Jump Over Block (JOB): An Efficient Line-of-Sight Checker for Grid/Voxel Maps With Sparse Obstacles

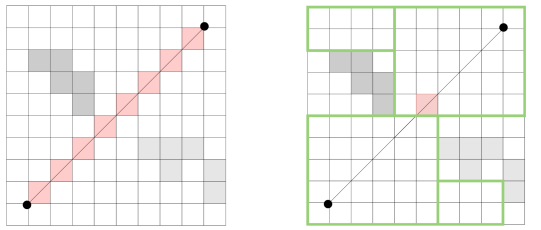
Zhuo Yao, Wei Wang, Jiadong Zhang, Yan Wang, and Jinjiang Li

***Abstract*—Line-Of-Sight (LOS) check plays a crucial role in collision avoidance and time comsuming, particularly in scenarios involving large-scale maps with sparse obstacles, as it necessitates a grid-by-grid state check. Specifically, LOS check consumes more than half of the computational time in any-angle path planning algorithms, such as Theta*, Visibility Graph, and RRT. To address this issue, we propose an efficient LOS checker for maps of arbitrary dimensions with sparse obstacles. Our approach involves a two-step process. Firstly, we partition the passable space into blocks until there is no vacancy for a minimum-sized block. When the adapted Bresenham algorithm reaches a surface of a block, it bypasses grid-by-grid traversal within the block and directly jumps to the opposing surface. This method significantly reduces the number of grids examined, resulting in higher efficiency compared to traditional LOS checks. We refer to our approach as Jump Over Block (JOB). To demonstrate the advantages of JOB, we compare its performance against traditional LOS checks using a widely recognized public dataset[1]. The results indicate that JOB incurs only 1/6 to 1/5 of the computational cost associated with raw LOS checks, making it a valuable tool for both researchers and practitioners in the field. In order to facilitate further research within the community, we have made the source code of the proposed algorithm publicly available[2]. We anticipate that this framework will contribute to the development of more efficient path planning algorithms and expedite various aspects of robotics that involve collision checks.**

***Index Terms*—Collision avoidance, computational geometry, motion and path planning.**

I. Introduction

IN COLLISION avoidance for path planning [1], [2] and motion planning [3], [4], the line-of-sight (LOS) check plays a pivotal role. The Bresenham algorithm is commonly employed in LOS checks for various path planning algorithms, including Theta*, Visibility[1] Graph, and RRT. However, its computational demands can be significant, particularly when[2] dealing with large-scale or sparse maps. In practice, more than half of the time in path planning is devoted to LOS checks. Consequently, accelerating the LOS check process becomes crucial for achieving more efficient path planning and motion planning, enabling faster navigation system responses and reducing computational resource requirements.



(a): how raw LOS check work (b): how JOB's LOS check work

Fig. 1. Two figures depict a illustrative example of how blocks can accelerate line-of-sight (LOS) checks within a grid map. The left figure demonstrates the process of the raw LOS check, while the right figure illustrates the implementation of the JOB LOS check method. Obstacle grids are visually indicated in gray, grids subjected to LOS evaluation are highlighted in pink, passable grid blocks are represented in green, and the initiation and termination points of the LOS assessment are denoted by black dots. In the context of the raw LOS check, each grid that the line traverses requires check. Conversely, by disregarding internal grids within the blocks, the JOB method only necessitates the examination of grids positioned along the boundary of each block.

In this manuscript, we introduce an innovative technique aimed at expediting the LOS check process for maps of any dimension. This is achieved through a reduction in the number of grids that need to be examined. Our method involves the partitioning of the entire passable space into distinct blocks, each of which is then expanded to its maximum extent until encountering obstacles or other adjacent blocks. This strategy yields remarkable efficiency in scenarios where the map contains sparse obstacles, as illustrated in Fig. 1. Importantly, it is essential to emphasize that the proposed JOB technique yields outcomes that are equivalent to those produced by the conventional raw LOS check, ensuring the preservation of accuracy without any compromise.

The selection of blocks, as opposed to alternative shapes, is driven by their storage properties and their ability to impose less demands on computational resources when determining intersections with lines. By adopting the block-based approach, we ensure storage efficiency while simultaneously reducing the computational overhead required for line intersection computations.

The subsequent sections of this article are structured as follows. Section II provides an introduction to relevant studies concerning the reducing the time cost of LOS check within

Manuscript received 4 September 2023; accepted 19 September 2023. Date of publication 28 September 2023; date of current version 10 October 2023. This letter was recommended for publication by Associate Editor P. Gao and Editor A. Bera upon evaluation of the reviewers' comments. This work was supported by the National Key Research and Development Program of China under Grant 2020YFB1313600. *(Corresponding author: Wei Wang.)*

The authors are with the School of Mechanical Engineering and Automation, Beijing University of Aeronautics and Astronautics, Beijing 100191, China (e-mail: yaozhuo@buaa.edu.cn; wangweilab@buaa.edu.cn; zhangjiadong@buaa.edu.cn; huohuoqwe000@163.com; ljjbuaa@buaa.edu.cn).

Digital Object Identifier 10.1109/LRA.2023.3320435

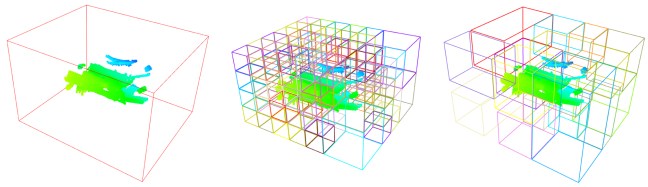
[1][Online]. Available: https://www.movingai.com/benchmarks/voxels.html

[2][Online]. Available: https://joeyao-bit.github.io/posts/2023/05/26/

the context of path planning. Section III describes the major processes involved in our method. Section IV presents a comparison between our method and the raw Bresenham algorithm, as well as provides details on block detection and the differences between the blocks detected by our method (JOB) and Octomap. Finally, in Section V, we discuss the results obtained and potential drawbacks of the proposed method.

