Tunable computing Slam navigation environments

Trang Hoang Thuy Nguyen¹,², S V Shidlovskiy¹

¹ National Research Tomsk State University, Tomsk, Russia
² Vietnam Maritime University, Hai Phong, Viet Nam
E-mail: trannght@vimaru.edu.vn

Abstract. In many cases, to solve applied problems, the robot needs to know its real location, which is most often different from the data stored in the on-board system. For unmanned robotic devices, as well as for ground-based robots, it is most efficient to use local navigation algorithms, which consist in determining the coordinates of the device with respect to a certain starting point. The paper discusses localization algorithms provided that the map of the area is known in advance. Particular attention is paid to the Monte Carlo localization method because of several advantages. The paper presents an example of modeling the algorithm operation.

1. Introduction

Nowadays, simultaneous localization and mapping approach (SLAM) has become one of the most modern technical methods used for mobile robots to create maps in unknown or inaccessible locations and to update maps to a specific area while tracking your current location and distance. We built a highly structured system using mathematical models in conjunction with the Octomap environment to navigate a mobile robot.

2. Slam

In mapping and robotic navigation, simultaneous localization and mapping (SLAM) is one of the problems that scientists are interested in to create or update maps. The unknown environment also tracks the position of the internal agent. To overcome and improve positioning over short distances, synchronization simultaneous localization and mapping (SLAM) have made significant progress in the development of science and technology. Navigation can be divided into three main areas: the first is localization, the second is mapping, and the third is path planning. Localization performs the task of accurately determining the current position of the robot in the environment. Mapping is responsible for collecting sensor data and storing it in a form that is convenient for further processing. Path planning searches a route in an environment where the robot will move from start to goal using the data collected during a mapping process. These three tasks are dependent on each other. For example, localization will more accurately determine the location of the object on the map, and a good map will increase localization accuracy. In order to solve these dependencies better, simultaneous localization and mapping (SLAM) methods have been developed.

SLAM combines localization and mapping processes to make the combination both faster and more efficient. SLAM algorithms can currently solve specific environmental or hardware constraints. For example, lidar sensors and images combine Slam methods [1].
3. Octomap
Some applications for robots require a three-dimensional model of the environment. Although 3D maps are an integral part of many robotic systems, there are still a few flaws that are rarely deployed in a reliable and efficient system. The absence of such a deployment leads to the cloning of basic software components. Therefore, it can be considered as an important point in the study of robots.

Octomap is good for creating 3D probability maps. The matching process allows us to integrate measurements with a certain probability to explain sensor infiltration and other exceptions. In addition, it is a very efficient and relatively fast memory. Octomap is the process of mirroring with multiple resolutions; by changing the resolution and range we get a significant rate of change. In addition, the Octomap is a casting rail that allows you to distinguish between unidentified, free, and busy voxels that do not have other display processes (point clouds, elevation and multi-level maps, and surface).

The map to be navigated by the path planner will be in the form of an OctoMap, published in ROS and connected the Octomap server. The OctoMap library uses an octree based cubic grid system to represent three-dimensional space, with data structures and mapping algorithms provided in C++.

An octree is a hierarchical data structure for spatial subdivision in 3D (Meagher 1982; Wilhelms and Van Gelder 1992). Each node in an octree represents the space contained in a cubic volume, usually called a voxel. This volume is recursively subdivided into eight sub-volumes until a given minimum voxel size is reached (Figure 1) [2]. Cubes with octant components that are occupied or open can be saved as a single element. This results in a significant reduction in the size of the file on the map with large areas where all vertices are either occupied or open. The minimum voxel size determines the resolution of the octree. Since octree is a hierarchical data structure, a tree can be cut at any level to get a more complex part if the inner nodes are supported accordingly.

![Figure 1. The structure of an octree and the cube shaped space it represents.](image1)

This allows Octomap to efficiently store models of 3D environments that can be quickly sent between ROS robots or between nodes on a single ROS device. This compression affects the Octomap messages that are stored and sent, but there is no real effect on the operation of the scheduling algorithm; Octomap consists of cubes of equal size that corresponds to the map resolution (Figure 2).

