Special Issue Article

Advances in Mechanical Engineering

Advances in Mechanical Engineering
2018, Vol. 10(2) 1–15

DOI: 10.1177/1687814018761296
journals.sagepub.com/home/ade
SAGE

A fast incremental map segmentation algorithm based on spectral clustering and quadtree

Yafu Tian, Ke Wang, Ruifeng Li and Lijun Zhao

Abstract

Currently, state-of-the-art simultaneous localization and mapping methods are capable of generating large-scale and dense environmental maps. One primary reason may be the applications of map partitioning strategies. An efficient map partitioning method will decrease the time complexity of simultaneous localization and mapping algorithm and, more importantly, will make robots understand a place anthropomorphically. In this article, we propose a novel map segmentation algorithm based on quadtree and spectral clustering. The map is first organized hierarchically using quadtree, and then a user-friendly criterion is utilized to construct the corresponding Laplacian matrix for quadtree so that spectral clustering can be solved efficiently based on the sparse property of the matrix. In this article, we go further to provide a real-time, incremental, parallel algorithm that can be implemented on multi-core CPU/GPU to enhance the performance of the proposed basic algorithm. Our algorithms are verified under multiple environments including both simulation and real-world data, and the results reveal that the algorithm can provide a correct and user-friendly segmentation result in a short runtime.

Keywords

Autonomous map segmentation, quadtree, spectral clustering

Date received: 10 July 2017; accepted: 19 January 2018

Handling Editor: Fei Chen

Introduction

Simultaneous localization and mapping (SLAM) is a fundamental ability for robots which operate autonomously in unknown places. Some state-of-the-art SLAM methods are able to construct large-scale[1] or dense environmental maps with a laser range finder,[2–4] monocular vision,[5] or an RGB-D sensor.[6,7] Successful methods generally adopt a divide-and-conquer strategy[5] which aims to segment the whole map into several submaps to accelerate map construction from a computational perspective.[8]

Currently, semantic SLAM is becoming a hot topic. In terms of environmental understanding, it is nature to segment a big environment into several small places with semantic functions. These semantic information helps robots to recognize places,[9] navigate, and interact with people anthropomorphically.[10] As semantic SLAM usually combines semantic and geometry information,[9] automatic map segmentation can add semantic "label" to robots' working area and identify semantic boundaries of particular places. In this way, robots may act more like humans and be able to extract semantic information kitchen from the given command "Go to kitchen" and navigate to the corresponding segmented kitchen place.

State Key Laboratory of Robotics and Systems, Harbin Institute of Technology, Harbin, P.R. China

Corresponding author:
Ke Wang, State Key Laboratory of Robotics and Systems, Harbin Institute of Technology, Harbin 150080, P.R. China.
Email: wangke@hit.edu.cn

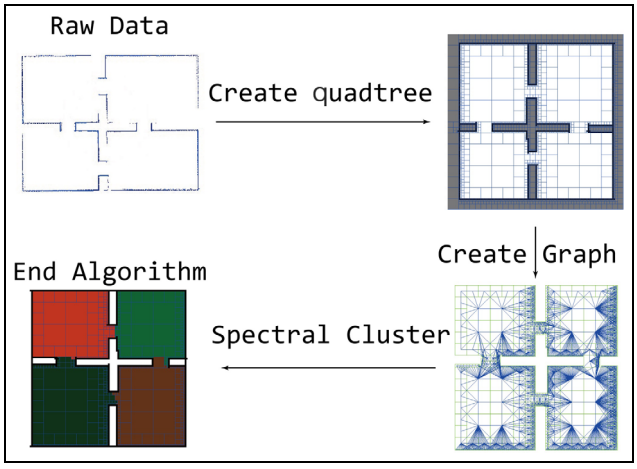

Figure 1. Diagram of the proposed algorithm.

x_{map}, y_{map}	Scale of the workplace
$\mathcal{T}$	QuadTree
$G < V, E >$	Corresponding topological graph of $\mathcal{T}$
v	Nodes in $\mathcal{T}$ and $\mathcal{T}$, represent a map unit
V	Collection of nodes in graph G
e	Edge in graph G
E	Collection of edges e_{ij} in graph G .
η	Normalizing factor
v	Node in graph G, or node in QuadTree $\mathcal{T}$
e_{ij}	Edge between v_i and v_j. It's weight represents the node's relevance.
v_r	Root node of QuadTree
Th	Prior given threshold
Obv_i	Sensor observation result in world coordinate.
S_a, S_b	Submap generated by map segmentation algorithm.

Figure 2. Symbols and their meanings.

In this article, we provide a novel autonomous map partitioning algorithm based on quadtree and spectral clustering. This algorithm intends to provide robots a human-like recognition which is able to segment indoor rooms from a constructed map. Considering both computational efficiency and recognition rationality, we present a criterion to generate a sparse Laplacian graph for spectral clustering. A corresponding parallel and online incremental map partitioning algorithm is also presented for a multi-core CPU/GPU processor to improve the computational speed.

The rest of the article is organized as follows: After a brief introduction on some related works in section "Related work," two basic algorithms, quadtree and spectral clustering, are introduced in sections "Constructing a robot map with quaternary tree" and "Map partitioning based on spectral clustering." The details and improvement of parallel incremental algorithm is discussed in section "Algorithm implementation." Finally, some simulation and experimental results demonstrating the efficiency of precision of the proposed algorithm are presented.

A brief introduction to our algorithm is shown in Figure 1. The algorithm accepts raw laser data as input and generates segmented quadtree-based map representation. By optimizing the algorithm from many aspects, it can run dynamically when robots explore the world and generate real-time segmentation results.

The symbols used in this article and their meanings are shown in Figure 2.

Related work

Most of the map segmentation works focus on the feature-based map representation. Leonard and Feder[1] proposed an early algorithm called decoupled stochastic mapping. This algorithm was designed to balance the time complexity and precision in robot state estimation. And Pinies et al.[11] provide another method called CI-Graph. This method can efficiently solve complex trajectories with low computation and memory cost.

Another way to segment the map is to divide robot workspace into many atomic map units and re-cluster those units. There are many ways to make such segmentation:

1. Occupancy grid map representation,
2. Voronoi graph[4],
3. Spatial search tree (quadtree, KD-Tree, Octree).

Most SLAM algorithms use occupancy grid map representation. However, this representation may create too many map units and may lead to some difficulty in the next clustering process. Voronoi graph is an excellent and widely used algorithm. It creates less map units than quadtree and gives a more human-like map representation[4] based on Voronoi graph to divide map into parts. Based on this work, Sjöö[3] provides a method without using topological graph and transformed the map segmentation method into an energy function optimization problem. But the main disadvantage of Voronoi graph is that the scale and position of each unit are irregular. Finding the belonging triangle of a specific point in Delaunay triangle mesh is not as fast as in spatial search tree, which will bring additional time complexity of indexing and clustering operations.

KD-Tree, which is similar to quadtree, utilizes the variance and mean value of the sample to balance the tree. In this way, KD-Tree based algorithms reduces their mean time complexity. When robot tried to update map, the whole tree will be reconstructed

because the samples' median and variance changed. As KD-Tree is a particular kind of balanced binary tree, it has less efficiency in construction and operation. quadtree map does not need to reconstruct the whole tree every time. In most cases, only a small subtree will be reconstructed. Figure 13 presents the performance advantage.

Similar to Voronoi graph, each KD-Tree leaf's scale is irregular. This brings additional difficulty in algorithm design. So quadtree is a better choice in most cases. But in some extreme situations, for example, if the distribution of obstacles concentrates on a small area, quadtree will degenerate and lose its resolution. And KD-Tree, as it is a balanced binary tree, may have advantage in performance.

quadtree, as a widely used spatial search tree, can construct maps dynamically with multiple resolutions. And such a layered spatial representation provides great convenience for scan matching and robot localization.[12] Kraetzschmar et al.[13] proposed a method called probabilistic quadtree, which is commonly implemented in robot mapping with high accuracy and low computational resource occupation.[14] And such data structure also presents great convenience for robot navigation (trajectory planning).[15]

Furthermore, Octree,[7] similar to quadtree, is widely used in three-dimensional (3D) reconstruction and map construction. Einhorn et al.[16] provide an improved Octree algorithm called ND-Tree. This algorithm can build topological maps in adaptive scale.

Topological graph partitioning is a well-studied problem. Shi and Malik[17] and Von Luxburg[18] provide an excellent introduction to the spectral clustering algorithm, which gives the minimum cut of an undirected graph. And based on this algorithm, Vazquez-Martin et al.[5,19] propose a map segmentation method. On the other side, Tian et al.[2] provide an incremental map segmentation algorithm to improve performance and they also proposes some suggestive criteria to construct a similarity matrix.