This letter contributes the following:

1) An efficient LOS checker for grid spaces of any dimension with sparse bstacles.
2) An algorithm for partitioning arbitrary-dimension grid maps, dividing traversable space into maximum-sized blocks.



(a): raw voxel map (b): blocks of octomap (c): blocks of JOB

Fig. 2. Figures provided illustrate a voxel map with dimensions $246 \times 154 \times 205$ and depict the blocks detected by the JOB algorithm and Octomap, under the same minimum block width constraint ($\tau = 20$). In Octomap's result, a total of 562 blocks were identified, whereas our block detection algorithm detected 16 blocks. The map used is the Complex.3dmap from a publicly available voxel map dataset [11].

II. Related Works

A strategy to speed up Line of Sight (LOS) checks is to reduce the total number of LOS checks and only perform LOS checks when necessary. Hauser et al. [5] proposed two asymptotically-optimal planners, Lazy-PRM* and Lazy-RRG*, that eliminate the majority of collision checks using a lazy strategy. This implies that edges are not immediately checked for collision; rather, they are checked only when a better path to the goal is found. This approach avoids checking the vast majority of edges that have no chance of being on an optimal path. Zhang et al. [6] and [7] introduced Late Line of Sight-Check A* (LLA*) to reduce the frequency of LOS checks. This involves performing an LOS check of a vertex v and its parent when v is about to be expanded.

Białkowski et al. [8] proposed an alternative approach to decrease the frequency of collision checking within the framework of sampling-based motion planning. During LOS check, it is determined whether the distances from both ends of a line to the nearest vertex v (a randomly sampled point) are smaller than v's distance to its nearest obstacle. If this condition is met, the line is considered collision-free; otherwise, a standard line collision check is performed.

The outcomes presented by the aforementioned algorithms demonstrate their enhanced effectiveness compared to the original path planning methods, achieved by reducing the frequency of LOS checks. However, it is important to note that these approaches do not alter the fundamental nature of LOS checks. Consequently, their applicability remains confined to graph-based path planning scenarios and lacks the generality required to address arbitrary scenes necessitating LOS checks. Compared to these methods, JOB reduces the time required for each LOS check, instead of decreasing the frequency of LOS checks. Therefore, it can be applied generically to any area requiring LOS verification.

Another strategy to expedite LOS checks is to limit the range of examination to the local map instead of checking between two waypoints. Anya [9] introduced the concept of interval search, which requires no preprocessing and expands from the current vertex to identify all visible vertices, stopping upon encountering obstacles. This method was designed for 2D grid maps. Another similar method, called "Line-of-sight scan," was proposed by ENL-SVG [10], incorporating a precomputation technique. Line-of-sight scans exhibit runtime dependence on the number of visible vertices for each vertex, while Line-of-Sight Checks depend on the total number of vertices in the entire map. Line-of-sight scans are considerably faster, especially in larger maps with crowded obstacles.

However, for large-scale maps with sparse obstacles, such as voxel maps in Brewer et al. [11], local scan methods often necessitate scanning nearly the entire space. Furthermore, existing works primarily focus on 2D grid maps, and to the best of our knowledge, there is currently no LOS scan method available for 3D maps or higher-dimensional maps. In contrast, JOB is specifically designed to handle maps with sparse obstacles of arbitrary dimensionality. Furthermore, the sparser the map, the more efficient JOB becomes in terms of LOS check costs.

While methods like Octomap [12] and quadtree [13] also divide the passable space into blocks (although this was not their primary purpose), they generate blocks with sizes fixed to powers of 2, so octomap tends to generate more block than JOB, as illustrated in Fig. 2. Consequently, these methods necessitate traversing more blocks during the LOS check process. In contrast, by generating blocks in as large a manner as possible, fewer blocks need to be traversed while still maintaining coverage, leading to more efficient LOS checks compared to Octomap and quadtree. Meanwhile, since jumping over a block requires more calculations compared to checking a grid's state, having too many small blocks can slow down the Line of Sight (LOS) check process. In Section III, we will explain how to determine the optimal minimum block width to achieve optimal performance.

III. Methodology

A. Definitions

In this section, we present fundamental definitions pertaining to JOB. Considering that our method is dimension-independent, the definitions provided apply to any-dimensional grid space $\mathcal{C}_{\mathcal{N}}$.

1) Grid Space: Let $\mathcal{C}_{\mathcal{N}}$ $(\mathcal{N} \geq 2)$ denote a finite $\mathcal{N}$-dimensional integer Euclidean space, where the size of the space is defined as $\mathcal{D} \rightarrow \mathcal{C}_{\mathcal{N}}$, with $\mathcal{D} = d_1, d_2, \ldots, d_i, \ldots, d_{\mathcal{N}}$ and $d_i \in \mathbb{N}$. The coordinates of an element g in this space are defined as a vector $(x_1, x_2, \ldots, x_i, \ldots, x_{\mathcal{N}})$, where $x_i \in ([0, d_i) \cap \mathbb{N})$. Typically, for $\mathcal{C}_2$ (grid maps) or $\mathcal{C}_3$ (voxel maps), the coordinates are presented as x, y or (x, y, z), respectively.