![Figure 2. OctoMap storing one occupied vertex (black) within a larger free (white) space.](image2)
In the basic form, an octree can be used to model a logical attribute. In the context of mapping robots, volume is usually taken into account. If a certain volume is measured as occupied, the corresponding node in the octree will be initialized. Any uninitialized node could be free or unknown in this Boolean setting [2]. The OctoMap plugin for ROS can open an OctoMap from a file or create one of the point clouds. The point cloud is a list of coordinates in which a detector such as a laser scanner has detected a surface. When creating an OctoMap from a point cloud, it is possible to set any resolution that is considered appropriate for the task at hand, depending on factors such as the computing power available and the size of the robot. When loading maps from a file, a minimum cube size can be set to the size of a file, which enables faster processing of non-essential high resolution maps [3].

Here, the algorithm I am going to do is divided into point clouds, each of which contains four cubes, dividing the cubes until they stop dividing. The occupied areas will receive a value of 1 (black), the occupied area will receive a value of 0 (white) (Figure 3).

\[ \text{Figure 3. Octree structure: the white nodes of the octree leaves have a free state, eight gray leaf nodes are occupied by the status.} \]

\[ \text{4. High-performance reconfigured systems} \]

At present, with the continuous development of science and technology, the construction and design of intelligent control systems for various industrial automation objects to ensure coordination of their work are efficient with well-defined algorithms that are one of the important tasks of automation of production processes. In recent years, instead of using small and medium-sized microchips, people have switched to software logic devices, followed by large integrated circuits that are used as computer cores on industrial computers or in programmable logic controllers. Automating decision-making requires the use of modern mathematical methods and new technical means. Currently, when building control devices, boolean models are widely used, but some developers avoid direct use of logical algebraic functions. This is explained by the fact that considerable difficulties arose for the development of efficient algorithms for calculating complex functions of the algebra of logic of multiple arguments, and this also requires more intensive research and more thorough solving of them. The growing interest in the functions of the algebra of logic and its computational problems led to the creation of the theory of homogeneous structures. The paper proposes a logical model that can be adapted to a specific class of Boolean formulas. This model allows to solve the problem of the computing system of Boolean formulas from ordered and unordered iterative classes, as well as the class of Boolean formulas in the order of repetition and Boolean systems with and without arguments.

In mathematics and mathematical logic, Boolean algebra is a branch of algebra in which the values of variables are true and false truth values, usually denoted by 1 and 0, respectively. Instead, elementary algebra, where the variables are numbers and operation are simple addition and multiplication, the basic operations of Boolean algebra are the conjunction and are designated as $\land$, designated as a disjunction or $\lor$, and negation, and referred to as $\neg$.

The function whose values and arguments are binary is called Boolean.
Boolean functions (BFU) are defined as truth tables. Such a BFU entry is a formula that uses logical operators called the Boolean formula (BF).

In this article, we consider the formula in the basis of \{AND, OR, NOT\}. First, we consider the fully defined BFU of n variables defined on 2n input sets. The main metric of a BFU is the number of variables n. In principle, the boolean formula is divided into two groups, repeatless and repetitive. The classification of non-repetitive BF reduction is in Figure 4.

A formula will be called non-repetitive if each argument, taken in direct or inverse form, is included in it no more than once. In all other cases, the formula is repeated.

If in the notation of Boolean iteration formulas \( f(x_i) \), where \( i = 1,2, ..., n \), - n is the number of arguments, the index i is increased from left to right with the help of logical arguments, then we assume that the formula is sorted. We assume that the formula is ordered even with the same transformations, so we get a formula with incremental values and arguments that increase from left to right. If there are no arguments to any index in the record of a Boolean formula that does not repeat, we will assume that this formula contains the flaws of the corresponding arguments and calls it the formula with omitted arguments.