Finman et al.[8] produced an incremental map segmentation method of RGB-D map. As objects (e.g. table, chairs, and computer) are a part of RGB-D map, they demonstrate the potential that map segmentation can not only apply to semantic SLAM and trajectory planning, but also in object recognition area.

This article is based on our previous research.[2] The method proposed in Tian et al.[2] leaves an unsolved problem that the result of map segmentation relies on the trajectory (and the key frame) of robots. Instead of feature-based map representation, we use quadtree-based map representation to reduce the computational cost. This method not only improves the performance but also solves the above problem. quadtree-based map segmentation algorithm can generate a robot state–independent, consistent map representation. And the incremental algorithm framework is inherited by Tian et al.;[2] in this way, we stabilized the computational resource occupation.

Constructing a robot map with quaternary tree

Quaternary tree (quadtree) is a data structure of spatial indexing. quadtree map can provide a hierarchical representation with different resolutions. In this way, quadtree can reduce the space complexity while remaining necessary details of the workspace. And algorithms based on quadtree still have an excellent time complexity. For example, time complexity of indexing and deleting operations in quaternary tree is $O(1)$. And constructing/adding a node in quadtree is highly dependent on the distribution of data. The time complexity of such operations is $O(h)$ in the best case and $O(n)$ in the worst case (h is the height of quadtree and n is the number of obstacles). The worst case supposes that the obstacles are dense everywhere. But the real workspace of the map is always sparse, and the average time complexity tends to be $O(h)$.

A demonstration of how quadtree splits the space is shown in Figure 3. Figure 3(a) presents a simple quadtree with three small obstacles in the space. The Z-axis of Figure 3(b) represents the layer, while the X–Y plane represents the nodes in each layer. Each rectangle represents a tree node.

The robot construction process of quadtree map can be described as in Algorithm 1, and an incremental version of this algorithm can be described in section "Incremental map construction/segmentation method."

1. Set the scale x_{map}, y_{map} and the resolution of the map in advance.
2. Read observation data from the sensor and calculate Obv_i in world coordinates.
3. Build the quadtree map with $Obv_1, \ldots, Obv_i$. If it is an incremental algorithm, update the quadtree map from Obv_i. The incremental algorithm

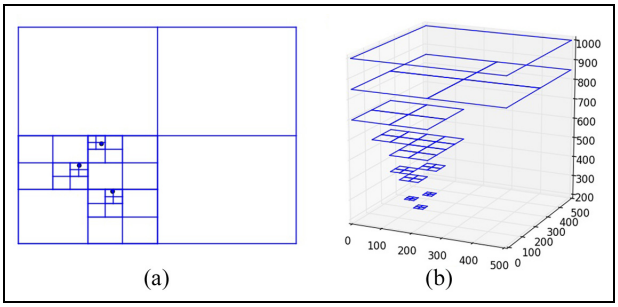

Figure 3. Demonstration of a quadtree in a simple map with three obstacles.

Algorithm 1. Constructing map using quadtree.

```
1: procedure Constructquadtree
2:     (obvList, x_range, y_range, thisLayer, maxLayer)
3:   /*Sensor observation results obvList*/
4:   /*The area scale of this node x_range, y_range*/
5:   /*x_range = (x_min, x_max)*/
6:   /*y_range = (y_min, y_max)*/
7:   /*Layer of this Node: maxLayer*/
8:   /*Max layer of Tree: maxLayer*/
9:   /*Return: quadtree Root Node v_r*/
10:  if obvList = [] then
11:    /* This is a leaf node */
12:    return None
13:  end if
14:  x_Middle = (x_min + x_max)/2
15:  y_Middle = (y_min + y_max)/2
16:  c1 = c2 = c3 = c4 = ∅
17:  for all ObstaclePoint ∈ obvList do
18:    if ObstaclePoint.x ≥ x_Middle then
19:      if ObstaclePoint.y ≥ y_Middle then
20:        c1.append(ObstaclePoint)
21:      else
22:        c4.append(ObstaclePoint)
23:      end if
24:    else
25:      if ObstaclePoint.y ≥ y_Middle then
26:        c2.append(ObstaclePoint)
27:      else
28:        c3.append(ObstaclePoint)
29:      end if
30:    end if
31:  end for
32:  for collection ∈ {c1, c2, c3, c4} do
33:    if collection ≠ ∅ then
34:      temp = constructquadtree(corresponding dataset,
         range, thisLayer + 1, maxLayer)
35:      addCorrespondingChild(temp)
36:    else
37:      addCorrespondingChild(None)
38:    end if
39:  end for
40:  Return: v
41: end procedure
```

is described in section "Algorithm implementation." And if it is not, then reconstruct the quadtree with Algorithm 1.

4. If the properties of the quadtree map reach the termination condition, stop reconstruction and perform the following operations:
   - Eliminate the node which presents undetected area.
   - Check the connectedness of each node. (This step is easy if the corresponding topological graph G is built. Just traverse graph G from a source node and mark all the nontraversed nodes.)
   - Eliminate the small connected component which is unconnected with the main branch. There are two simple methods to do this, which have been discussed in section "Algorithm implementation."

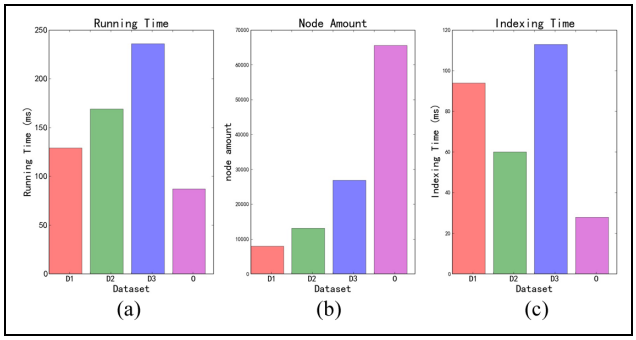

Figure 4. Performance comparison between quadtree and grid map: (a) running time, (b) node amount, and (c) indexing time.

Resolution of quadtree map is related to the map's scale x_{map}, y_{map} and the amount of layer l. The smallest unit of map is described in equation (1)

$$Area = \frac{x_{map} y_{map}}{4^l} \tag{1}$$

Experimental results show that the higher resolution needed in map construction process, the more advantage over the occupancy grid map. Figure 4 shows the performance analysis between quadtree and the occupancy grid map. Figure 4(a) shows the map construction time of several datasets. And Figure 4(b) shows the memory occupancy of quadtree and grid map. Figure 4(c) presents the indexing time of quadtree and grid map (10,000 times random indexing). Dataset d1 represents the test dataset shown in Figure 11 and dataset d2 represents the MIT-Csail-3rd dataset shown in Figure 11. Dataset d3 represents the Intel-Lab dataset[20] shown in Figure 18. By contrast, the dataset O represents the occupancy grid map's average performance.

Map partitioning based on spectral clustering

The next step of autonomous map segmentation is to cluster the leaf node of the pre-constructed quadtree into submaps. Spectral clustering is an excellent algorithm proposed by Shi and Malik.[17] As this algorithm not only vectorized deeply but can also be solved efficiently on a computer, it outperformed in poorly distributed datasets. These features make spectral clustering an excellent algorithm for robot working environment segmentation task.

In order to obtain a consistent segmentation result between human and robot, the cluster criterion has to be similar to the way human recognizes the environment. Humans tend to regard some connective space as a whole area and separate different areas (unlike in

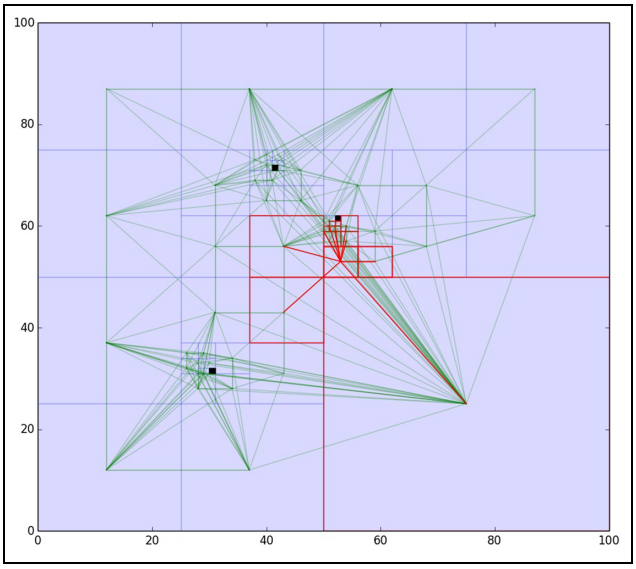

Figure 5. A sample of the corresponding topological graph G of the quadtree $\mathcal{T}$ ($\mathcal{T}$ is shown in Figure 3).

function, privacy, and so on) with obstacles like walls and tables. In this article, we provide two criteria to cluster map units (nodes in $\mathcal{T}$).

