Octree-Based Repetitive Pose Detection of Large-Scale Cyclic Environments

Yibo Wang, Hongbiao Zhu, Weidong Wang*
State Key Laboratory of Robotics and System
Harbin Institute of Technology (HIT)
Harbin, China
wyb3_14@163.com, ibiaozi@163.com, wangweidong@hit.edu.cn

***Abstract*—Aiming at the time-consuming problem of repetitive detection for mobile robots in large-scale cyclic environments, this paper presents an octree-based repetitive pose detection approach which can reduce the detection time effectively. Compared with the traditional graph-based detection method, the novelty of our algorithm lies in the high accuracy, in particular, consuming time does not rise with the expansion of the poses storage. Based on the octree, the 3D space of position is segmented and converted into a binary format for storage or search. The orientations are stored at the leaf nodes of the octree, and the similarity of them is determined by the vectors matching method. The proposed algorithm is evaluated using real-world experiments and different simulated datasets including KITTI and Oxford, the evaluation results indicate the superior performance in terms of efficiency and accuracy using our algorithm in the large-scale cyclic environments.**

Keywords-octree; pose-detection; mobile robot;

I. INTRODUCTION

Repetitive pose (position plus orientation) detection is used to detect whether the same pose have been experienced in the past scenes for mobile robot in this paper, it is widely used in various applications including the situations that a robot moves continuously in a cyclic environment [1] and the scenes that multiple robots [2] detect the repetitive poses by sharing information with each other. On the other side, robots can choose to move away from the repetitive poses or close to the repetitions, the first case often occurs in the unknown environment exploration tasks [3] of mobile robots aiming to reduce the repeated exploring areas and complete the tasks in the shortest time while ensuring the integrity of exploration, in that case, information of repetitive poses can be used as an important evaluation criterion, the repetitions can inevitably lead to the overlap of the exploration areas, while the explorations can be implemented faster by moving far away from the repetitions; the second case can occur in trajectory-tracking tasks [5], in that case, robots can record the trajectories optimized by the control algorithms and then a faster operation can be achieved by retracing trajectories including information of poses without recalculation. Loop-closure detection [6], an important part of backend optimization for SLAM, is a typical process of repetitive pose detection, for instance, the odometry based loop-closure detection [1] proposed by Hahnel D *et al*, builds parametric model of the odometry error through the experiments and confirms loop-closure by comparing the relative poses.

The small-scale pose detection can be simply realized by container storage and traversal, so it is often ignored by the researchers. But with the increase of storage which often occurs in large-scale cyclic environments, consuming time will also increase obviously and can't be ignored anymore.

The existing methods of repetitive pose detection is mainly implemented by graph structure. In Pose-Graph [7-9], vertices are used to represent robot poses, and edges are used to estimate the relative motions between vertices, it is a pity that searching for poses is a time-consuming process by the way of traversal in the Pose-Graph, whether the depth first search or the breadth first search, and the process of searching the specified data has a certain randomness according to the initial value.

Octree can realize the segmentation of 3D space, its typical application is OctoMap [10,11], a well-known 3D volumetric map, which is generally applied to the tasks of navigation [12]. In this paper, an octree-based repetitive pose detection method is proposed, which is faster than the detection method based on graph structure especially in the large-scale environments. In particular, consuming time does not rise with the expansion of the storage of poses using our algorithm. The number of position search times is independent of the amount of storage and the maximum number of times per search is d equaling to the depth of octree set by the user. The orientation detection is executed after position search, the required execution times are determined by the angle threshold θ_{th} set by the user and the quantity of orientations that have been stored, in this way, the number of orientations to be searched is limited to a certain extent to ensure the high efficiency of algorithm.

In the following sections we will explain the proposed repetitive pose detection algorithm in more detail and then evaluate our approach using real-world experiments and different simulated datasets including KITTI and Oxford.