Thus, in accordance with the above classification (Figure 4), the set of Boolean formulas splits into pairwise disjoint classes — sets of the same type of formulas. Each formula of a particular class can be chosen as a representative of this class. Boolean formulas belonging to one class are realized by physically identical schemes, therefore, for each class, it suffices to implement only one scheme, the structure of which is described by the formula of a representative of a class.

We synthesize such an automaton, which will provide, at a certain setting, the calculation of all the BF groups provided in Figure 4, while the input arguments are not the same.

5. Tunable automata
A tunable automaton can be represented as a set of automata with the same outputs, and the setting determines the automaton whose outputs are considered at this setting as the outputs of the entire
automaton being tuned. Therefore, tunable automaton cannot implement anything except automaton mappings.

If the tunable automaton at any settings generates the value of the signals at its outputs, which depends only on the values of the signals at its inputs, it is called a combinational tunable automaton, or a multifunctional logic module.

We form automaton mappings (Figure 5), which the tunable automaton should implement, in order to provide the calculation of the Boolean formula specified in the class. Let us write systems of Boolean formulas corresponding to each automaton mapping:

1) \( z_1 = 0, \ z_2 = 0, \ z_3 = 0, \ z_4 = 0, \ z_5 = 0, \) \( (\text{Figure 5a}) \)
\[
f_r^1 = f_r^3 = 0, \ f_r^2 = i_1, \ f_r^4 = i_5;
\]

2) \( z_1 = 0, \ z_2 = 0, \ z_3 = 0, \ z_4 = 0, \ z_5 = 1, \) \( (\text{Figure 5b}) \)
\[
f_r^1 = i_2, \ f_r^2 = f_r^3 = f_r^4 = 0;
\]

3) \( z_1 = 0, \ z_2 = 0, \ z_3 = 0, \ z_4 = 1, \ z_5 = 0, \) \( (\text{Figure 5c}) \)
\[
f_r^1 = f_r^2 = f_r^4 = 0, \ f_r^3 = i_1 \cdot i_4;
\]

4) \( z_1 = 0, \ z_2 = 0, \ z_3 = 0, \ z_4 = 1, \ z_5 = 1, \) \( (\text{Figure 5d}) \)
\[
f_r^1 = f_r^2 = f_r^3 = f_r^4 = 0, \ f_r^3 = i_1 \cdot i_2;
\]

5) \( z_1 = 0, \ z_2 = 0, \ z_3 = 1, \ z_4 = 0, \ z_5 = 0, \) \( (\text{Figure 5e}) \)
\[
f_r^1 = f_r^2 = f_r^3 = f_r^4 = 0, \ f_r^3 = i_1 \cdot i_2;
\]

6) \( z_1 = 0, \ z_2 = 0, \ z_3 = 1, \ z_4 = 0, \ z_5 = 1, \) \( (\text{Figure 5f}) \)
\[
f_r^1 = f_r^2 = f_r^3 = f_r^4 = 0, \ f_r^3 = i_4;
\]

7) \( z_1 = 0, \ z_2 = 0, \ z_3 = 1, \ z_4 = 1, \ z_5 = 0, \) \( (\text{Figure 5g}) \)
\[
f_r^1 = f_r^2 = f_r^3 = f_r^4 = 0, \ f_r^4 = i_1;
\]

8) \( z_1 = 0, \ z_2 = 0, \ z_3 = 1, \ z_4 = 0, \ z_5 = 0, \) \( (\text{Figure 5h}) \)
\[
f_r^1 = f_r^3 = f_r^4 = 0, \ f_r^2 = i_1;
\]

9) \( z_1 = 0, \ z_2 = 1, \ z_3 = 0, \ z_4 = 0, \ z_5 = 0, \) \( (\text{Figure 5i}) \)
\[
f_r^1 = f_r^3 = f_r^4 = 0, \ f_r^2 = i_1, \ f_r^4 = i_2;
\]