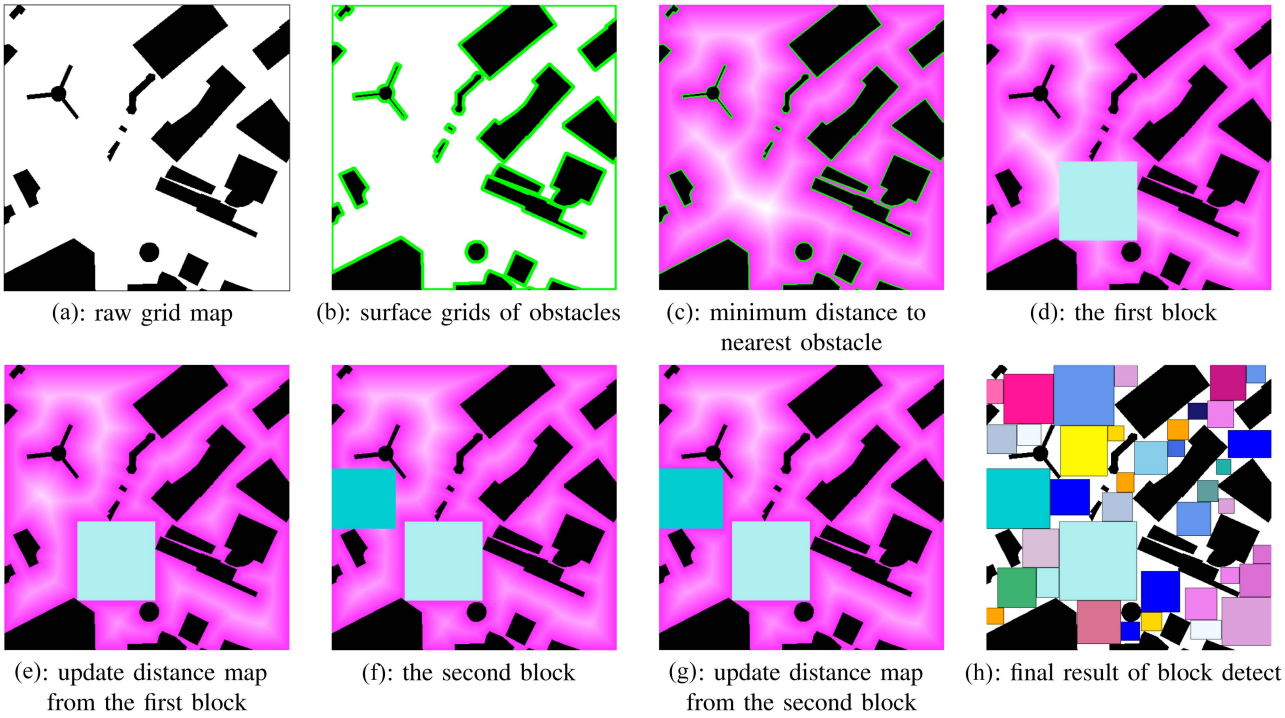


Fig. 3. Figures in this section illustrate key outcomes obtained during the block detection process. (b) Offers a visualization of the grids in proximity to obstacles, represented by the color green. In (c), varying shades of purple depict the distances to the nearest obstacles. (d) Outlines the process of generating the initial block, originating from the location with the maximum distance to an obstacle, while (e) demonstrates the subsequent alterations to $\mathcal{D}$ following the update of the distance map from the initial block. By iteratively following the steps of block generation and $\mathcal{D}$ update until no blocks possess a width $\geq \tau$, the final set of blocks is derived, as depicted in (h).

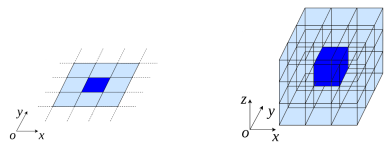


Fig. 4. Toy examples of neighborhood in $\mathcal{C}_2$ and $\mathcal{C}_3$ are shown in this figure, where g is shown in deep blue, $\delta(g)$ are shown in light blue.

2) Surface: The surface of a grid space $\mathcal{C}_\mathcal{N}$ is an $\mathcal{N}$-dimensional grid space. A block in $\mathcal{C}_\mathcal{N}$ has $2\mathcal{N}$ surfaces.

3) Grid State: There are only two possible states for a grid/element in $\mathcal{C}_\mathcal{N}$: passable or unpassable. The set of all passable grids in $\mathcal{C}_\mathcal{N}$ is denoted as $\mathcal{F} \to \mathcal{C}_\mathcal{N}$, while the set of all unpassable grids is denoted as $\mathcal{O} \to \mathcal{C}_\mathcal{N}$. Therefore, $(\mathcal{F} \to \mathcal{C}_\mathcal{N}) \cup (\mathcal{O} \to \mathcal{C}_\mathcal{N}) = \mathcal{C}_\mathcal{N}$.

For convenience, we denote all grids outside of $\mathcal{C}_\mathcal{N}$ as unpassable.

4) Neighborhood of Grid: Denote $\delta(g)$ as the neighborhood of grid g, which consists of the closest $3^\mathcal{N} - 1$ grids near g. Toy examples of neighborhoods in $\mathcal{C}_2$ and $\mathcal{C}_3$ are shown in Fig. 4.

5) Line Collision Condition: Denote the line connecting two grids g_1 and g_2 as $g_1 \to g_2$. We define the set of all grids that $g_1 \to g_2$ crosses as $\omega(g_1 \to g_2)$. The line $g_1 \to g_2$ is said to be collided if there exists $g \in \omega(g_1 \to g_2)$ such that $g \in \mathcal{O} \to \mathcal{C}_\mathcal{N}$. If $g_1 \to g_2$ collides, it is denoted as $(g_1 \to g_2) \in \mathcal{O}$; otherwise, it is denoted as $(g_1 \to g_2) \in \mathcal{F} \to \mathcal{C}_\mathcal{N}$.

6) Distance Metrics: The distance between two grids g_1 and g_2 is denoted as $\Omega(g_1 \to g_2)$, which is defined as the Euclidean distance between the two grids.

7) Distance Map: A distance map $\mathcal{D}$ is defined with the same size as $\mathcal{C}_\mathcal{N}$, storing the distances to the nearest obstacles or boundaries. $\mathcal{D}(g)$ represents the distance from grid g to the nearest obstacle, and $\mathcal{D}(g).nearest$ represents the grid of the nearest obstacle to g.

