions.

TABLE I: Simulation configuration

Parameter	Value	Parameter	Value
Grid resolution	0.05 m	IMU update rate	1000 Hz
Scan update rate	6 Hz	IMU angular stddev	2e-4
Min scan range	0.22 m	IMU angular bias mean	7.5e-6
Max scan range	10.00 m	IMU angular bias stddev	8e-6
Scan stddev	0.02 m	Arena size	$23 \times 38m^2$
Running duration	460 s	Running path length	128.4 m

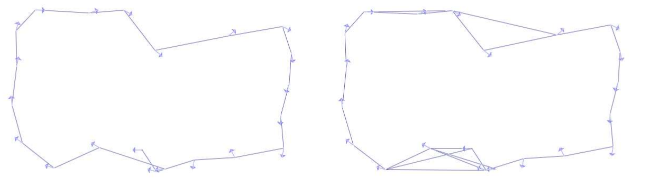


Fig. 7: Left: Pose graph before loop closure detection. Right: Pose graph after loop closure detection and graph optimization.

We run our SLAM system on an Intel i7-7700k CPU on a single thread, which means the backend is run after all the 21 submaps with a typical size of 400×400 pixels and 23 feature corner points are finished. This is not our final practice, but it helps us analyze the time consumption of each module of the system.

By using the speed-up methods in V-D, we only compare 5.84% of all the possible pairs (after kicking out long-edge triangles) going for an AC check, or 37500 pairs per detection, on average, while an average number of 23 feature corner points will lead to 3136441 times of triangle comparing every detection by brutal force.

Our SLAM finishes building a global map in $194.3 \pm 0.3s$, which is about 2.4 times of real-time. Since our loop closure exits the moment it finds impossible to detect a loop, matching between 2 submaps with no loops is significantly faster than whose having loops. The calculation time of 2 submaps having loops is about $0.03s$ every time, and calculation of no loop is typically under $0.01s$, sometimes even under $0.001s$. In the global loop searching period, where 21 submaps are trying to find loop with the submaps whose id is 2 less equal than its own, 190 loop closure attempts were made, and a total of 9 loops (including local loops and global loops) are found, and the total calculation time is $1.90 \pm 0.02s$, which means $0.01s$ per loop detection on average. At this rate, when the number of submaps growing larger, the loop detecting time cost will stay modest.

The pose graph before and after loop closure and optimization is shown in Fig. 7, and the global map and comparison is shown in Fig. 8.

For simplicity, the man-made simulation arena doesn't have too many randomized clutter scenes which is common in indoor environments (like tables, potted plants, piles of sundries, etc.). This means our submaps have to be large enough to include enough feature corner points, while larger submaps means larger error. We expect our system to have better performance in a feature-rich environment.

VII. Conclusions

In this paper, we propose a geometric environment descriptor called a *triangle feature* (TF). A TF based submap-to-submap matching and loop closure detection method is proposed. And a graph SLAM solution using the loop closure method is developed and tested on simulation data, the result outperforms some popular existing SLAM solutions.

One problem of our loop detection method is that in highly identical scenarios with the same major geometric features, like a long hallway with rooms of the same design on both sides, incorrect constraints may be added to the optimization. Human-activity-caused environment features or the building designs themselves are not highly repetitive. Techniques like lazy decision loop checks can be added to improve robustness in repetitive environments.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[2] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

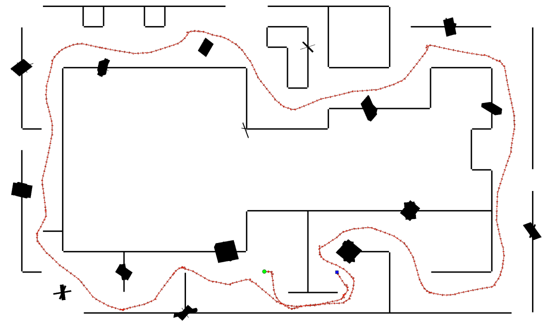
[3] G. Grisetti, C. Stachniss, W. Burgard *et al.*, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, p. 34, 2007.

[4] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. IEEE, 2011, pp. 155–160.