10) \( z_1 = 0, \ z_2 = 1, \ z_3 = 0, \ z_4 = 0, \ z_5 = 1, \) \( (\text{Figure 5j}) \)
\[
f_r^1 = f_r^2 = f_r^3 = 0, \ f_r^3 = i_1 \cdot i_4;
\]

11) \( z_1 = 0, \ z_2 = 1, \ z_3 = 0, \ z_4 = 1, \ z_5 = 0, \) \( (\text{Figure 5k}) \)
\[
f_r^1 = f_r^2 = f_r^4 = 0, \ f_r^3 = i_1 \cdot i_2;
\]

12) \( z_1 = 0, \ z_2 = 1, \ z_3 = 0, \ z_4 = 1, \ z_5 = 1, \) \( (\text{Figure 5l}) \)
\[
f_r^1 = f_r^3 = f_r^4 = 0, \ f_r^2 = i_1 \cdot i_4, \ f_r^4 = i_2;
\]

13) \( z_1 = 0, \ z_2 = 1, \ z_3 = 1, \ z_4 = 0, \ z_5 = 0, \) \( (\text{Figure 5m}) \)
\[
f_r^1 = f_r^2 = f_r^4 = 0, \ f_r^3 = i_1;
\]

14) \( z_1 = 0, \ z_2 = 1, \ z_3 = 1, \ z_4 = 0, \ z_5 = 1, \) \( (\text{Figure 5n}) \)
\[
f_r^1 = f_r^3 = 0, \ f_r^2 = i_4, \ f_r^4 = i_2;
\]

15) \( z_1 = 0, \ z_2 = 1, \ z_3 = 1, \ z_4 = 1, \ z_5 = 0, \) \( (\text{Figure 5o}) \)
Based on the structured automation approach \cite{4, 5}, we get the following boolean system that will describe the entire basic PVS:

\[ f_r^1 = f_r^2 = f_r^3 = 0, \quad f_r^4 = i_2; \]

16) \[ z_1 = 0, \quad z_2 = 1, \quad z_3 = 1, \quad z_4 = 1, \quad z_5 = 1, \text{ (Figure 5p)} \]

\[ f_r^1 = f_r^3 = f_r^4 = 0, \quad f_r^2 = i_4; \]

17) \[ z_1 = 1, \quad z_2 = 0, \quad z_3 = 0, \quad z_4 = 0, \quad z_5 = 0, \text{ (Figure 5q)} \]

\[ f_r^1 = f_r^3 = f_r^4 = 0, \quad f_r^2 = i_1 \lor i_4; \]

18) \[ z_1 = 1, \quad z_2 = 0, \quad z_3 = 0, \quad z_4 = 0, \quad z_5 = 1, \text{ (Figure 5r)} \]

\[ f_r^1 = f_r^3 = f_r^4 = 0, \quad f_r^2 = i_1 \cdot i_4; \]

19) \[ z_1 = 1, \quad z_2 = 0, \quad z_3 = 0, \quad z_4 = 1, \quad z_5 = 0, \text{ (Figure 5s)} \]

\[ f_r^1 = f_r^3 = f_r^4 = 0, \quad f_r^2 = i_2 \cdot i_3; \]

20) \[ z_1 = 1, \quad z_2 = 0, \quad z_3 = 0, \quad z_4 = 1, \quad z_5 = 1, \text{ (Figure 5t)} \]

\[ f_r^1 = f_r^4 = 0, \quad f_r^2 = i_4, \quad f_r^3 = i_1; \]

21) \[ z_1 = 1, \quad z_2 = 0, \quad z_3 = 1, \quad z_4 = 0, \quad z_5 = 0, \text{ (Figure 5u)} \]

\[ f_r^1 = f_r^2 = f_r^4 = 0, \quad f_r^3 = i_1 \cdot i_2 \cdot i_4; \]

22) \[ z_1 = 1, \quad z_2 = 0, \quad z_3 = 1, \quad z_4 = 1, \quad z_5 = 0, \text{ (Figure 5v)} \]

\[ f_r^1 = f_r^2 = 0, \quad f_r^3 = i_1, \quad f_r^4 = i_2; \]

