Merging Two Octree Maps

Tetsuya Idota
University of Hawaii at Manoa
2500 Campus Rd
Honolulu, HI 96822
tidota@hawaii.edu

ABSTRACT

Octree map is three-dimensional map about occupancy. The environment is described as a tree structure where an internal node has eight children nodes. For its efficient data size, it is used by an autonomous mobile robot for obstacle avoidance. However, this mapping format entails a problem regarding map update. A global map is sometimes constructed from multiple local maps generated by mobile robots traversing in the environment. This process requires two octree maps to be merged, but it is not straight forward because of their arbitrary tree structure.

One existing algorithm expands nodes so that both octrees have the same structure. But this algorithm traverses entire nodes in the trees. Another existing algorithm avoids expansion by replacing subtrees. However, it only deals with Boolean values. In this project, a proposed algorithm is designed so that it can run as fast as the second algorithm and yet deals with occupancy probability values.

The two existing and proposed algorithms were analyzed in terms of an expected number of nodes to update. They were also simulated to measure average number of recursion calls and elapsed time. The results of simulation showed the improved efficiency by the proposed algorithm.

CCS Concepts

•**Mathematics of computing** → **Trees;** •**Theory of computation** → *Data compression;* •**Computer systems organization** → Robotics;

Keywords

Octree; Occupancy Map; Merging

1. INTRODUCTION

Occupancy map provides a probability for each unit volume to be occupied by objects. This occupancy information is useful for the navigation of an autonomous mobile robot in terms of obstacle avoidance. However, because of the three-dimensional space, the size of data required to be stored tremendously increases as the map resolution grows. Octree map is one of compression methods by using octree [5]. In this mapping format, space is composed of cubes with different sizes. When eight adjoin cubes have almost or exactly the same value, they are combined into a single bigger cube. This rule adjusts the resolution of mapping for each specific region, and the environment can be described with the smaller number of cubes avoiding loss of information.

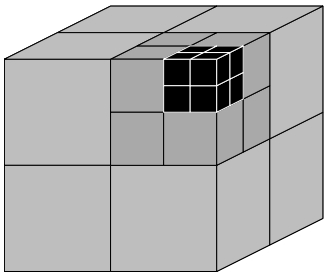

Figure 1: Octree Map: containing multiple levels of resolution.

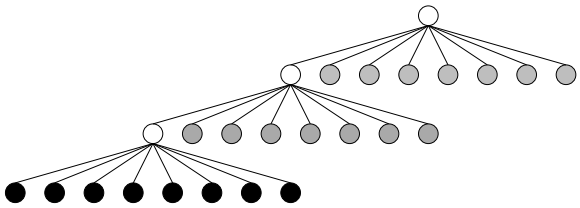

Figure 2: Octree Map: corresponding tree form.

One of the problems entailing this mapping format is integration of two octree maps. A global map is sometimes constructed with multiple local maps created by mobile robots and different types of sensors. In this project, two existing (Jessup's [2] and Pham's [4]) and one proposed algorithms were analyzed in terms of merging speed. Simulation codes were also written in Java, and average number of updated nodes and elapsed time were measured.

2. OCTREE MAP AND MERGING TREES

Figure 1 shows an example of octree map. Smaller cubes provides higher resolution, and larger ones keeps data size small by assigning the same value to a large space. Octree represents these divided spaces as tree nodes. Figure 2 shows the tree formation corresponding the cubes in Figure 1. The color of a leaf corresponds that of a cube which the leaf represents. The root node represents the entire space. A node has eight children if its corresponding cubical space is divided. These children are connected to their parent node by reference [1]. The maximum tree height determines the minimum size of a cube. When the maximum height is three, the minimum size will be $1/2^3$ of the size of the entire cube. Each node has a value for its corresponding cube, e.g., a probability, logodds, Boolean value, and so on.

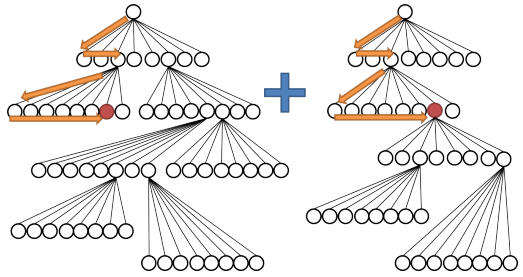

Figure 3: Traversing to Merge Two Trees.