8) Block: A block is a fully passable box-shaped space b defined by two grids: a grid $g_{\min}$ with minimum coordinates and a grid $g_{\max}$ with maximum coordinates. There is no overlap between any two grids. Each surface of a block is adjacent to another block or an obstacle grid. An example of blocks in $\mathcal{C}_2$ is shown in Fig. 2. The set of all blocks is denoted as $\mathcal{B}$, and the set of all blocks in $\mathcal{C}_\mathcal{N}$ with a minimum required width τ is denoted as $\mathcal{B} \to (\mathcal{C}_\mathcal{N}, \tau)$.

B. Block Detection

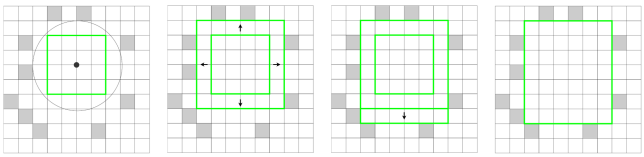
In this section, we present how to split passable space of $\mathcal{C}_\mathcal{N}$ into blocks, in short, block detection. The process of block detection involves a single parameter, the minimum required block width τ. The determination of the optimal τ for $\mathcal{C}_\mathcal{N}$ will be discussed in the next section. The block detection process consists of the following steps:

1) Detection of all surface grids, which are passable grids near obstacles.

Algorithm 1: Block Detect.

Input: $\mathcal{C}_\mathcal{N}, \tau$ // grid space $\mathcal{C}_\mathcal{N}$ and minimum block width τ
Output: $\mathcal{B} \rightarrow (\mathcal{C}_\mathcal{N}, \tau)$ // All blocks in $\mathcal{C}_\mathcal{N}$

$\mathcal{D} \leftarrow$Wave Front from surface grids of $\mathcal{C}_\mathcal{N}$;
$\mathcal{B} = \emptyset$;
while $True$ **do**
  $d_{max} \leftarrow$ maximal distance in $\mathcal{D}$;
  if $2\frac{d_{max}}{\sqrt{\mathcal{N}}} >= \tau$ **then**
    b = initilize block from d_{max};
    expand b till no room for expansion;
    set b as obstacle and update $\mathcal{D}$;
    $\mathcal{B} \leftarrow b$;
  else
    break;
return $\mathcal{B}$;



(a) initial block (b) the 2nd step (c) the 3rd step (d) final block

Fig. 5. These figures demonstrate how a block is initialized and expanded under $\mathcal{C}_2$. Green rectangles represent the block at each step, and the center of the circle represents the local maximum, with the circle's radius denoted as $d_{\max}$. The first step involves initializing a block that has the largest inscribed block with the same side length as the circle (a). In the second step, all boundaries of the block are expanded in $2\mathcal{N}$ directions (b). In the third step, only one direction is available for expansion (c). When there is no more space to expand, the final block configuration is obtained (d). In practice, a block may undergo multiple expansion steps until no further expansion is possible.

2) Generation of the distance map to the nearest obstacles. This is achieved by expanding the minimum distance to the obstacles from all surface grids until no more grids need to be updated.
3) Detect the grid that has the maximum distance (noted as $d_{\max}$) to obstacles. If $d_{\max}$ is larger than $\frac{\tau\sqrt{\mathcal{N}}}{2}$, initialize a block based on its distance to the nearest obstacles, as illustrated in Fig. 5(a). Otherwise, terminate the process. The initial block is designed with all sides having the same length of $2\frac{d_{\max}}{\sqrt{\mathcal{N}}}$, where the value is rounded down to ensure safety. This length corresponds to the maximum side length of a block that maintains equal side lengths inside a sphere with a radius of $d_{\max}$.
4) Expand the current blocks until all of their $2\mathcal{N}$ surfaces make contact with obstacles or other blocks. The block expansion follows a reverse onion-peeling process, as depicted in Fig. 5.
5) Update the distance map using a wavefront algorithm after designating all surfaces of the preceding block as obstacles, subsequently proceeding to step 3.

The block process is depicted in Algorithm 1. An example of detecting blocks within $\mathcal{C}_2$ is demonstrated in Fig. 3. While this process remains consistent in higher-dimensional spaces, its visual representation might not be as straightforward as in the case of $\mathcal{C}_2$. After the block detection concludes, the outcomes are stored in binary files for convenient.

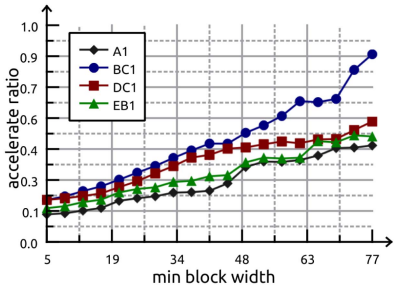


Fig. 6. This figure illustrates the variation in acceleration ratio as τ increases across different maps characterized by sparse obstacles. A comprehensive set of 10,000 random samples of start and target of LOS check was generated, and each individual sample was repeated 100 times to ensure robust statistical significance. Specifically, the voxel maps denoted as A1, BC1, DC1, and EB1 have been extracted from a publicly available voxel map dataset, as referenced. The purpose of this representation is to provide insight into how the acceleration ratio evolves in response to incremental changes in τ, within the context of distinct map configurations featuring sparse obstacle distributions.