1. Two map units are connected if there are no obstacles in the line segment formed by the two units' center.
2. Weight of the connection relies on the map unit's layer. The higher the map unit's layer (root node has the highest layer), the higher the connection's weight. In other words, the larger node dominates the graph and the smaller node add some details. As shown in Figure 3, a node may establish edges to a far node if it has a higher hierarchy. The weight of e_{ij} (edge between the tree nodes i and j) is defined in equation (2).

Define a weighted undirected graph G as the corresponding topological graph of quadtree $\mathcal{T}$. The corresponding topological graph is also an important auxiliary data structure for autonomous navigation and trajectory generation. $V = \{v_1, v_2, \ldots, v_n\}$ is the collection of all leaf nodes of the quadtree $\mathcal{T}$, and $E = \{e_{ij}|i,j \leq n\}$ includes all the connection leaf nodes. The incidence matrix of G is also called "similarity matrix." In this article, it represents the degree of connection in this graph. A sample of such a graph G is shown in Figure 5, which was generated by the quadtree shown in Figure 3. The green node in Figure 5 represents the edge set E. And the red rectangle represents a specific node and the node it is connected to

$$e_{ij} = \begin{cases} 0 & d(v_i, v_j) \leq Th \\ 0 & \text{obstacle exists} \\ \dfrac{1}{\eta(2^{l_i} + 2^{l_j})} & \text{no obstacle}, d(v_i, v_j) \leq Th^3 \end{cases} \quad (2)$$

In order to make G a sparse graph, equation (3) defines the threshold Th. Edges longer than Th will not be added into the graph. It is just like a ϵ-nearest graph[18] with different layers. Equation (3) can establish the correct connection between nodes and avoid connecting too many nodes. Coefficient 1.05 is added to avoid floating point calculation error

$$Th_{l_i, l_j} = 1.05 \frac{\max(x_{map}, y_{map})(2^{l_i} + 2^{l_j})}{2^{l_i} 2^{l_j}} \quad (3)$$

In this way, a quadtree segmentation problem is transformed to a weighted undirected topological graph's segmentation problem. Spectral clustering is an excellent solution to the graph division problem. A brief example of such a graph is shown in Figure 5. This method can be solved efficiently and often outperforms the traditional methods like k-means.[18] First, generate the normalized Laplacian matrix $L_{i,j}^{sym}$ by function (4)

$$L_{i,j}^{sym} = \begin{cases} 1 & i = j, deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{deg(v_i) deg(v_j)}} & i \neq j, S_{ij} \neq 0 \\ 0 & otherwise \end{cases} \quad (4)$$

If k (the number of submaps) is given in advance, then find the k minimum eigenvectors $p_1, p_2, \ldots, p_k$ of $L_{i,j}^{sym}$. Define the auxiliary matrix P as follows

$$P = [p_1, p_2, \ldots, p_n] = \begin{pmatrix} p_{11} & p_{21} & \cdots & p_{k1} \\ p_{12} & p_{22} & \cdots & p_{k2} \\ \vdots & \vdots & \ddots & \vdots \\ p_{1n} & p_{2n} & \cdots & p_{kn} \end{pmatrix} \quad (5)$$

Regard each row of matrix P as the corresponding sample. Then k-means clustering result of those k samples is the spectral clustering result

$$Sample_j = \{p_{ij}|1 \leq i \leq k\}, \quad j \in [1, n] \quad (6)$$

In most cases, the value of k cannot be estimated. That means the spectral clustering algorithm does not know the number of submaps. So a score function should be defined to evaluate the performance of the clustering result. Silhouette coefficient may be a convenient method for evaluating a clustering algorithm's behavior. Let X_i, X_j be two clusters in the clustering result. Silhouette coefficient can be defined using equuations (7) and (8).[21,22] (Here $S(x_i, x_j)$ represents the Gaussian distance or other user-defined distance between two nodes.)

$$a_i = \frac{\sum_{m=1}^{|X_i|} S(x_j, x_m)}{|X_j|}, \quad j \neq i \tag{7}$$

$$b_i = \min \left(\frac{\sum_{m=1}^{|X_i|} S(x_j, x_m)}{|X_j|} \right), \quad j \neq i \tag{8}$$

For sample x_j, Silhouette coefficient s_i is given by

$$s_i = \frac{(b_i - a_i)}{\max(a_i, b_i)} \tag{9}$$

Algorithm implementation

In this section, some details of the algorithm implementation will be introduced. These details are conducted to reduce the time and space complexity of the algorithm. Through the runtime analysis, we figured out that the performance bottleneck lies in the following aspects:

1. In the construction process of graph G's corresponding matrix, the relevance between every pair of nodes in quadtree $\mathcal{T}$ needs to be calculated. That means the time complexity of this process is $O(n^2)$ (n is the number of nodes in the graph G).
2. Space complexity of the graph G's corresponding matrix is also $O(n^2)$. If the depth of the quadtree is too deep, a lot of memory will be utilized to store the corresponding matrix.
3. Calculating the eigenvalues and eigenvectors of a huge Laplacian matrix will take a lot time (even only calculate the first k of them).

For the aspects 1 and 2, this article introduces three additional criteria to reduce the computational complexity of this algorithm. Instead of the traditional matrix representation, DIA sparse matrix storage scheme (sparse matrix with diagonal storage)[23] is used to decrease the space complexity of the algorithm. Furthermore, the corresponding matrix construction process has a strong parallelism. So the task can be assigned to several CPU/GPU cores. In this way, the advantage of high-performance process on robots can be fully taken. As a last resort, this article produces an incremental version of map segmentation algorithm. This algorithm can restrain the growth of computational complexity efficiently. By combining the map segmentation and robot exploring process, the redundant computational resources can be fully utilized.

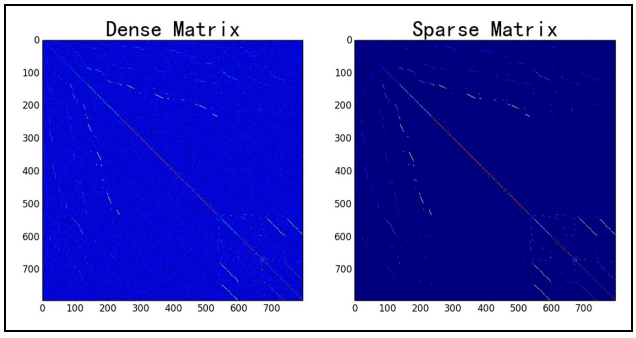

Figure 6. Comparison between the dense matrix and the filtered sparse matrix.

Sparse matrix representation

Because every map unit is a leaf node in quadtree, the corresponding matrix construction process can be regarded as a breadth first search (BFS) traversal of the quadtree $\mathcal{T}$. Consider that if the resolution of the map is high, the number of nodes will be huge. Since such a process may require a lot of computational resources, we provide three auxiliary rules to reduce the occupation of computational resources. In this way, we reduce the number of "edges" in graph G. In other words, make the corresponding matrix more sparse. A comparison between the dense matrix and the filtered sparse matrix is shown in Figure 6. The number of nodes (in other words, shape of the matrix) is 795 and the number of elements in the filtered sparse matrix is 59,462 (diagonal matrix; data come from a test experiment shown in Figure 11). That means the sparse matrix only takes 0.47% memory space of the dense matrix.

1. Each node has max relevance in itself (to avoid singularity).
2. For each node v_i, calculate only the relevance with node v_j if the hierarchy of v_j is lower than that of v_i.
3. For the node v_i with radius r, calculate only the relevance with nodes in the $2r$ range (according to the distance between centers).

Storing the corresponding matrix with normal schema will occupy a lot of memory space. However, if the definition of distance is well enough so that the matrix is sparse enough, DIA sparse storage schema (diagonal storage) will decrease the memory to 1% or less.