II. PROPOSED ALGORITHM

The proposed robot repetitive pose detection algorithm includes the handling of position and orientation, regardless of the search or the storage procedure, the priority of the position information is higher than the orientation information, that is, the position information is searched and stored firstly, and then the orientation, which is determined by the algorithm structure: the storage and search of position are implemented based on octree, it is necessary to construct or search the complete tree structure firstly, and then store the orientation at the leaf nodes of octree or use the vectors comparison algorithm to compare the orientations stored at

the leaf nodes. We also set the adjustable storage intervals for pose to improve the practicability of the algorithm, in which the storage interval of position is expressed by the resolution r of the octree, and the orientation interval is expressed by the angle threshold θ_{th} of vectors.

The priority of the search is higher than the storage in the algorithm, that is, the search first and then the storage. After the acquirement of information to be searched, the algorithm searches for the information in the stored structure of octree, and determines whether to store the position or orientation into the octree according to the search result:

- Nothing will be updated if both the position and orientation to be searched are within the interval set by user.
- If the position information is not found, the robot can update the to be searched pose directly without matching the orientation information anymore.
- If the position is searched successfully but the to be searched orientation is not near the stored ones, only the orientation will be updated.

The main steps of proposed algorithm are described in Algorithm 1.

Algorithm 1. Repetitive pose storage and detection

```
Initialize parameters; // (e.g., r, θth , etc.)
Repeat //main loop
    get the 3D position data; // type: double
    convert position data into binary;
    combine binary data as shown in Fig. 2;
    for i=1, …, d do
        search the ith combined binary;
        if (Search Failed) then
            // failed to search position;
            store pose;
            return (no position detected);
        end
    end
    get the rotation matrix R_R^W ;
    get V_W^s using Eq. (1) and Eq. (2);
    num = 0;// num: number of similar orientations
    for i=1, …, n do
        compare V_W^s and V_W^i using Eq. (3);
        if ( θ ≤ θth ) then
            num++;
        end
    end
    if (num == 0)
        // failed to search orientation;
        store the orientation only;
        return (no orientation detected);
    else
        return (pose detected);
    end
Until be stopped by user;
```

A. *Position Storage and Search*

The process of position storage and search is implemented based on octree, the resolution of octree (r) represents the accuracy of stored positions, and octree depth (d) represents the number of spatial segmentations. In the process, the robot need to get the 3D coordinates of position firstly, and then transform the 3D data (x, y, z) into three sets of binary numbers respectively, the number of binary digits in each set is the same as the octree depth and the high digits are complemented by 0, then the segmentation of three-dimensional space can be described by binary combinations, Figure 1. shows data storage format and the way of binary combination, in each spatial segmentation, each set of binary numbers provides only one bit of binary data (0 or 1) in sequence, and is spliced into three-digit binary numbers in the order of (z, y, x), corresponding to 0-7 in decimal. The above process can be understood from different aspects: from the geometric point of view, it divides the three dimensions of space and forms eight independent sub-regions (refer to Figure 2.), the combinations of binary from high digits to low digits can represent the coarse to fine segmentation process of space; from the perspective of tree structure, it corresponds to the process of spreading from the root node to the leaf node, and when the number of subdivisions is equal to the set tree depth, it is considered that the entire subdivision process of the space is completed, at this time the minimum region length corresponds to the resolution of the octree. We set the tree depth to d and the resolution to r, then each dimension can represent the distance of $r \cdot 2^d$.

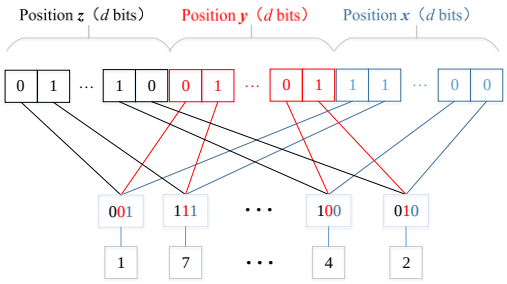


Figure 1. Data storage format and binary combination.

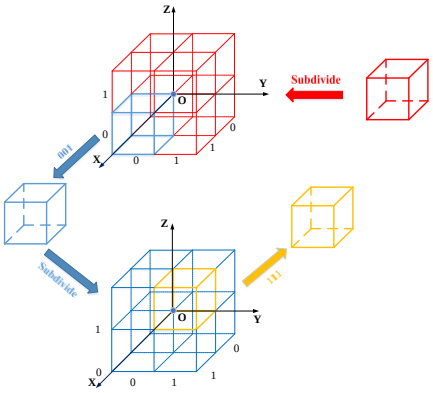


Figure 2. Space segmentation using octree. It shows the correspondence between three dimensional coordinates and binary, the space is subdivided 2 times corresponding to the first 2 times binary combination in Figure1.