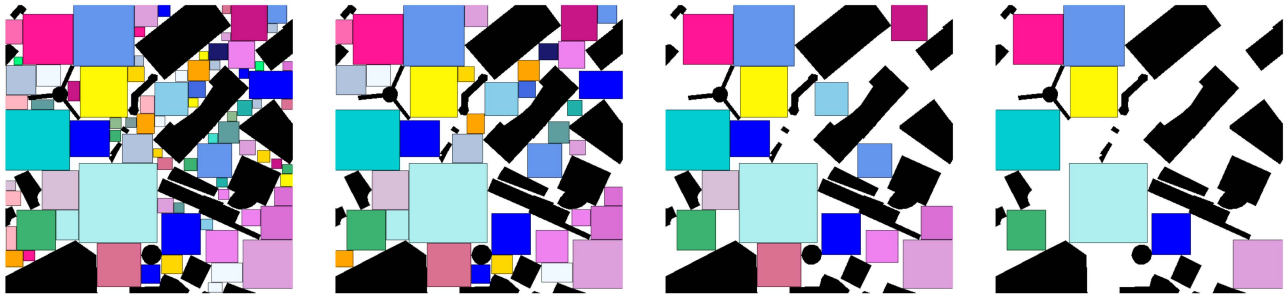
Upon implementing block detection, we conducted a performance analysis using Perf to evaluate the time expenditure associated with this process. Notably, we observed that a significant proportion (approximately 70%) of the block detection time was attributed to the utilization of the wavefront algorithm for updating $\mathcal{D}$. Given that free grid cells may be traversed multiple times during the wavefront computation, optimizing the total count of free grids could yield a substantial acceleration in the wavefront algorithm.

To achieve this, a strategy was employed wherein the raw grid space was initially zoomed out to $1/\tau$ of its original dimensions. During this stage, shrunken blocks were detected, and subsequently, the identified blocks were zoomed back in to restore them to their original scale within the raw grid space. Experimental results indicated that this approach led to a remarkable reduction of approximately one-tenth in the time required for block detection, all while maintaining the quality of the identified blocks.

It is important to note that the time expenditure associated with block detection diminishes with an increase in τ, as larger values of τ entail the detection of blocks within a smaller grid space. However, an elevated value of τ also introduces a filter that eliminates smaller blocks. This filtering effect may potentially reduce the overall efficiency of the process, thereby necessitating a judicious equilibrium between expediting block detection and optimizing line-of-sight checks.

C. Accelerate LOS Check With Blocks

In this section, we elucidate the intricacies of how blocks contribute to the acceleration of line-of-sight (LOS) checks. The difference between raw line-of-sight (LOS) calculations and LOS checks with block acceleration is the addition of a jump over blocks for acceleration and adaptation to arbitrary



(a): τ = 20 grids, 82 blocks (b): τ = 40 grids, 39 blocks (c): τ = 60 grids, 16 blocks (d): τ = 80 grids, 8 blocks

Fig. 7. Block detection process was applied to a grid map with various block widths (τ). The results clearly show that using a small τ leads to the generation of numerous tiny blocks, whereas a large τ produces only a few large blocks. The map used for this analysis is the Shanghai_0_512.map from the mentioned public grid map dataset, and its size is 512 $\times$ 512.

Algorithm 2: Block Accelerated Bresenham.

Input: $g_1, g_2, \mathcal{C}_\mathcal{N}, \mathcal{B} \rightarrow (\mathcal{C}_\mathcal{N}, \tau)$ // two ends of line, grid space $\mathcal{C}_\mathcal{N}$ and related blocks $\mathcal{B}$

Output: Whether current line collide with obstacles

step = 0;
while $step < \Omega(g_1 \rightarrow g_2)$ **do**
  if $g \in \mathcal{O} \rightarrow \mathcal{C}_\mathcal{N}$ **then**
    return True; // $g_1 \rightarrow g_2$ is collide with obstacle
  if $g \in b \in \mathcal{B} \rightarrow (\mathcal{C}_\mathcal{N}, \tau)$ **then**
    step = step + length of line $g_1 \rightarrow g_2$ intersect with b;
  else
    step = step + 1;
return False; // $g_1 \rightarrow g_2$ not collide with obstacle

dimensional space. During the traversal of all grids in a line from g_1 to g_2 with a given block width τ, if the current grid is on the surface of a block b, the algorithm jumps over the current block to another surface of b in the direction of the line. Hence, the LOS checks performed by JOB exhibit no compromise in terms of accuracy. This process allows for faster LOS calculations by skipping the interior grids of the block. The algorithm for this jump over block acceleration is illustrated in Algorithm 2.

D. Choose Minimal Block Width

In this section, we delve into the deliberation of selecting an optimal minimal block width, informed by the performance characteristics of JOB. To determine the optimal minimal block width, we evaluate the performance of LOS checks with block acceleration. We measure the performance by calculating the ratio between the time cost of LOS checks with blocks and the time cost of raw LOS checks using the Bresenham algorithm. The acceleratetion ratio remains unaffected by the computing device and is solely contingent upon the density of obstacles and the chosen value of τ. A smaller ratio indicates a more efficient acceleration. This ratio serves as an indicator that is independent of the specific map. The time cost represents the duration of LOS checks between two uniformly randomly sampled free grids.

In our experiments, we observed that using a small block width results in many tiny blocks, while a large block width produces a few large blocks, as shown in Fig. 7. A concern that arises pertains to the potential increase in time cost due to an abundance of small blocks. To address this concern, an experiment was conducted to observe how the ratio evolves as τ is incrementally adjusted across various voxel maps. The resultant findings are depicted in Fig. 6. Evidently, an upward trend is observed in the ratio as τ increases. This trend can be attributed to the proximity of small blocks to obstacles, wherein the incorporation of a higher number of small blocks contributes positively to the performance of JOB.

It is reasonable to consider setting τ proximate to a value of 1. However, it is advisable for τ to be greater than 1. A block possessing a width of 1 implies its containment of just one grid cell, rendering it devoid of meaningful traversal optimization. Striking a balance is imperative, as a reduction in τ escalates the time cost associated with block detection, while maintaining an equilibrium is essential to cater to both block detection (precomputation) and line-of-sight checks (online computation).

In our comparative analysis between octomap's blocks (OCM) and the blocks detected by our method (JOB), we discovered that our method exhibited superior performance in terms of acceleration ratio and the number of blocks generated. Octomap, due to it splits passable space into blocks with fixed width (power of 2), tends to generate a larger number of smaller blocks. In contrast, our method demonstrated the advantage of having a smaller number of blocks while achieving efficient LOS checks.