COO sparse matrix storage schema (coordinate storage) use three vectors I, J, V to represent the matrix. Lengths of I, J, V are the same, the vectors I, J represent the non-zero element index, and J represents the corresponding value. Schema of this storage is easy to construct but the efficiency in compression and

Algorithm 2. Generating sparse graph G.

```
1: procedure GenerateGraph(v)_r
2:   /* quadtree Root Node v_r*/
3:   /* Return Sparse Corresponding Matrix I, J, V*/
4:   count = 0;
5:   for all v_i ∈ v_r do
6:     if v_i.isLeaf and notv_i.isObstacle then
7:       v_i.num = count;
8:       count + = 1;
9:     end if
10:  end for
11:  for all v_i ∈ v_r do
12:    for all Layer l_j ≥ l_i do
13:      Let Th = 1.05 max(x_map, y_map)(2^l_i + 2^l_j) / (2^l_i 2^l_j);
14:      for all v_j ∈ v_r and v_j.layer≤v_i.layer do
15:        for all dist(v_i, v_j)≤Th do
16:          if DetectNoObstacle(v_i, v_j) then
17:            I.append(v_i.num, v_j.num);
18:            J.append(v_j.num, v_i.num);
19:            V.append(e_ij, e_ij);
20:          end if
21:        end if
22:      end for
23:    end for
24:  end for
25:  Return: I, J, V
26: end procedure
```

algebraic operation is sacrificed. So, after constructing the corresponding matrix, this matrix should be transformed into DIA storage schema to reach better efficiency.

The corresponding matrix construction algorithm is shown as follows:

Parallel algorithm

In the process of generating the corresponding matrix, most computational resources were occupied when calculating the relevance between map units. Actually, those calculations are independent. That means calculating the relevance of a pair of nodes needs nothing more than the quadtree $\mathcal{T}$. So these operations can be assigned to different cores on CPU (or GPU). By performing these computations parallelly, a lot of time will be saved (Figure 7). A parallel version of such computation can be described as follows:

1. First, give the quadtree $\mathcal{T}$ and the number of subprocesses n.
2. For each pair of nodes v_i, v_j, if the distance between v_i and v_j is less than the given threshold Th, then append the tuple (v_i, v_j) to a wait queue.
3. Count the number of tuples in the waiting queue and divide them into n parts.
4. Make a deep copy of the quadtree $\mathcal{T}$. (This step is important because different processes may require the same resource. Although this will not cause a deadlock, it may lead to performance loss. As the quadtree only takes a little space, deep copy operation's computational resources can be ignored.)
5. Initiate subprocesses. Pass the quadtree $\mathcal{T}$ and the index of subqueue into each subprocess.
6. In each subprocess, calculate the relevance of each pair of nodes in the subqueue. And remove pairs if its relevance value is zero.
7. Combine the results of each subprocess in the main process.

The value of n depends on the number of idle processor cores. Too many processes or unbalanced process may deteriorate the performance.

Incremental map construction/segmentation method

The algorithm mentioned above works effectively in the experiment. But there is always some problem. When the number of nodes increases with the robot exploring the workspace, the runtime of this algorithm is unstable and increases with the number of nodes. (Although the scale of the map is given, with the increase of the number of obstacles, the number of leaf nodes in the quadtree also rises. In the worst case, the number of leaf nodes is the same as that in an occupancy grid map.) Accordingly, this article presents an incremental map construction/segmentation method to stabilize the computational resource occupancy.

The incremental map construction/segmentation method involves the three components described below. The roles of these components while the robot is exploring the environment are shown in Figure 8.

1. Dynamic construction of the quadtree $\mathcal{T}$.
2. Generation of the graph G's local corresponding matrix.
3. Division and fusion between submaps.

Dynamic construction of quadtree. Dynamic quadtree construction problem can be explained as follows:

Given the root node v_r of the quadtree $\mathcal{T}$ and the observation result Obv_i, the robot parameter from robot sensor (Obv_i in absolute coordinates). Combine Obv_i into $\mathcal{T}$ without quadtree reconstruction.

In this article, we provide an algorithm defined as follows:

1. Define $\mathcal{V}_c$ as in equation (10). $\mathcal{V}_c$ is the collection of leaf nodes which might be modified at the moment i

$$\mathcal{V}_c = \{v | v \text{ in } \mathcal{T} \text{ and } v \text{ in sensor range}\} \quad (10)$$

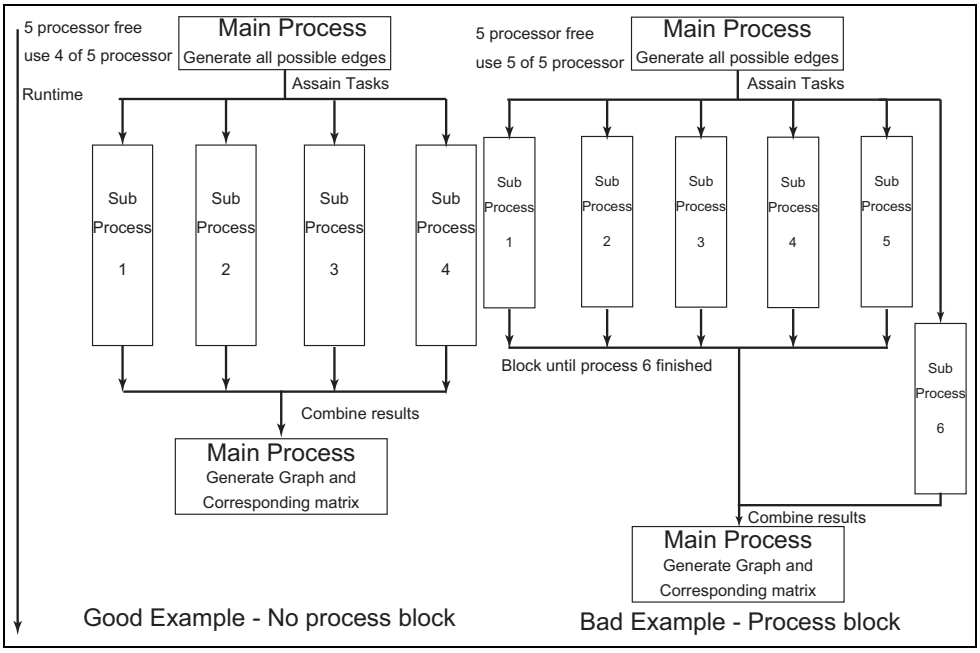

Figure 7. A chart of how multiprocessing improves the performance. This figure has been enlarged and transformed to a vector graph.

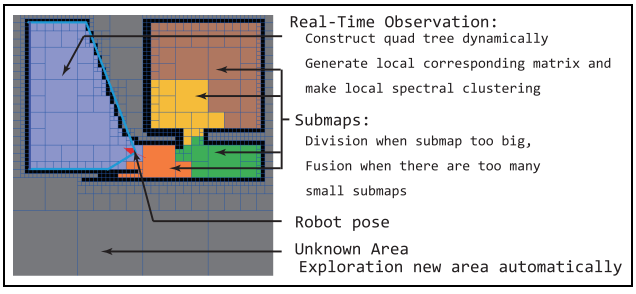

Figure 8. Types of nodes in the quadtree. When the robot is exploring the environment, different types of nodes are processed by different operations. Each operation corresponds to a subsection below. This figure has been enlarged and transformed to a vector graph.

2. Traverse $\mathcal{V}_c$. If the node $v_i \in \mathcal{V}_c$ does not contain obstacles in Obv_i, then delete all subtrees recursively from v_i until the following condition occurs:
   - The subtree includes nodes which is out of the sensor's range.
   - The subtree includes nodes which contains obstacles in Obv_i.
3. If $v_i \in \mathcal{V}_c$ represents an obstacle and such an obstacle is confirmed by Obv_i, then delete the corresponding data in Obv_i.
4. If there are still some obstacles remaining in Obv_i after steps 2 and 3, then for each $p_i \in Obv_i$ find the minimum subtree in $\mathcal{T}$ whose node is v_{ri}. Construct a subtree under v_{ri} recursively to combine p_i into $\mathcal{T}$.

The observation range of Obv_i can be inferred by robot poses and sensor parameters. In this article, the robot is equipped with a laser range finder. So the observation area relies on the sensor's viewing angle/range and the robot's pose. Sometimes there are obstacles in the sensor's range so that the map unit behind such an obstacle cannot be observed. To obtain the real observation area, for each node v_i, detect if there are obstacles between v_i and the sensor. If there exist obstacles, then the node v_i is actually not in the observation area.

The reason for deleting the subtree recursively in step 2 is that if the node v_i has been updated, nodes near v_i have a larger chance to be updated.

The effect of such optimization is shown in Figure 13. From this figure, in most of the time this method reduces the computational resources to 25% or less.