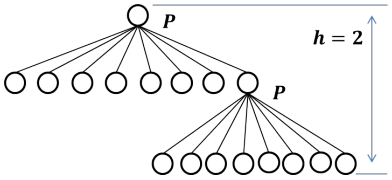

Figure 4: Assumptions for Expected Value: probability of expansion P and the maximum tree height h. The figure shows a case of $h = 2$.

A mobile robot constructs and updates its map dynamically by updating the structure and values of the corresponding octree.

As shown in Figure 3, a merging algorithm traverses two trees simultaneously starting at their root nodes, and each pair of corresponding nodes are combined by calculating their containing values. However, when either one is a leaf and the other one has children, the tree structure needs to be updated by a specific process.

In the following chapters, each algorithm is discussed and analyzed in terms of merging speed.

3. ANALYSIS OF ALGORITHMS

The existing algorithms (Jessup's and Pham's) and the proposed algorithm were analyzed. Since the merging speed is determined by the number of nodes to update, the expected number of nodes was evaluated. To get the expected value, there are the following assumptions. As shown in Figure 4, there are two assumptions: P and h. P represents a probability that a node is expanded and has children, and h represents the maximum height of a tree. Based on these assumptions, the total number of nodes in a specific octree whose maximum height is h is defined as a random variable $X(h)$.

$$X(h) = 1 + I\{expanded\} \sum_{i=1}^{8} X_i(h-1) \tag{1}$$

$$X(1) = 1 + 8I\{expanded\} \tag{2}$$

$I\{expanded\}$ is an indicator. It is one when the root node is expanded. Otherwise, it is zero. X_i is the number of nodes in a subtree. Thus, $X(h)$ is expressed by summation of the root node and nodes in its subtrees. The base case $X(1)$ says that there could be the root node and its leaves.

The expected number of nodes in an octree is expressed as $E[X(h)]$ as follows.

Algorithm 1 MERGE-OCTREE(n_1,n_2)

```
1: if n1 or n2 has children then
2:    if n1 is a leaf then
3:       EXPAND(n1)
4:    else if n2 is a leaf then
5:       EXPAND(n2)
6:    end if
7:    for i = 1 to 8 do
8:       MERGE-OCTREE(n1.children[i],n2.children[i])
9:    end for
10: else
11:    n1.key ← n1.key + n2.key
12: end if
```

Algorithm 2 EXPAND(n)

```
1: add 8 nodes to n
2: for i = 1 to 8 do
3:    n.children[i].key ← n.key
4: end for
```

$$\begin{aligned} E[X(h)] &= 1 + E[I\{expanded\}] \sum_{i=1}^{8} E[X_i(h-1)] \\ &= 1 + 8PE[X(h-1)] \end{aligned} \tag{3}$$

$$\begin{aligned} E[X(1)] &= 1 + 8E[I\{expanded\}] \\ &= 1 + 8P \end{aligned} \tag{4}$$

where $P = E[I\{expanded\}]$. Since expansion of nodes at the same depth is assumed to be equally likely, $E[X_i(h-1)] = E[X(h-1)]$. Finally, the recursive form above is solved as followes.

$$E[X(h)] = \sum_{i=0}^{h} (8P)^i = \frac{(8P)^{h+1} - 1}{8P - 1} \tag{5}$$

In the following subsections, each algorithm is discussed.

3.1 Jessup's Algorithm

Jessup et. al designed a method which expands specific nodes so that both trees have the same structure [2] [3]. Algorithm 1 shows pseudocode of Jessup's algorithm. As shown in the code, a recursion occurs when either node has children at line 8. If the other node is a leaf, it is expanded at lines 3 and 5 by Algorithm 2, which takes constant time. The update of node value at line 11 also takes constant time. Thus, the running time depends on the number of calls of the recursion. Since this call happens at every node to update, the total number of recursion is equivalent to the number of nodes updated by the algorithm. This number is eventually equivalent to the nodes in the resulting tree.