Figure 3. illustrates the structure of octree, in which layers are connected by pointers, and the pointers point from the root node to the next layer until the leaf nodes. In the initial state, each node in the octree is set to NULL, and the

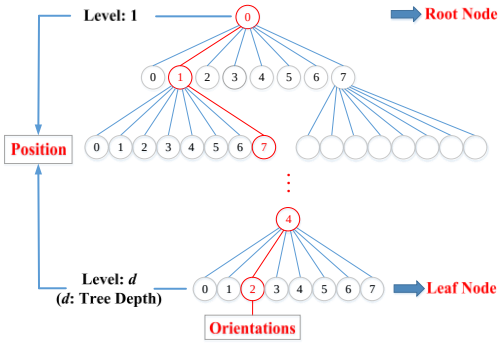


Figure 3. The structure of the octree. Position data is stored from level 1 (root node) to level *d* (leaf node), and the orientations are hanged up under the leaf nodes after the storage of position. The red numbers correspond to the decimal result by combining binary shown in Figure 1.

position storage process is to update the NULL of each layer 0-7 position to True according to the result of binary combinations.

The position search is the reverse process of the stored procedure. The new information of position needs to be converted into binary numbers and combined into 0-7 in the same manner as the storage mentioned above, and then compare the decimal digit with position data stored in the octree along the direction of pointers, when all the data is searched, it is judged that the searched position had been reached before and will not be stored one more time. Otherwise, the searched position is determined to be a new one, and it needs to be stored in the octree for later search.

B. Orientation Storage and Search

The orientations are stored in containers in the form of vectors, as shown in Figure 3. we set the pointer pointing from the leaf node to the orientations, which is used to contact the position and orientation information. This setting can realize a one-to-many mapping of position and orientation, that is, one position can correspond to multiple orientations, which is consistent with common sense. Sparse orientation storage can be achieved by setting the storage interval (angle threshold θ_{th}) and comparing the new vector to the stored vectors. Therefore, the number of orientations in the containers is limited, and the larger the angle threshold, the smaller the maximum number of orientations stored in the containers, in this way, the number of orientation comparisons can be effectively limited according to demand.

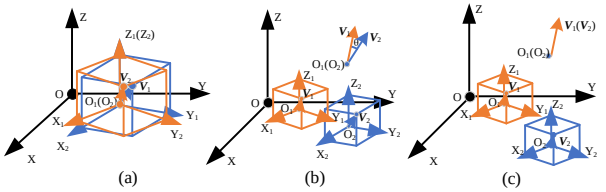


Figure 4. The process of vectors matching. The black coordinate system represents the world coordinate system, and the remaining two coordinate systems represent robot coordinate systems. (a) shows vectors matching when positions coincide. (b)(c) show vectors matching when positions coincidence incompletely, the angle between two vectors is expressed as θ, and θ=0 when vectors are parallel or overlapped as shown in (c).

Orientation search is implemented by matching the angle between two vectors. Since we use the octree resolution as the position storage interval, the defined 'repetitive position' is not strictly coincident, and the resolution can be treated as the permissible position error. Using vectors can ensure that the orientation search results will not be affected by the position error. Figure 4. illustrates the process of vectors matching. We set the robot coordinate system to $\{R\}$, and the world coordinate system is set to $\{W\}$, which coincides with the robot coordinate system at the initial position and remains unchanged, then bind the unchanging virtual points on $\{R\}$, expressed as $P_0^R = [0,0,0]^T$ and $P_1^R = [1,1,1]^T$, they form a fixed vector in the robot coordinate system, expressed as $\boldsymbol{V_R} = [1,1,1]^T$.

The homogeneous transformation matrix converted from the robot coordinate system to the world coordinate system:

$$T_R^W = \begin{bmatrix} R_R^W & p_R^W \\ 0 & 1 \end{bmatrix} \tag{1}$$

where R_R^W is the rotation matrix, and p_R^W is the translation vector.