Local corresponding matrix generation. The algorithm above is designed to reduce the time complexity of updating the quadtree map. However, finding k small eigenvalues and eigenvectors still occupies a lot of computational resources. Based on a simple idea that only those nodes near the robot will be updated and those submaps which are far away from the robot, that is, not in the observation range, will not be calculated again, this article presents an algorithm to calculate

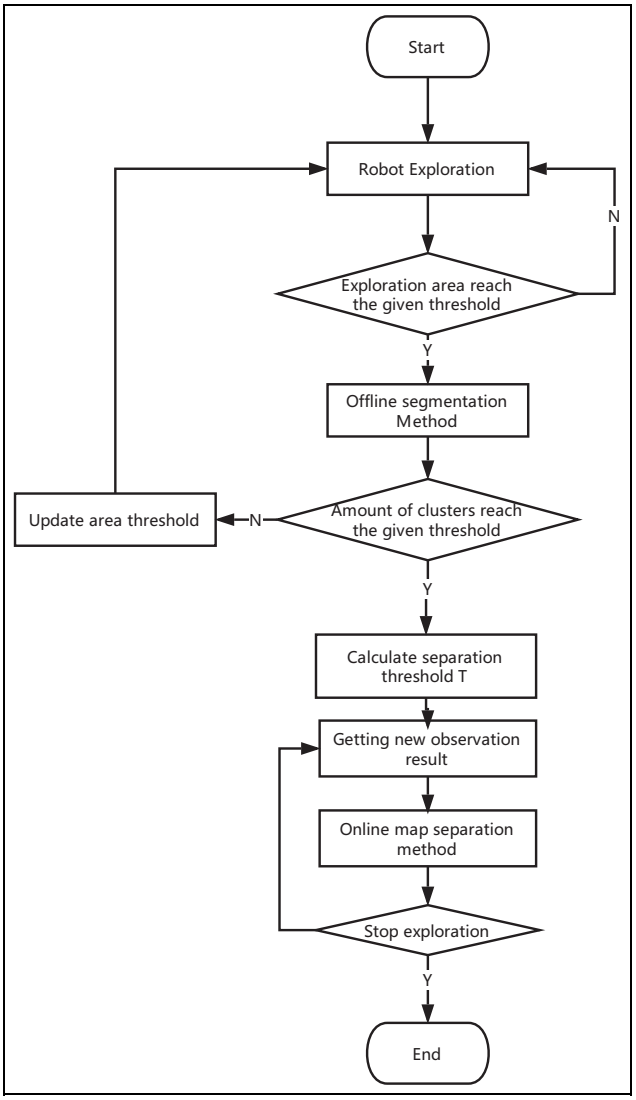

Figure 9. Online incremental map segmentation method framework. The offline part in this chart can be found in our previous paper.[2] This figure has been modified to a standardized form.

multiple small matrices' eigenvectors instead of calculating the global matrix's eigenvectors.

Define the relevance between submaps as follows: The submap S_a, S_b is relevant if there is at least one edge in the graph G connecting map units in S_a and S_b. Physically relevant map can be regarded as neighbor submap in space. And robot can go from S_a to S_b without passing other submaps.

An incremental map segmentation framework can be described as shown in Figure 9 from our previous paper:[2] At the beginning, let the robot explore the workspace until it stores enough information to make the first map segmentation. Then the robot continues exploring the new area (sometimes reviewing the old part) of workspace and use the incremental (online) version of the segmentation algorithm.

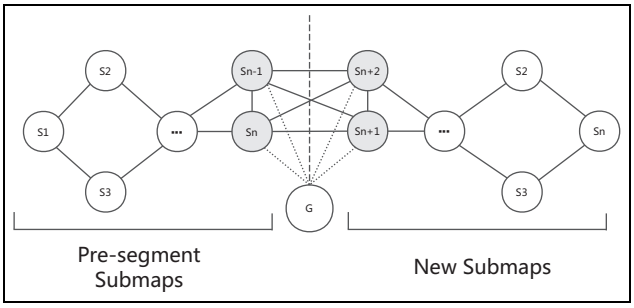

Figure 10. The relevant topological graph of submaps. The graph G in this figure represents the local corresponding graph G_{local}.

Assume that the robot has pre-segmented submaps $S_1, S_2, \ldots, S_i$ and the corresponding correlative observation results $Obv_1, Obv_2, \ldots, Obv_j$. The mission is to fuse those observation results into $\mathcal{T}$ and make new submaps $S_{i+1}, \ldots, S_n$.

A disadvantage of the above spectral clustering algorithm is that, each time the map updates, the graph G and its corresponding matrix will be reconstructed. As this process is the performance bottleneck of the whole algorithm, our goal is to reach a good enough clustering result without reconstructing the whole graph G (Figure 10).

The incremental version of the algorithm can be described as follows:

1. Combine $Obv_1, \ldots, Obv_j$ into the quadtree $\mathcal{T}$ with the algorithm described in subs_dynamicMapConstruction. Mark all the updated leaf nodes in this process.
2. Generate the local corresponding matrix G_{local} and perform spectral clustering of this graph. Submaps generated in this step are marked as $S_{i+1}, \ldots, S_n$.
3. Calculate the relevance between $S_{i+1}, \ldots, S_n$.
4. For observation $Obv_i \in \{Obv_1, \ldots, Obv_j\}$, if there exist an old submap $S_p \in \{S_1, \ldots, S_i\}$ and a new generated submap $\{S_{i+1}, \ldots, S_n\}$ together in Obv_i, then perform the following steps:
   - Put S_p and S_q in a temporary list.
   - Delete S_p and S_q in their corresponding collection.
5. After traversing all observation results, combine all the submaps in the temporary list into an auxiliary graph G_{temp}.
6. Perform spectral clustering of the graph G_{temp} and generate new submaps.

7. Combine such new submaps and all the remaining submaps. As the result of incremental map segmentation algorithm.

After constructing the corresponding graph G, an important step is to eliminate small connected components because such components will affect the performance. There are two methods that can find all the connected components of the graph G. The first one is based on BFS traversal of the map. And the second method utilizes the properties of the Laplacian matrix of the graph G.

If G is an undirected graph with non-negative weights. The number of zero eigenvalues is the number of connected components. And each eigenvector of the corresponding zero eigenvalue is the indicator vector of such components.[18] So after calculating the k smallest eigenvalues of the Laplacian matrix L_{sym}, just sort those eigenvalues by the number of non-zero elements. Each eigenvalue represents a connected component and the index of non-zero element is the index of nodes that are in this component.

Division and fusion between submaps. If the segmentation result is still far away from ideal, then, by presetting the threshold of an ideal scale of submaps, small submaps can be combined and a large submap and a too large submap will be divided again. And by defining the relevance between submaps, submap hierarchy corresponding to map G_{submap} can be generated. The clustering result of the graph G_{submap} can help find submaps which will be combined. This is not an easy work and it is better when the user determines which submap will be divided again or fuses with other graphs. Here we provide an easy criterion that helps in this process. Define the submap relative coefficient $r(s_p, s_q)$ as follows

$$r(s_p, s_q) = \frac{\sum_{v_i \in s_p,\, v_j \in s_q} weight(e_{v_i, v_j})}{\sum_{node \in (s_p \vee s_q)} area(node)} \tag{11}$$

The larger the $r(s_p, s_q)$ value is, the higher the possibility that the submaps s_p and s_q will merge into one submap.

Besides, the criteria about when a submap will divide can be taken as the maximum route length l_{s_p} of the submap (let all $e_ij \in s_p = 1$).

Simulation and real-world experiment

Simulation

This article first simulates the algorithm in a well-structured workplace. Then, such algorithms are tested on the MIT-Csail-3rd dataset and the Intel-Lab dataset from Radish.[20] The algorithm was implemented in a computer with an i7-4700MQ CPU with 8GB memory without any GPU acceleration. The algorithm was implemented in Python 2.7 with the third-party libraries SciPy and NetworkX, and run on windows 7 platform. The experimental result shows the correctness of the algorithm. And the computational resource occupation is excellent. The advantage of this algorithm over occupancy grid map is shown in Figure 4.

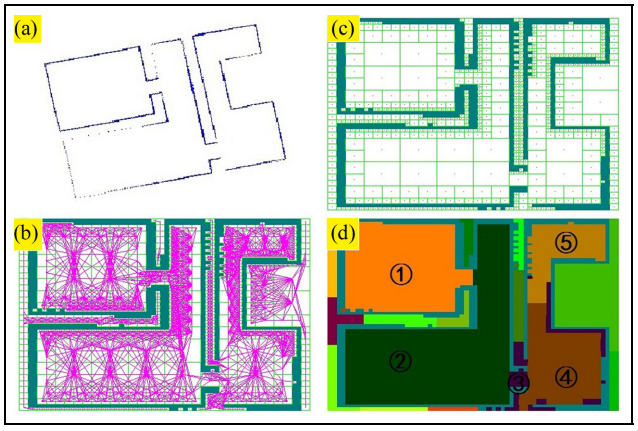

Figure 11. quadtree map construction and map segmentation in a well-structured workspace.