Let $X_{T_1}(h)$ and $X_{T_2}(h)$ be the number of nodes in the two trees to merge respectively. Then, the expected number of nodes updated by the algorithm $E[X_{Jessup}(h)]$ is

$$\begin{aligned} E[X_{Jessup}(h)] &= E[X_{T_1 \cup T_2}(h)] \\ &= E[X_{T_1}(h)] + E[X_{T_2}(h)] - E[X_{T_1 \cap T_2}(h)] \\ &= 2\frac{(8P)^{h+1} - 1}{8P - 1} - \frac{(8P^2)^{h+1} - 1}{8P^2 - 1} \end{aligned} \tag{6}$$

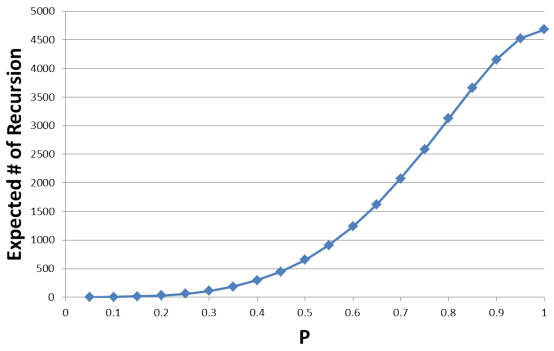

Figure 5: Expected Number of Nodes Updated by Jessup's Algorithm for Each P. This is a case for $h = 4$.

Algorithm 3 MERGE-OCTREE2(n_1,n_2)

```
1: if both n1 and n2 have children then
2:   for i = 1 to 8 do
3:     MERGE-OCTREE2(n1.children[i],n2.children[i])
4:   end for
5: else if n1 has children then
6:   if n2.key = false then
7:     replace n1 with n2
8:   end if
9: else if n2 has children then
10:  if n1.key = black then
11:    replace n1 with n2
12:  end if
13: else
14:  n1.key ← n1.key AND n2.key
15: end if
```

$X_{T_1 \cap T_2}(h)$ is the nodes existing in both octrees. That means, they are expanded in both trees. Hence, the expansion probability for each node is P^2. As a result, the expected number of the common nodes is $\frac{(8P^2)^{h+1}-1}{8P^2-1}$. Figure 5 shows a graph representing the expected number of nodes updated by the algorithm. h is 4 and P changes at intervals of 0.05.

This approach is simple to implement but it could run slow when many nodes are to be newly expanded.

3.2 Pham's Algorithm

Pham et. al provided a different approach which moves subtrees from a tree to the other one [4]. This algorithm is specialized for cutting operation of an object. If a cube filled with material, a corresponding node is assigned with true. Otherwise, it has false. An octree map in this field describes the shape of an object after applying a specific cutting operation. When two octree maps are merged, the resulting map shows the resulting object applied with two corresponding operations. Algorithm 3 shows pseudocode of the algorithm. The algorithm updates the first tree by using the second tree. After the update, the first tree is eventually the resulting tree.

Update of nodes is based on the rules of cutting operation. If a space is indicated as open space by either octree map, the result is also open. Otherwise, if a space is divided to provide detailed map by either one, the result also has the same divided space.

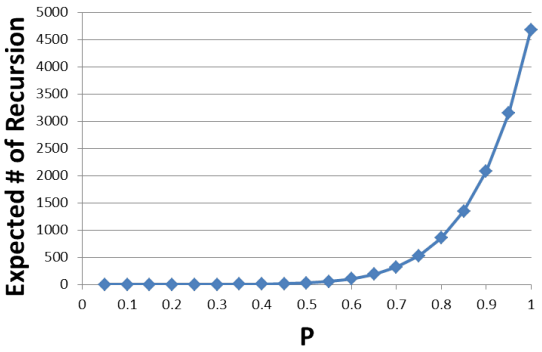

Figure 6: Expected Number of Nodes Updated by Pham's Algorithm for Each P. This is a case for $h = 4$.