For more detailed information and comprehensive results, please refer to Section IV of this study.

IV. Results

In this section, we assess the performance of the line-of-sight (LOS) check by measuring its time cost to determine whether the connection between two randomly sampled free voxels collides with obstacles. Our experiments are conducted using a well-known voxel map dataset [11],[3] as illustrated in Fig. 8. The

[3][Online]. Available: https://www.movingai.com/benchmarks/voxels.html

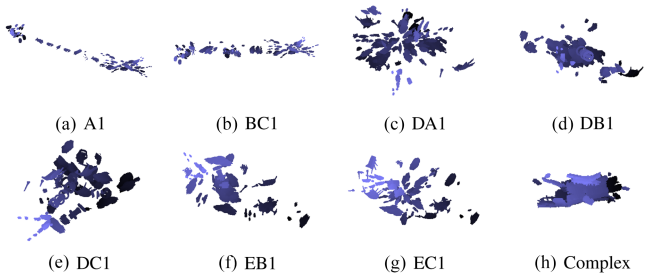


Fig. 8. Some 3D voxel maps used in Section IV.

TABLE I
DETAILS OF MAP USED IN COMPARISON

Index	Name	Scale(voxel)	Total States
0	Simple	$105 \times 132 \times 105$	1,454,787
1	Complex	$246 \times 154 \times 205$	7,719,921
2	A1	$896 \times 390 \times 255$	88,983,963
3	A2	$896 \times 390 \times 255$	88,984,381
4	A3	$896 \times 390 \times 255$	88,974,265
5	A4	$896 \times 390 \times 255$	88,977,600
6	A5	$896 \times 390 \times 255$	88,964,548
7	BC1	$853 \times 245 \times 321$	66,861,521
8	BC2	$853 \times 245 \times 342$	71,204,502
9	DA1	$389 \times 313 \times 511$	62,077,325
10	DA2	$377 \times 314 \times 535$	63,163,816
11	DB1	$409 \times 261 \times 462$	49,151,914
12	DB2	$418 \times 260 \times 466$	50,489,933
13	DC1	$396 \times 344 \times 533$	72,431,588
14	DC2	$375 \times 354 \times 554$	73,364,794
15	EB1	$445 \times 306 \times 617$	83,852,941
16	EB2	$465 \times 321 \times 617$	91,947,421
17	EC1	$500 \times 312 \times 562$	87,503,008
18	EC2	$468 \times 314 \times 565$	82,901,309

details of the voxel maps used in our experiments are provided in Table I.

For each map, we randomly sample 10,000 start and target combinations, and each LOS check is repeated 100 times for the sake of reliable comparison. The experiments were performed on a laptop running Ubuntu 20.04, equipped with a Ryzen 7 5800 h (3.2 GHz) CPU and 16 GB of memory.

A. Block Detection Comparison

In this section, we present a demonstration of the performance of our block detection method (JOB) compared to octomap's block detection (OCM). The pursuit of diverse methodologies for the sake of comparison is importance. However, to our best knowledge, it appears that octomap stands as the only method capable of segmenting traversable space into distinct blocks. We evaluate the performance based on the following metrics: time cost, number of blocks detected. In the context of JOB, a judicious equilibrium between block detection and LOS checks has been achieved by setting the minimum block width to 5. This choice strikes a balance, ensuring efficient precomputation and streamlined LOS checks. In order to facilitate a fair and accurate comparison, the blocks generated by octomap adhere to the same minimum block width constraint as employed in the case of JOB.

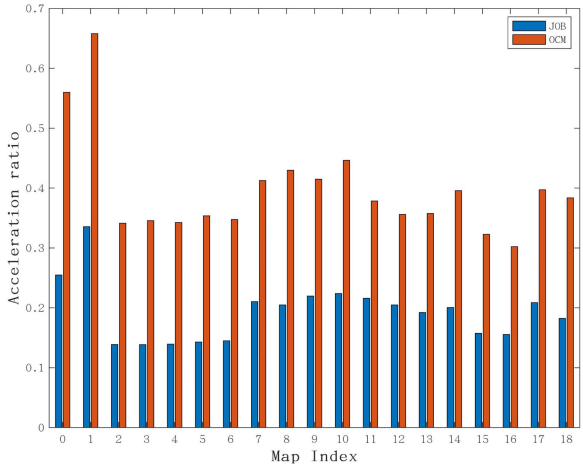


Fig. 9. These figures illustrate the influence of block detection of JOB and OCM on the time cost of LOS checks within voxel maps. The time cost of LOS signifies the duration taken for LOS checks with block acceleration, utilizing randomly sampled start and target positions. Each sample encompasses 10,000 randomized combinations, and each combination is replicated 100 times to ensure robust statistical precision.

In terms of number of blocks, on average, JOB generates 197.4 blocks while octomap generate 2538.6 blocks, our block detection method (JOB) has approximately only 1/10 of the number of blocks compared to octomap. This resulted that JOB requires a significantly smaller file size to save the detected blocks, JOB need less than 5 KB to save blocks while OCM need 50 KB in average. Additionally, in terms of time cost, on average, JOB takes 9.15 s while octomap takes 49.1 s, JOB outperforms octomap, taking approximately 1/5 of the time compared to octomap. This improved efficiency can be attributed to JOB's utilization of downsampling the grid space prior to block detection. In summary, JOB outperforms OCM in terms of block detection by offering both efficiency and flexibility.

B. Acceleration Ratio Comparison

In this section, we compare the time cost of LOS check accelerated by our block detection method (JOB) and octomap. We evaluate the performance by calculating the ratio between the time cost of LOS accelerated with blocks and the time cost of raw LOS check using the Bresenham algorithm. The results are shown in Fig. 9. To further illustrate the difference between our method's blocks (JOB) and octomap's blocks (OCM), we also visualize the visited grids/blocks during the LOS check in Fig. 10.