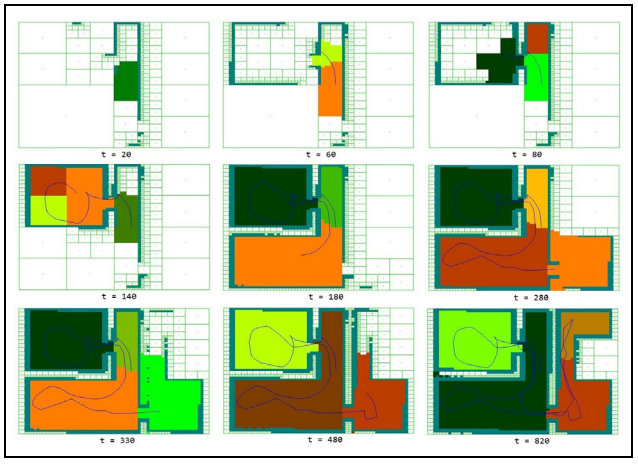

Figure 12. Process of incremental map construction and segmentation.

Simulation in a test environment. Figure 11 shows the main process of the map segmentation algorithm in a test environment. Figure 11(a) presents the raw laser observation results obtained by a laser range finder and the inertial measurement unit (IMU) of the robot. Figure 11(b) and (c) shows the corresponding topological graph G and the quadtree $\mathcal{T}$. Edges in subgraph B represent the non-zero relevance between leaf nodes in $\mathcal{T}$. The subgraph D shows the map segmentation result.

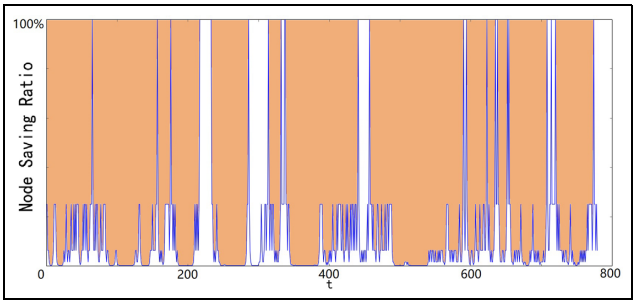

Figure 13. The chart of performance saving ratio.

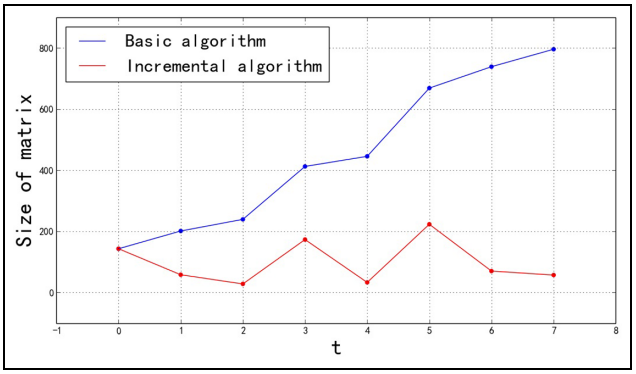

Figure 14. Performance comparison between the basic algorithm and the incremental algorithm.

From the result, our algorithm divided the main area of the map into five submaps which is high in cohesion and low in coupling. In this environment, the interior or exterior of the workspace is not defined. So the algorithm also divided the exterior of the map into several submaps.

Figure 12 shows the dynamic map construction and segmentation process. As shown in the figure, with the robot exploring its workspace (blue line represents its trajectory), the quadtree is constructed dynamically. And the submaps are divided and then combined by algorithm. Finally, the result is similar to the result shown in Figure 11. The clustering result may have some differences because the k-means result relies on the initiating value. If the clustering result's score reaches the preset threshold, such result will be accepted by the algorithm.

In Figure 12, an additional step is introduced in the algorithm. For each submap divided by the algorithm, judge if the submap intersects with the robot's trajectory. If the submap does not intersect with the trajectory, then this submap is invalid. This is an easy way to define the interior and exterior of the map.

When executing dynamically the quadtree construction algorithm, the robot will reconstruct only a part of the quadtree instead of rebuilding the whole tree. Define the performance saving ratio ps (percentage of reconstructed nodes) as follows

$$ps = \frac{reconstructed\ node\ amount}{all\ node\ amount} \tag{12}$$

It can be observed from Figure 13 that the performance saving ratio ps changes over time. When the robot's observation crosses the middle of the map, then the ratio is 100%, which means that the robot has to reconstruct all the nodes. In many cases, the robot just reconstructs less than 25% of the nodes. Compared to the KD-Tree algorithm which reconstructs the whole tree, our algorithm saves $1 - ps$ percent of computational resource usage.

Figure 14 shows the performance between the basic algorithm and the incremental algorithm. In this graph, the performance is evaluated by the size of the corresponding matrix of the graph G. As the main process of the algorithm is to construct and calculate the corresponding matrix, its size will be a good indicator of performance. It can be observed from Figure 13 that the basic algorithm's time complexity increases over time, whereas the incremental algorithm's performance remains stable.

Figure 15. quadtree $\mathcal{T}$ generated by MIT-Csail-3rd dataset.[20]

Simulation on MIT-Csail-3rd dataset. Figures 15–17 show the map construction and segmentation results on the MIT-Csail-3rd dataset from Radish.[20] Figure 15 presents the quadtree $\mathcal{T}$ and Figure 16 presents the topological graph G. Figure 17 presents the result of map segmentation when $k = 10$. The red line in this graph represents the connectivity between submaps. The experimental result shows that the submaps have high cohesion and low coupling. Moreover, the algorithm makes a balance of sizes in different subgraphs. The number of clusters was selected by Silhouette coefficient. However, by changing the k value and the score

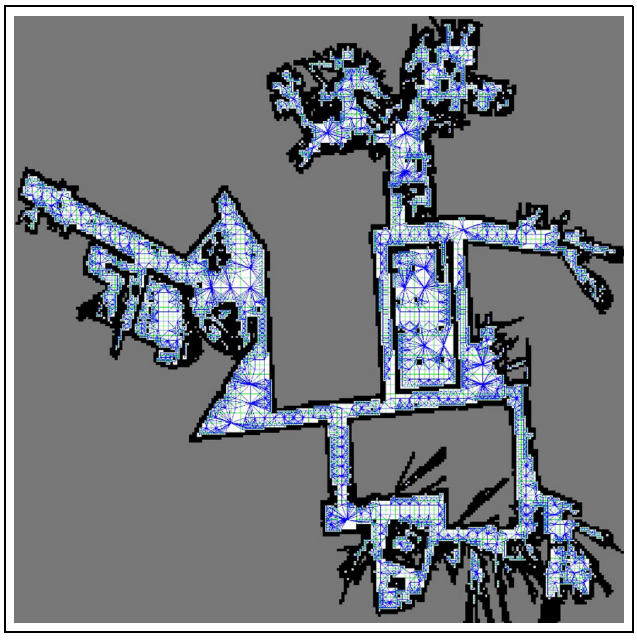

Figure 16. Topological graph G generated by $\mathcal{T}$ in Figure 15.

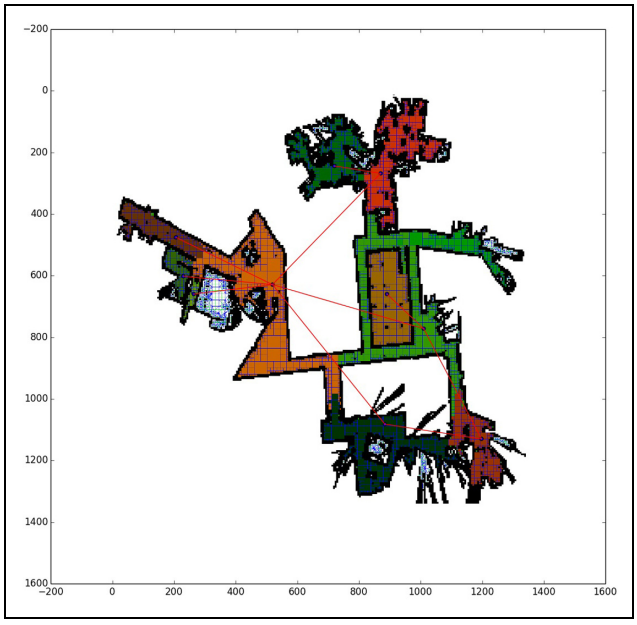

Figure 17. Autonomous map segmentation result from Figure 16.

function, the scale and shape of the cluster can be manually modified.