The vector $\boldsymbol{V_R}$ in the world coordinate system can be expressed as:

$$\boldsymbol{V_W} = T_R^W P_1^R - T_R^W P_0^R = R_R^W \boldsymbol{V_R} \tag{2}$$

where $\boldsymbol{V_W}$ is the transformed robot pose needs to be stored in our algorithm. The set of the stored vectors is expressed as: $\boldsymbol{V} = \{\boldsymbol{V_W^1}, \boldsymbol{V_W^2}, \cdots, \boldsymbol{V_W^i}, \cdots, \boldsymbol{V_W^n}\}$, where $i \in [1, n]$, and n is the number of members $\in \boldsymbol{V}$.

The orientation search is the process of comparing the search vector with the stored vectors, and obtained angle θ is:

$$\theta = \arccos \frac{\boldsymbol{V_W^s} \cdot \boldsymbol{V_W^i}}{\left|\boldsymbol{V_W^s}\right| \cdot \left|\boldsymbol{V_W^i}\right|} \tag{3}$$

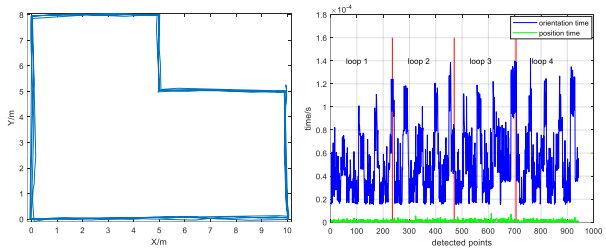
It is significant to define the so-called 'similarity' in the process of comparing vectors, we use a user setting parameter θ_{th} (angle threshold) to represent the vector similarity, if $\theta \le \theta_{th}$ the orientation is judged to be a repetitive orientation, since the position information had been searched in the previous work, the searched pose can be considered as a repetitive pose and will not be stored one more time. Otherwise, although the position is repetitive, the orientation information is new to the octree and needs to be stored.

III. EXPERIMENTS

In the section, we implement the real-world experiments based on Robot Operating System (ROS) and use the datasets of KITTI and Oxford to further evaluate the effect

of the proposed repetitive pose detection algorithm in the large-scale environments.

Figure 5. The mobile robot experimental platform.



(a) Experimental trajectory (b) Consumed time

Figure 6. Experimental trajectory and time consuming. (a) shows partly coincident experimental trajectories of the 4 loops. (b) shows consumed time of position (green) and orientation (blue) detection in every loop.

In order to verify the proposed algorithm, it is necessary to provide the robot pose information firstly, which includes the information of position and orientation. Figure 5. shows our mobile robot experimental platform. In the real-world experiments, the positioning function is realized by data fusion between GNSS and IMU based on EKF. The GNSS is based on RTK (Real-Time Kinematic) technology, which can realize centimeter-level positioning accuracy through differential signal processing between base station and mobile terminal. The IMU uses a nine-axis fusion algorithm of gyroscope, accelerometer and magnetometer to provide the motion orientation of mobile robot. This positioning method can eliminate the accumulated error in a large-scale environment while ensuring high-precision positioning.

Figure 6(a). shows the experimental trajectory of 4 loops for repetitive pose detection using our mobile robot, we recorded the consumed time data of the robot pose detection as shown in Figure 6(b). and TABLE I. , according to the experimental data, consumed time of per detection is maintained at the range of 1.0×10^{-6}s-1.0×10^{-4}s, the robot

TABLE I. CONSUMED TIME (MOBILE ROBOT)