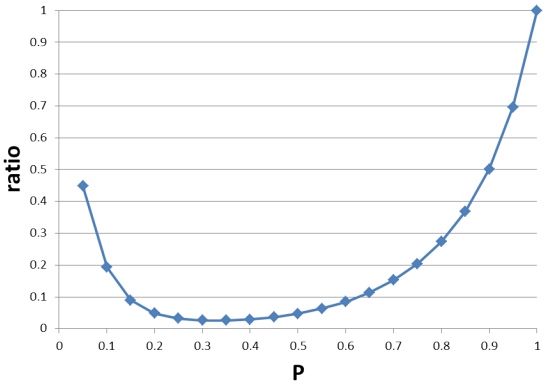

Figure 7: Ratio of $\frac{E[X_{Pham}(h)]}{E[X_{Jessup}(h)]}$ for Each P. This is a case for $h = 4$.

Line 7 is executed when a node in the first tree has children and the corresponding node in the second one is false. This line overwrites the node in the first tree with false. Line 11 is executed when a node in the first tree is true and the other has children. This line moves the children or subtree from the second tree to the first one. These operations only require update of links to nodes, taking constant time. At line 3, it recurses only when both nodes have children. This leads that the algorithm traverses only the common nodes. Hence the expected number of nodes updated by the method $E[X_{Pham}(h)]$ is expressed as follows.

$$E[X_{Pham}(h)] = \frac{(8P^2)^{h+1} - 1}{8P^2 - 1} \tag{7}$$

Figure 6 shows a graph of the expected number in case of $h = 4$. It is smaller than that of Jessup's algorithm. Figure 7 shows the ratio of the expected numbers of Pham's algorithm against Jessup's in the same case.

$$ratio = \frac{E[X_{Pham}(h)]}{E[X_{Jessup}(h)]} \tag{8}$$

As shown in the graph, there is a significant difference when P is not very close to either zero or one.

Thus, this method can avoid node expansion and reduce the number of nodes to update. But the leaf value is Boolean,

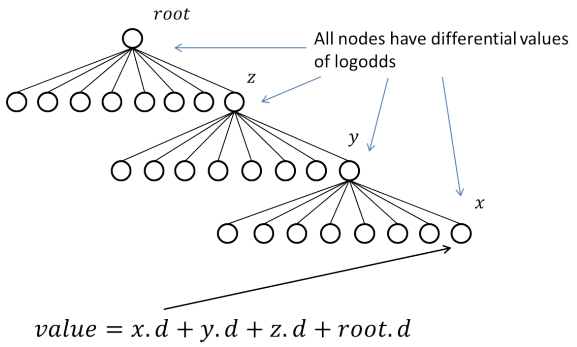

Figure 8: Differential Value of Log Odds

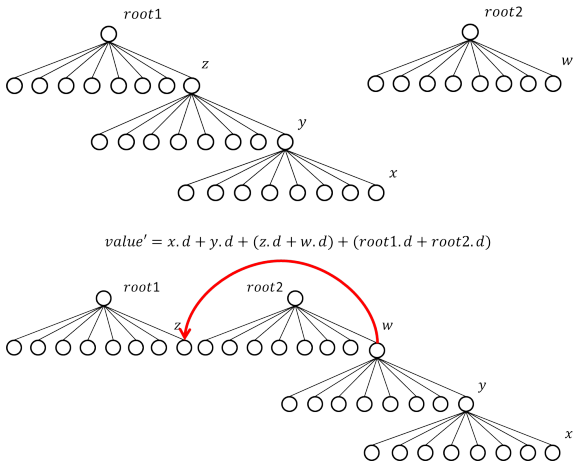

Figure 9: Update of Differential Log Odds. Top: Update of Corresponding values. Bottom: Replacing a Subtree.

and it is not feasible to extend it for use of occupancy probability.

3.3 The Proposed Algorithm

In this project, a new value representation and merging algorithm are proposed. The proposed algorithm deals with occupancy probability and yet uses similar approach as Pham's algorithm does, leading potentially better efficiency against Jessup's algorithm.