Simulation on Intel-Lab dataset. The result of map segmentation on the Intel-Lab dataset is shown on Figure 18. Compared to the above dataset, this one is more

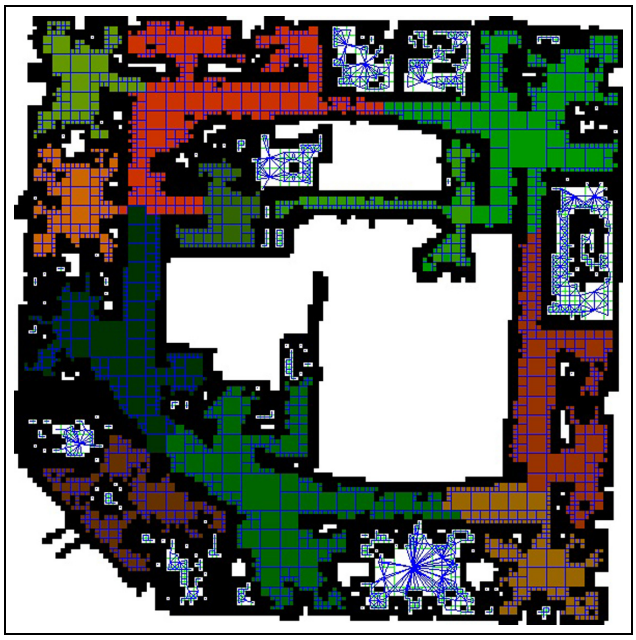

Figure 18. Another map segmentation on Intel-Lab dataset.[20]

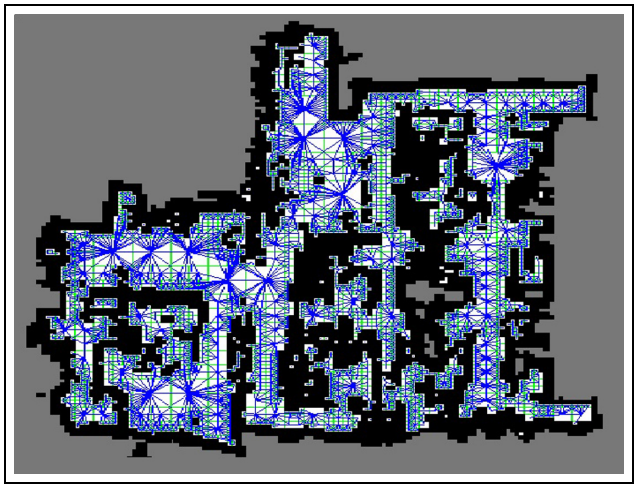

Figure 19. A brief introduction to the real environment. This map segmentation was performed by the algorithm described in our previous paper.[2] A more accurate map segmentation result is shown in Figure 22.

complicates and sounder in structure. Because the Intel-Lab and MIT-Csail-3rd datasets were recorded from a real environment, the segmentation results can represent the performance in a real environment. And the result shows the correctness of the algorithm. Although more than 17,300 ms is needed to make a segmentation of the whole map, it takes a little time to make a local segmentation. When the local corresponding graph has an average of 150 nodes, it takes only 43 ms to make a segmentation. So the robot can

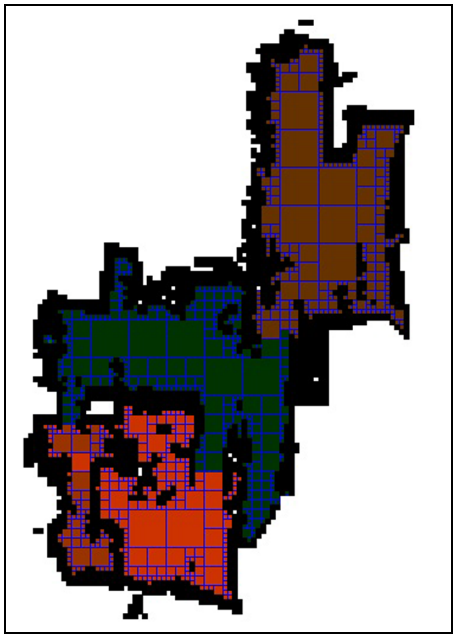

Figure 20. quadtree map built by the PeopleBot-SH robot in a real environment.

maintain a segmented map in real time and adjust the segment result in free time.

Real-world experiment

Finally, the algorithm was tested on a real robot. The map was constructed by a PeopleBot-SH mobile robot. And the sensor we used is a Hokuyo UST-10LX laser range finder. The environment comes from the 3rd floor of State Key Laboratory of Robotics and System, Harbin Institute of Technology. The environment is shown in Figure 19. And the corresponding quadtree map is shown in Figure 20. Figure 21 presents a dynamic, temporary map segmentation result when the robot explored nearly half of the environment. This figure includes four submaps which are very similar to the final segmentation result.

Result of map segmentation is shown in Figure 22. Although the complexity of this environment is very high and there are a lot of noises on the map, our algorithm can adapt the environment very well. The laboratory was divided by lattice room, debris, and partition. And the result of map segmentation fit in the structure of the room.

Discussion

The experiment above shows the correctness and efficiency of our algorithm. The algorithm can work not only under simple environments but also under several complex environments. Compared to the feature-based map segmentation method, our method does not rely on specific feature extraction method or environment consumption. Compared to occupancy grid map, our method provides a better efficiency, as shown in Figure 4. Other clustering methods, such as k-means algorithm, may be faster than spectral clustering (the construction of G is omitted), but those algorithms can hardly process poorly distributed data. For example, in

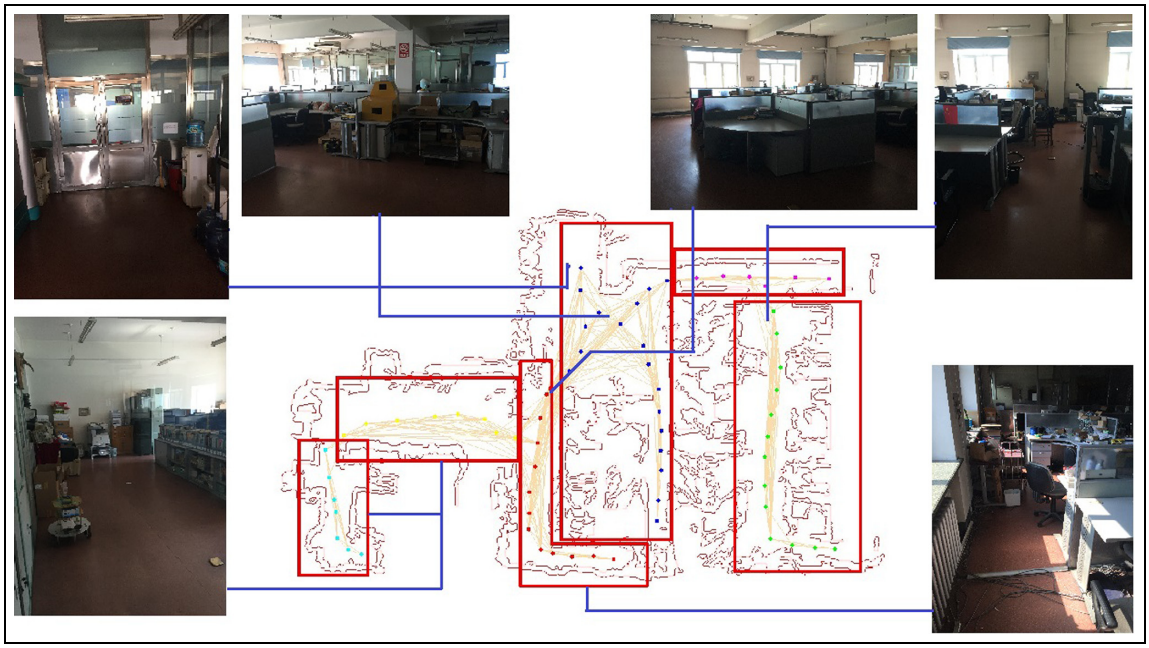

Figure 21. A temporary segmentation result built by the robot when exploring the environment.