23) \[ z_1 = 1, \quad z_2 = 0, \quad z_3 = 1, \quad z_4 = 1, \quad z_5 = 0, \text{ (Figure 5w)} \]

\[ f_r^1 = f_r^2 = f_r^3 = 0, \quad f_r^4 = i_1 \lor i_4. \]
Let us examine the PVS of the matrix type with the calculation nodes of the two-dimensional action, as shown in Figure 5. We found, according to the classification (Figure 4) of a repeated, ordered, without missing the arguments, presented in the DNF formulas:

\[
f_r^4 = i_2 \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}),
\]

\[
f_r^2 = i_4 \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee i_2 \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee (i_1 \cdot i_4) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee
\]

\[
\vee (i_1 \cdot i_4) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee i_4 \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}).
\]

\[
f_r^3 = (i_1 \cdot i_4) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee (i_1 \cdot i_2) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee (i_1 \cdot i_4) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee
\]

\[
\vee (i_1 \cdot i_4) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee i_4 \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}.
\]

\[
f_r^4 = i_3 \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee (i_1 \cdot i_2) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee (i_1 \cdot i_2) \cdot
\]

\[
(\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee i_1 \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee
\]

\[
(i_1 \cdot i_2) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5} \vee (i_1 \cdot i_2) \cdot (\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}) \vee (i_1 \cdot i_2) \cdot
\]

\[
(\overline{z_1 \cdot z_2 \cdot z_3 \cdot z_4 \cdot z_5}).
\]

6. Modeling

Let us examine the PVS of the matrix type with the calculation nodes of the two-dimensional action, as shown in Figure 5. We found, according to the classification (Figure 4) of a repeated, ordered, without missing the arguments, presented in the DNF formulas:

\[
f_i = x_1 \cdot x_2 \vee x_3 \cdot x_4 \cdot x_5.
\]  

(1)

A homogeneous computing environment that implements formula (1) is shown in Figure 6.

\[f_i = x_1 \cdot x_2 \vee x_3 \cdot x_4 \cdot x_5.\]
Let us build a tunable computing environment, to refine the octomap algorithm. In this case, you can use the matrix structure of the calculator (Figure 7). The elementary calculator for one of the subtasks can be described by the following formula:

\[
\begin{align*}
    f_1 &= x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5, \\
    f_2 &= x_1, \\
    f_3 &= x_1, \\
    f_4 &= x_1,
\end{align*}
\]

where as \(x_1 - x_4\), data from a cloud of points is fed in, and \(x_5\) is information about performed calculations in selected subgroups.

7. Conclusion
In this article, we used a point cloud to create an octree tree. The results show that when using uniformly structured models and parallel processing, the results and implementation time of the algorithm are faster at each stage of performing the tasks of image processing in three dimensions. This environment allows you to build a map based on octree very quick.

Acknowledgments
The research was carried out in Tomsk State University with the financial support of the Ministry of Education and Science of the Russian Federation, a unique project identifier: RFMEFI57817X0241. The authors are grateful to Tatiana B. Rumyantseva from Tomsk State University for English language editing.

References
[1] Lavrenov R., Zakiev A., Magid E. 2017 International Conference on Mechanical, System and Control Engineering, ICMSC 1-2
[2] Hornung A., Wurm K. M., Bennewitz M., Stachniss C., Burgard W. 2013 Autonomous Robots 189–206 doi: 10.1007/s10462-013-9297-6
[3] Bergström J. 2018 Path Planning with Weighted Wall Regions using OctoMap (Publishing House Luleå University of Technology: Lulea)
[4] Shidlovsky S. V. 2006 Automatic control. Tunable structures (TSU Publishing House: Tomsk)
[5] Kuznetsov D., Syryamkin V. 2015 AIP Conference Proceedings 1688 040004 doi: 10.1063/1.4936037