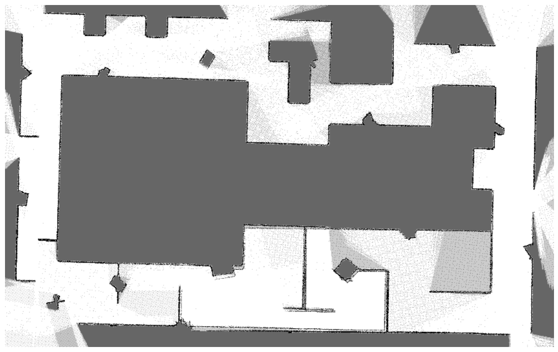
[5] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[6] M. Labbe and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 2661–2666.

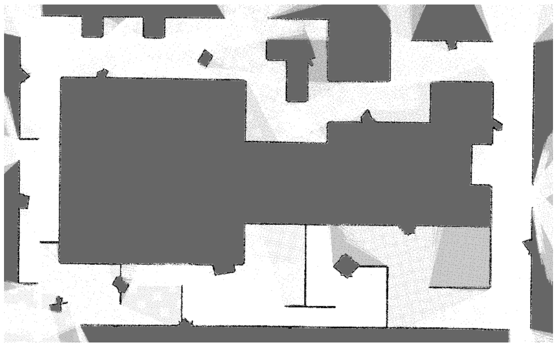
[7] G. D. Tipaldi and K. O. Arras, "Flirt-interest regions for 2d range data," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 3616–3622.



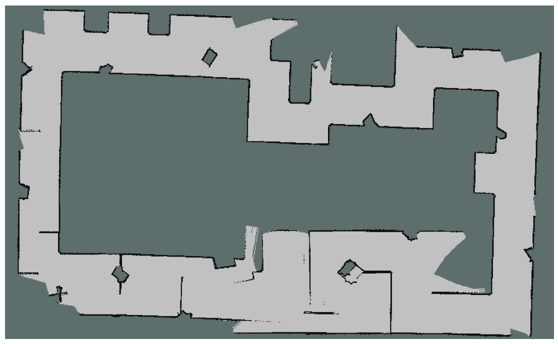
(a) The ground truth map. The red line is ground truth robot path, from the green circle to the blue square.



(b) Map built by our SLAM, before loop closure.



(c) Map built by our SLAM, after loop closure and graph optimization.



(d) Map built by gmapping [3].

Fig. 8: Simulation results and comparison with gmapping

[8] W. Yoo, H. Kim, H. Hong, and B. H. Lee, "Scan similarity-based pose graph construction method for graph slam," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 131–136.

[9] J.-L. Blanco, J. González-Jiménez, and J.-A. Fernández-Madrigal, "A robust, multi-hypothesis approach to matching occupancy grid maps," *Robotica*, vol. 31, no. 5, pp. 687–701, 2013.

[10] M. Himstedt, J. Frost, S. Hellbach, H.-J. Böhme, and E. Maehle, "Large scale place recognition in 2d lidar scans using geometrical landmark relations," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 5030–5035.

[11] S. Carpin, "Fast and accurate map merging for multi-robot systems," *Autonomous Robots*, vol. 25, no. 3, pp. 305–316, 2008.

[12] E. B. Olson, "Real-time correlative scan matching," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 4387–4393.

[13] A. Censi, "An icp variant using a point-to-line metric," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 19–25.

[14] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[15] E. Olson, "M3rsm: Many-to-many multi-resolution scan matching," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5815–5821.

[16] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 22–29.

[17] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[18] D. G. Lowe *et al.*, "Object recognition from local scale-invariant features." in *ICCV*, vol. 99, no. 2, 1999, pp. 1150–1157.

[19] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[20] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf." in *ICCV*, vol. 11, no. 1. Citeseer, 2011, p. 2.

[21] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration." *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[22] J. Shi and C. Tomasi, "Good features to track," Cornell University, Tech. Rep., 1993.

[23] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 4, pp. 376–380, 1991.

[24] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957. [Online]. Available: https://doi.org/10.1137/0105003

[25] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–607.

[26] N. Kiryati, Y. Eldar, and A. M. Bruckstein, "A probabilistic hough transform," *Pattern Recognition*, vol. 24, no. 4, pp. 303–316, 1991.