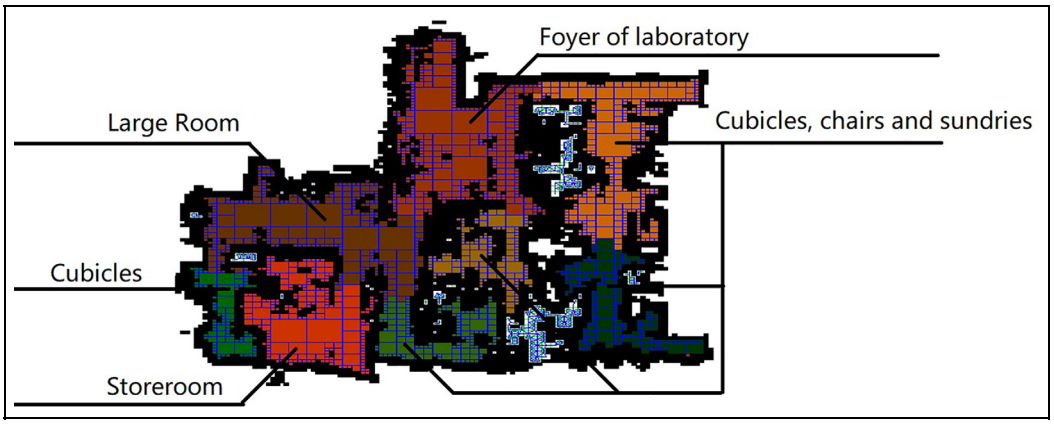

Figure 22. Map segmentation experiment in the same environment shown in Figure 21.

Figure 17, the green cluster is surrounded by the brown part, which is hard to archive with *k*-means algorithm because it "prefers" Gaussian data distribution.

Another advantage of the spectral clustering algorithm is that it keeps a balance between each cluster's shape. That means even the input data contain some noise like that shown in Figure 21 and some isolated small connected components; the algorithm still gives a good enough result and ignore the noise.

However, this algorithm is still not perfect. The border between the submaps is not certain. In Figure 18, the segmentation of a corridor is not determined by its semantic meaning but the size of the room it connects to. And the border between the submaps is rough, which does not agree with human recognition.

Moreover, as the algorithm only receives laser data, it cannot determine the "soft" obstacles like chairs and "hard" obstacles like walls. (A fast algorithm for obstacle detection (line and rectangle intersection) can be found in Tilove.[24]) As shown in Figure 22, if there are some tables and chairs in a room, the algorithm will consider them as walls and segment the room into several submaps. There is still a long way to go before semantic map segmentation.

Conclusion and future work

This article provides a quadtree-based map segmentation algorithm. Depending on the problem occurred in testing, this article proposes an incremental, parallel version of the algorithm. By testing this algorithm in multiple environments, the map segmentation algorithm shows its correctness and rapidity.

The future work will be focused on the consistency of segmentation results and the conditional independence between submaps. By researching such independence, the algorithm can do reasoning on submap hierarchy. In this way, the computational cost of loop closure, localization, and navigation algorithm can be reduced significantly.

Besides, this algorithm is a bottom-level part of the robot's environment recognition system. Our further research on environment reasoning and understanding will take a pre-segmented map rather than the whole map as the input to enhance the performance. Meanwhile, by taking the morphology and connectivity of rooms into account, it is possiable to do some high-level reasonings. As the state-of-the-art research attention to the concrete object classification, our research might solve the problem of organizing those objects in room hierarchy.

Acknowledgements

The datasets used in this study were obtained from the Robotics Data Set Repository (Radish). Thanks go to Cyrill Stachniss and Dieter Fox for providing data.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Nature Science Foundation of China (Grant Nos 61673136 and 61473103) and the Natural Science Foundation of Heilongjiang Province (Grant No. QC2014C072).

References

1. Leonard JJ and Feder HJS. A computationally efficient method for large-scale concurrent mapping and localization. In: *Proceedings of the robotics research-international symposium, vol. 9*, pp.169–178, https://marinerobotics.mit.edu/sites/default/files/Leonard99isrr.pdf

2. Tian Y, Wang K, Li R, et al. Fast map segmentation method based on spectral partition for robot semantic navigation. In: *Proceedings of the 2016 IEEE international conference on mechatronics and automation*, Harbin, China, 7–10 August 2016, pp.1059–1065. New York: IEEE.
3. Sjöö K. Semantic map segmentation using function-based energy maximization. In: *Proceedings of the 2012 IEEE international conference on robotics and automation*, Saint Paul, MN, 14–18 May 2012, pp.4066–4073. New York: IEEE.
4. Friedman S, Pasula H and Fox D. Voronoi random fields: extracting topological structure of indoor environments via place labeling. *IJCAI* 2007; 7: 2109–2114.
5. Vzquez-Martn R, Nez P and Bandera A. Less-mapping: online environment segmentation based on spectral mapping. *Robot Auton Syst* 2012; 60: 41–54.
6. Silberman N, Hoiem D, Kohli P, et al. Indoor segmentation and support inference from RGBD images. *Comput Vis* 2012; 2012: 746–760.
7. Hornung A, Wurm KM, Bennewitz M, et al. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Auton Robot* 2013; 34: 189–206.
8. Finman R, Whelan T, Kaess M, et al. Efficient incremental map segmentation in dense RGB-D maps. In: *Proceedings of the 2014 IEEE international conference on robotics and automation (ICRA)*, Hong Kong, China, 31 May–7 June 2014, pp.5488–5494. New York: IEEE.
9. Cadena C, Carlone L, Carrillo H, et al. Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. *IEEE T Robot* 2016; 32: 1309–1332.
10. Riazuelo L, Tenorth M, Di Marco D, et al. RoboEarth semantic mapping: a cloud enabled knowledge-based approach. *IEEE T Automat Sci Eng* 2015; 12: 432–443.
11. Pinies P, Paz LM and Tardos JD. CI-Graph: an efficient approach for large scale SLAM. In: *Proceedings of the 2009 IEEE international conference on robotics and automation*, Kobe, Japan, 12–17 May 2009, pp.3913–3920. New York: IEEE.
12. Yan F, Yan F, Wang K, et al. A bio-inspired scan matching algorithm for mobile robots in outdoor environments. *Assembly Autom* 2016; 36: 159–171.
13. Kraetzschmar GK, Gassull GP and Uhl K. Probabilistic quadtrees for variable-resolution mapping of large environments. In: *Proceedings of the 5th IFAC/EURON symposium on intelligent autonomous vehicles*, Lisboa, 5–7 July 2004. Amsterdam: Elsevier.
14. Chen Y, Shuai W and Chen X. A probabilistic, variable-resolution and effective quadtree representation for mapping of large environments. In: *Proceedings of the 2015 international conference on advanced robotics (ICAR)*, Istanbul, Turkey, 27–31 July 2015, pp.605–610. New York: IEEE.
15. Cocaud C and Jnifene A. Environment mapping using probabilistic quadtree for the guidance and control of autonomous mobile robots. In: *Proceedings of the 2010 international conference on autonomous and intelligent systems (AIS)*, Póvoa de Varzim, 21–23 June 2010, pp.1–6. New York: IEEE.
16. Einhorn E, Schrter C and Gross HM. Finding the adequate resolution for grid mapping—cell sizes locally adapting on-the-fly. In: *Proceedings of the 2011 IEEE international conference on robotics and automation*, Shanghai, China, 9–13 May 2011, pp.1843–1848. New York: IEEE.
17. Shi J and Malik J. Normalized cuts and image segmentation. *IEEE T Pattern Anal* 2000; 22: 888–905.
18. Von Luxburg U. A tutorial on spectral clustering. Technical Report, Max Planck Institute for Biological Cybernetics, Tübingen, 2006.
19. Vazquez-Martin R, Nunez P, Bandera A, et al. Spectral clustering for feature-based metric maps partitioning in a hybrid mapping framework. In: *Proceedings of the 2009, ICRA'09 IEEE international conference on robotics and automation*, Kobe, Japan, 12–17 May 2009, pp.4175–4181. New York: IEEE.
20. Howard A and Roy N. The robotics data set repository (Radish), 2003, http://radish.sourceforge.net/
21. Rousseeuw PJ. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J Comput Appl Math* 1987; 20: 53–65.
22. de Amorim RC and Hennig C. Recovering the number of clusters in data sets with noise features using feature rescaling factors. *Inform Sci* 2015; 324: 126–145.
23. Horn RA and Johnson CR. *Matrix analysis*. Cambridge: Cambridge University Press, 2012.
24. Tilove RB. Set membership classification: a unified approach to geometric intersection problems. *IEEE T Comput* 1980; C-29: 874–883.