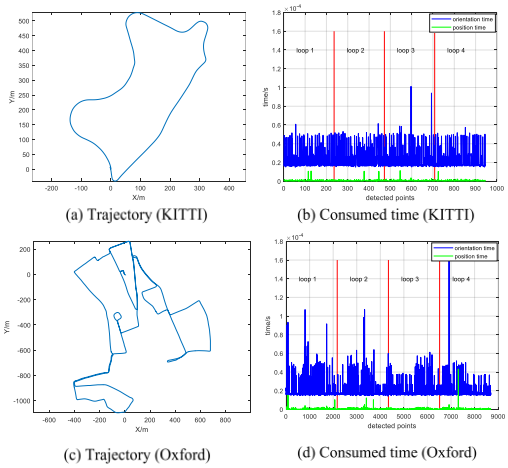
Average time(s)	Orientation	Position	Pose	Standard deviation
	3.7806e-05	9.6985e-07	3.8776e-05	3.0998e-05
Loop 1	3.3122e-05	6.5144e-07	3.3773e-05	1.7558e-05
Loop 2	3.9107e-05	9.7997e-07	4.0087e-05	2.1251e-05
Loop 3	4.4731e-05	1.2117e-07	4.5943e-05	2.2081e-05
Loop 4	3.4265e-05	1.0363e-07	3.5301e-05	1.8273e-05

TABLE II. UPDATE NUMBER (MOBILE ROBOT)

	4 Loops (r: 0.2m, θ_{th} :15°, d:16)			
	1	**2**	**3**	**4**
Update number	197	19	10	9

updates the storage after the detection of different pose, TABLE II. illustrates the update numbers in 4 loops (r: 0.2m, θ_{th} :15°, d:16), the first loop indicates the initialization process of the robot reaching the unknown environment for the first time, and the number of updates is relatively large, the remaining loops can detect the repetitive poses stored before but it is difficult to ensure that running trajectories are completely coincident with them, which leads to a small number of updates.

In order to verify the performance of the proposed algorithm in a large-scale environment, we used the poses information of KITTI dataset and Oxford dataset, which contain 1592 and 126362 different pose points respectively. Comparing to the above-mentioned mobile robot experiments, the datasets provide more poses and a larger range of motion. The trajectories are shown as Figure 7(a)(c). , we also set the loop number to 4 in this part. Figure 7(b)(d). and TABLE III. show us the consumed time per detection and its variation trend using two datasets. Comparing with the KITTI dataset, the Oxford dataset provide more poses to be searched and stored, but the time spent on detection remains stable and shows no trend of increase, that is, the time of detection does not increase with the amount of poses storage, which demonstrate the efficiency of the algorithm in a large-scale cyclic environment.



(a) Trajectory (KITTI) (b) Consumed time (KITTI)

(c) Trajectory (Oxford) (d) Consumed time (Oxford)

Figure 7. Trajectory and consumed time using KITTI and Oxford dataset.

The first loop in the experiment is used to traverse and store all the poses provided by datasets, and it is different from the previous real-world experiments that the other three loops use the same poses data as the first loop, so their trajectories coincide completely. TABLE IV. shows that except for a certain number of updates in the first loop, the number of updates in the remaining loops is 0, which means

the accuracy of the proposed repetitive pose detection algorithm is 100%.

The KITTI dataset has a low density of poses, while the data density in the Oxford dataset is large and there are some repetitive poses in the trajectory. We gradually increased the detection density by setting parameters (r, θ_{th}, d) and detected the number of pose update as shown in TABLE IV. According to the analysis, the increase of the detection density has a certain effect on the experiments using Oxford dataset which is with a higher pose density, it leads to an increase in the number of pose updates, confirming the practicability of the algorithm parameters which can be set as need by user.

TABLE III. CONSUMED TIME (KITTI & OXFORD)

Average time(s)		Orientation	Position	Pose	Standard deviation
KITTI (1592)		2.4639e-05	6.6793e-07	2.5306e-05	1.2208e-05
	1	2.4556e-05	6.9323e-07	2.5249e-05	1.1488e-05
	2	2.5091e-05	6.7145e-07	2.5762e-05	1.2133e-05
	3	2.5104e-05	6.5908e-07	2.5764e-05	1.3525e-05
	4	2.3803e-05	6.4794e-07	2.4451e-05	1.1499e-05
Oxford (126362)		1.7462e-05	2.0961e-07	1.7672e-05	5.2048e-06
	1	1.7422e-05	2.0268e-07	1.7625e-05	5.5866e-06
	2	1.7698e-05	2.2618e-07	1.7924e-05	7.0578e-06
	3	1.7440e-05	2.0281e-07	1.7643e-05	5.6958e-06
	4	1.7288e-05	2.0678e-07	1.7495e-05	5.6867e-06