As depicted in Fig. 9, the ratio of JOB's time cost to raw LOS check is only half the ratio of OCM. This can be attributed to the fact that JOB visits fewer blocks and blocks, as shown in Fig. 10. And this is owing to JOB having a flexible block width which aims to maximize the size of each block, while OCM's block width is fixed to powers of 2. OCM's blocks are incapable of covering as many free grids as possible. In comparison to raw LOS checks, JOB takes only approximately one-fifth of the time cost, which proves that JOB is quite effective under maps with sparse obstacles.

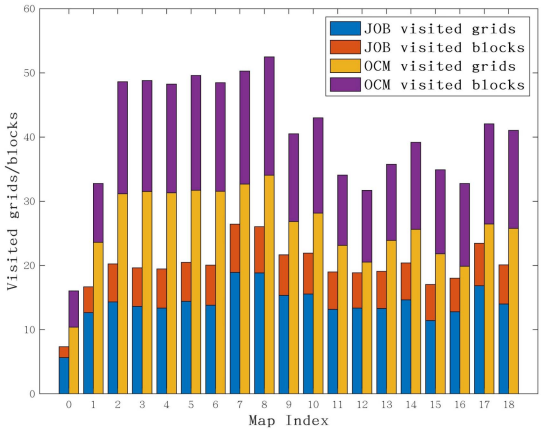
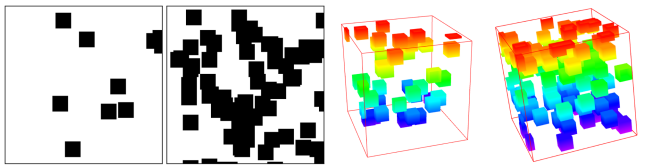


Fig. 10. This figure illustrates the number of grids and blocks visited by JOB and OCM during LOS check. JOB visits fewer grids and blocks compared to OCM, resulting in JOB's time cost being approximately only half that of OCM, as shown in Fig. 9.



(a) 2D, $\Phi = 31.1\%$ (b) 2D, $\Phi = 86.5\%$ (c) 3D, $\Phi = 31.6\%$ (d) 3D, $\Phi = 56.8\%$

Fig. 11. These figures showcase a selection of 2D and 3D random maps along with their corresponding Φ values. Notably, the figures reveal that as the obstacle density escalates, the corresponding Φ values also increase.

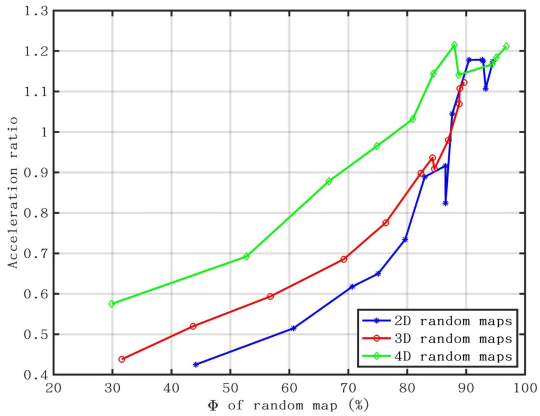


Fig. 12. This figure portrays the variation in the acceleration ratio of JOB as the density of obstacles (Φ) increases within the context of 2D, 3D, and 4D maps.

(a) Turtlebot 2 (b) A indoor map

Fig. 13. On the left is the platform we used to construct grid maps and test path planning algorithms, while on the right is an indoor map utilized in our experiments, measuring 47.8 meters by 64.6 meters in size.

C. Obstacle Density and Acceleration Ratio

JOB is specifically designed to excel in scenarios characterized by sparse obstacles, and its efficacy in such contexts has already been substantiated. Consequently, it becomes pertinent to investigate its performance as the density of obstacles in the map increases. To address this, we have generated a series of random maps featuring varying densities of obstacles across different dimensions. Through this endeavor, we aim to evaluate how the acceleration ratio evolves in response to escalating obstacle densities.

To quantify the obstacle density across various maps, we introduce the concept of the collision ratio between connections of two randomly selected free grid points within the grid space $\mathcal{C}_{\mathcal{N}}$. This collision ratio, denoted as Φ and ranging within the interval $[0, 1]$, serves as an indicator. It approaches 0 when sparse obstacles are prevalent and approaches 1 under conditions of dense obstacle presence. Notably, Φ remains independent of the grid space's scale or dimensionality.

In the forthcoming discussions, our focus centers on observing how the acceleration ratio evolves with increasing Φ, within the scope of 2D, 3D, and 4D random grid maps. The dimensions of these maps are 100^2, 100^3, and 100^4 respectively, with corresponding obstacle scales of 5^2, 5^3, and 5^4. Visual representations of certain 2D and 3D random maps, along with their respective Φ values, are presented in Fig. 11.

As depicted in Fig. 12, the acceleration ratio exhibits an upward trend with increasing Φ. Notably, when Φ approaches 1, the ratio approximates 1.2. This is because JOB requires an additional step to check whether a grid belongs to a block, in comparison to the raw LOS check. The associated computational cost can exceed the savings achieved by using blocks, especially when there are only a few blocks available. This signifies that JOB's efficiency diminishes as obstacle density increases, and its performance demonstrates negligible adverse effects when contrasted with raw line-of-sight (LOS) checks, up until Φ nears 0.8 (80%). Hence, JOB remains effective and acceptable in performance until the map's obstacle density reaches approximately 80%.

D. Experiment on Real Scenarias

In this section, we investigate how JOB accelerates path planning in real scenarios. We employ a Turtlebot 2 wheeled robot (Fig. 13(a)) as the mobile platform for mapping and path planning within an indoor environment (Fig. 13(b)). We execute several classic path planning algorithms with LOS checks on the actual map, initializing them with 10 random starting and target points. We then record the ratio between their execution times with and without JOB. To observe the differences at various resolutions, we employ maps with resolutions of 0.1 m, 0.05 m, and 0.01 m. The results are presented in Fig. 14.