For this algorithm, an octree has a value format different from Jessup's. As shown in Figure 8, every node has a differential value of log odds. The actual probability of occupancy for a leaf is calculated by summation of the differential values along its path from the root.

Figure 9 shows update of node values in two cases. The first octree eventually becomes a resulting tree by updating it with the second octree. In the top case, when a node in the first tree (node z) has children and the corresponding node in the second tree (node w) does not have, it stops recursion for this path. In the bottom case, when a node in the first tree (node z) is a leaf and the other (node w) has a subtree, the leaf is replaced with the subtree. In both cases, the algorithm only needs to access the common nodes of the two trees. Algorithm 4 shows the details of process. It updates node values at each accessed node and it recurses only when

Algorithm 4 MERGE-OCTREE3(n_1,n_2)

```
1: if both n1 and n2 have children then
2:    n1.d ← n1.d + n2.d
3:    for i = 1 to 8 do
4:       MERGE-OCTREE3(n1.children[i],n2.children[i])
5:    end for
6: else if n2 has children then
7:    n2.d ← n1.d + n2.d
8:    replace n1 with n2
9: else
10:   n1.d ← n1.d + n2.d
11: end if
```

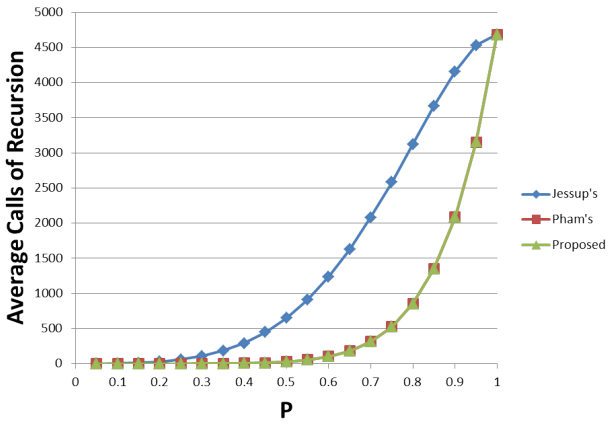

Figure 10: Average Calls of Recursion by Algorithms.

both nodes have children. The traversal is the same as that of Pham's algorithm. Thus, the expected number of nodes to update $E[X_{Proposed}(h)]$ is equivalent.

$$E[X_{Proposed}(h)] = E[X_{Pham}(h)] = \frac{(8P^2)^{h+1} - 1}{8P^2 - 1} \quad (9)$$

4. SIMULATION

To analyze the algorithms empirically, average calls of recursion and elapsed time were measured on simulation. The simulation code was written by Java. Octree was implemented as linked nodes. A parent node has an array holding references to its child nodes, making the time to access a child/parent node constant.

The algorithms were simulated with the following settings: $h = 4$ and P changing between 0.05 and 1 at intervals of 0.05. For merging, two trees are generated by randomly expanding nodes. Then, they are merged and elapsed time of merging is measured. This process was repeated 10,000 times for each P to calculate an average time. For accessing a leaf, a randomly chosen leaf value in the resulting octree is return for 1 million times, and the entire elapsed time is measured. This process was repeated 10 time for each P to generate an average time.

Figure 10 shows the average number of recursion calls. Figure 11 shows the ratio of recursions by the proposed algorithm against Jessup's algorithm. These results follow the

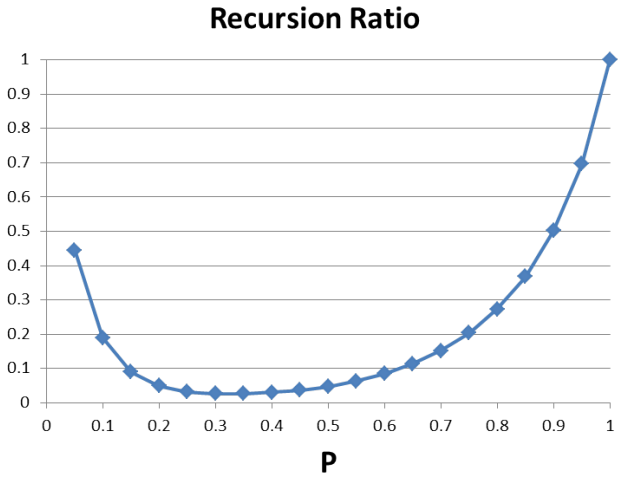

Figure 11: Ratio of Recursions by the Proposed Algorithm Against Jessup's Algorithm.