TABLE IV. UPDATE NUMBER (KITTI & OXFORD)

Update number	Loop	Parameters (r, θ_{th}, d)		
		(0.1,10,16)	(0.05,5,16)	(0.02,2,16)
KITTI (1592)	1	1592	1592	1592
	2-4	0	0	0
Oxford (126362)	1	70947	79700	85291
	2-4	0	0	0

IV. CONCLUSIONS

In this paper, we proposed an octree-based repetitive pose detection method, which addresses the problem of large time-consuming in the pose detection tasks, comparing to the traditional graph-based method, our approach can perform better in a large-scale cyclic environment. The major advantage of our approach is that while ensuring the high accuracy, the detection time does not rise with the increase of the number of poses storage, and it is conceptually straightforward and easy to be implemented. The position detection is mainly realized based on octree, and the orientation detection is implemented by vectors comparison on the basis of octree structure. In addition, we set up the adjustable octree resolution and angle threshold which can be used for sparse pose storage and search according to the requirement, it increases the practicability of the algorithm. In the experimental part, the GNSS-IMU framework is used to provide the pose data of outdoor mobile robot without error accumulation, which verifies the feasibility of our approach. Then the KITTI dataset and Oxford dataset were used to further complete the larger-scale experimental verification, the results of the experiments show that the average consuming time of our pose detection method is less than 1.0×10^{-4}s, and the repetitive detection accuracy is up to 100%.

The proposed algorithm is a basic method. For future work, we will pay more attention to the application of our approach, for instance, the exploration of unknown environment using mobile robots, it can be foreseen that the proposed octree-based repetitive pose detection algorithm can improve the exploration efficiency and complete the exploration tasks more quickly.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (61773141).

REFERENCES

[1] Hahnel D, Burgard W, Fox D, et al. An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements[C]//Ieee/rsj International Conference on Intelligent Robots and Systems. IEEE, 2003:206-211 vol.1.

[2] Amigoni F, Banfi J, Basilico N. Multirobot Exploration of Communication-Restricted Environments: A Survey[J]. IEEE Intelligent Systems, 2018, 32(6):48-57.

[3] Charrow B, Liu S, Kumar V, et al. Information-theoretic mapping using Cauchy-Schwarz Quadratic Mutual Information[C]// IEEE International Conference on Robotics and Automation. IEEE, 2014:4791-4798.

[4] Charrow B, Kahn G, Patil S, et al. Information-Theoretic Planning with Trajectory Optimization for Dense 3D Mapping[C]// Robotics: Science and Systems. 2015.

[5] Alessandretti A, Aguiar A P, Jones C N. Trajectory-tracking and path-following controllers for constrained underactuated vehicles using Model Predictive Control[C]// Control Conference. EUCA, 2013:1371-1376.

[6] Angeli A, Filliat D, Doncieux S, et al. Fast and Incremental Method for Loop-Closure Detection Using Bags of Visual Words[J]. IEEE Transactions on Robotics, 2008, 24(5):1027-1037.

[7] Dubbelman G, Browning B. COP-SLAM: Closed-Form Online Pose-Chain Optimization for Visual SLAM[J]. IEEE Transactions on Robotics, 2015, 31(5):1194-1213.

[8] Lee D, Myung H. Solution to the SLAM problem in low dynamic environments using a pose graph and an RGB-D sensor[J]. Sensors, 2014, 14(7):12467.

[9] Latif Y, Cadena C, Neira J. Robust loop closing over time for pose graph SLAM[J]. International Journal of Robotics Research, 2013, 32(14):1611-1626.

[10] Wurm K M, Hornung A, Bennewitz M, et al. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems[C]// Proc. of the ICRA Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation. 2010.

[11] Hornung A, Kai M W, Bennewitz M, et al. OctoMap: An efficient probabilistic 3D mapping framework based on octrees[J]. Autonomous Robots, 2013, 34(3):189-206.

[12] Hornung A, Phillips M, Jones E G, et al. Navigation in three-dimensional cluttered environments for mobile manipulation[C]// IEEE International Conference on Robotics and Automation. IEEE, 2012:423-429.