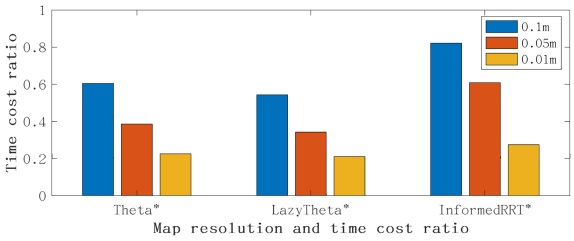


Fig. 14. This figure shows how JOB accelerates classic path planning in real world grid map with different resolution.

As illustrated in Fig. 14, JOB accelerates all three algorithms significantly, with the degree of acceleration increasing as the map resolution becomes higher. This phenomenon is attributed to the fact that, at higher resolutions, more free grids need to be checked during the raw LOS check process.

V. Discussion and Conclusion

We have proposed a novel method to accelerate the LOS check process in maps of any dimensionality, particularly effective in large-scale maps with sparse obstacles. The key principle of our method is to spliting passable space into blocks and speed up LOS check by jump over blocks. We choose block because it's convenient to store and require low computational resource to determine it's intersection with line.

The method consists of two main steps: block detection and LOS check acceleration with blocks. In the block detection step, only one parameter is required, namely the minimum block width (τ). The block detection process involves determining the centers of blocks, which are local maxima in the distance map and maintain a minimum distance from other block centers. Subsequently, the blocks are expanded simultaneously in all directions until no further expansion is possible.

Essentially, our method transfers the grid state check from an online process to an offline process, where blocks serve as a means to store the results of the grid state checks. This trade-off of space for time significantly improves the efficiency of the LOS check process. To avoid the need for block detection after each map loading, we save the blocks in binary files. The storage size required to save the blocks of the mentioned map dataset is minimal, only several kilobytes. This trade-off further optimizes the overall performance by reducing the computation needed for block detection.

Our investigation delved into the interplay between τ and the efficiency of acceleration. The experimental findings underscore that with an increase in τ, the acceleration ratio also rises. Concurrently, the time cost associated with block detection decreases. This underscores the significance of striking a balance that takes into account both block detection (precomputation) and LOS checks (online computation). The optimal choice of τ hinges upon achieving an equilibrium between these two essential components of the acceleration process.

Our results demonstrate that our method outperforms octomap in terms of block generation. On average, our method generates only 1/10 of the blocks generated by octomap while achieving better performance. Additionally, the mean detection time of our method is one-fifth of the time taken by octomap's block detection. Comparing our method to raw LOS check, our method reduces the time cost to 1/6 to 1/5, significantly improving the efficiency of the LOS check process. Additionally, it approximately halves the time cost of several classic path planning algorithms on real maps.

It's important to highlight certain limitations associated with JOB. Firstly, its reliance on precomputation (block detection) renders it less suitable for deployment in highly dynamic grid spaces. Additionally, its efficiency diminishes as the density of obstacles increases. Consequently, employing JOB in maps characterized by very dense obstacles (as demonstrated by our experiments, with $\Phi > 0.8$) becomes impractical and yields negligible benefits.

In the future, it will be crucial to apply our method to path planning and other domains that involve LOS checks. It would be interesting to evaluate how our method can enhance the efficiency of these tasks and explore its potential for further optimization and applications.

Acknowledgment

The first author thanks Lu Zhu for her encouragement and support during the consummation of JOB.

References

[1] B. H. Meng, I. S. Godage, and I. Kanj, "RRT*-based path planning for continuum arms," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 6830–6837, Jul. 2022.

[2] È. Pairet, C. Chamzas, Y. Petillot, and L. E. Kavraki, "Path planning for manipulation using experience-driven random trees," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 3295–3302, Apr. 2021.

[3] Z. Jian, S. Zhang, S. Chen, Z. Nan, and N. Zheng, "A global-local coupling two-stage path planning method for mobile robots," *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5349–5356, Jul. 2021.

[4] S. Nayak, M. Paton, and M. W. Otte, "A heuristic-guided dynamical multi-rover motion planning framework for planetary surface missions," *IEEE Robot. Automat. Lett.*, vol. 8, no. 5, pp. 2542–2549, May 2023.

[5] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2015, pp. 2951–2957.

[6] C. Zhang, Y. Tang, and H. Liu, "Late line-of-sight check and partially updating for faster any-angle path planning on grid maps," *Electron. Lett.*, vol. 55, no. 12, pp. 690–692, 2019.

[7] C. Zhang, Y. Tang, L. Zhou, and H. Liu, "Late line-of-sight check and prioritized trees for path planning," *Appl. Sci.*, vol. 9, no. 17, 2019, Art. no. 3448.

[8] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning," in *Proc. 10th Workshop Algorithmic Found. Robot.*, Springer, 2013, pp. 365–380.

[9] D. Harabor and A. Grastien, "An optimal any-angle pathfinding algorithm," in *Proc. Int. Conf. Autom. Plan. Scheduling*, 2013, pp. 308–311.

[10] S. Oh and H. W. Leong, "Edge N-level sparse visibility graphs: Fast optimal any-angle pathfinding using hierarchical taut paths," in *Proc. 10th Annu. Symp. Combinatorial Search*, 2017, pp. 64–72.

[11] D. Brewer and N. R. Sturtevant, "Benchmarks for pathfinding in 3D voxel space," in *Proc. Symp. Combinatorial Search*, 2018, pp. 143–147.

[12] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, vol. 34, pp. 189–206, 2013.

[13] J. Vörös, "Low-cost implementation of distance maps for path planning using matrix quadtrees and octrees," *Robot. Comput.- Integr. Manuf.*, vol. 17, no. 6, pp. 447–459, 2001.