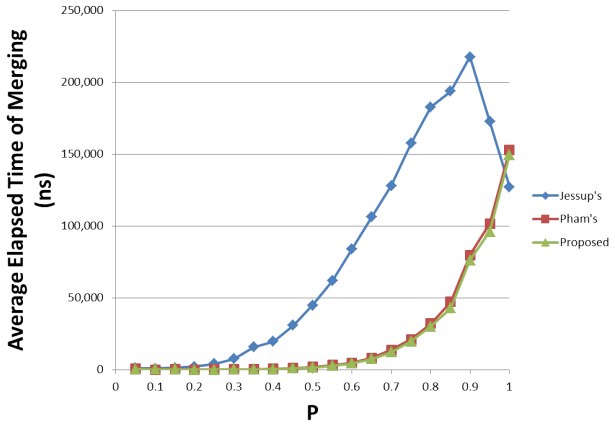

Figure 12: Elapsed Time of Merging Two Octrees.

analysis and verifies that the proposed algorithm requires less recursions.

Figure 12 shows average elapsed time of merging by each algorithm. It shows that the proposed algorithm merged octrees faster than Jessup's algorithm when P is less than one. The result of Jessup's algorithm had a peak. This might have been caused by expansion process which is not used by other algorithms.

Figure 13 shows average time to access a leaf. It shows there was no significant difference between the proposed and Jessup's algorithms. This indicates the new format in the proposed algorithm does not deteriorate accessing speed.

5. CONCLUSIONS

Jessup's, Pham's, and the proposed algorithms were analyzed in terms of merging speed. By setting assumptions on expansion, it was shown that the expected number of nodes updated by the proposed algorithm was equivalent to that of Pham's algorithm which is lesser than Jessup's. When the probability P is not close to zero or one, the difference

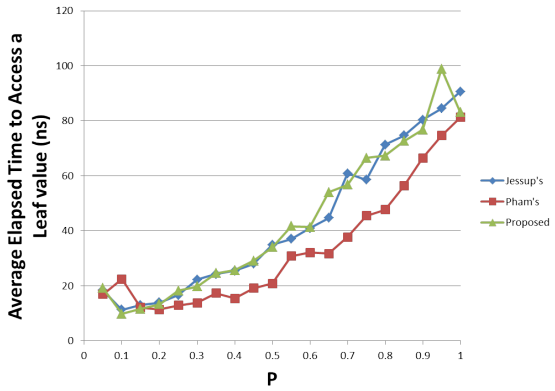

Figure 13: Elapsed Time to Access A Leaf.

was more significant.

Each algorithm was also simulated and the average calls of recursion and elapsed time were measured. The average recursion calls actually matched with the expected number of nodes, discussed in the analysis. The average elapsed time for merging also showed that the proposed algorithm ran significantly faster than Jessup's did. In addition, the difference in time to access a leaf value was not significant, indicating that the proposed algorithm does not undermine accessing speed.

In conclusion, the proposed algorithm improves the performance of merging. It speeds up the process to merge two randomly generated octrees, keeping access to a leaf value as fast as the original way.

6. REFERENCES

[1] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[2] J. Jessup, S. N. Givigi, and A. Beaulieu. Merging of octree based 3d occupancy grid maps. In *Systems Conference (SysCon), 2014 8th Annual IEEE*, pages 371–377. IEEE, 2014.

[3] J. Jessup, S. N. Givigi, and A. Beaulieu. Robust and efficient multirobot 3-d mapping merging with octree-based occupancy grids. 2015.

[4] T. T. Pham, Y. H. Kim, and S. L. Ko. Development of a software for effective cutting simulation using advanced octree algorithm. In *Computational Science and its Applications, 2007. ICCSA 2007. International Conference on*, pages 324–334. IEEE, 2007.

[5] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2, 2010.