Efficient Collision Detection in Sampling-Based Path Planning via Candidate Obstacle Filtering by Sorting Axis-Aligned Boundaries

Yechun Ruan
Institute of Future Networks
Southern University of Science and Technology
Shenzhen, China
ruanyc@mail.sustech.edu.cn

***Abstract*—Collision detection is considered to be the most expensive computational bottleneck in sampling-based path planning algorithms. In this report, a simple and efficient collision detection mechanism is proposed, which performs binary search based on sorted axis-aligned boundaries to obtain candidate obstacles, thus avoiding collision detection for obstacles that will never collide. This mechanism is referred to as the Candidate Obstacle Filtering (COF) mechanism in the following. The key idea of COF mechanism is that an obstacle whose right boundary is on the left side of the left boundary of the object must not collide with the object, and similarly for the left boundary. The path planning algorithm based on Rapidly-Exploring Random Tree was constructed to be used for experiments in the random world map, while the brute-force collision detection and the proposed candidate obstacle filtering collision detection mechanism were implemented separately. The experiments show that the proposed mechanism optimizes the number of obstacles to be detected in a single collision detection from $O(n)$ to $O(1)$, and the additional time overhead is negligible. The resulting performance improvement is proportional to the individual obstacle collision detection time, the number of obstacles, and the number of collision detection executions. The experimental code is available at https://github.com/Ryyyc/EEE5058-Introduction-to-Information-Technology.**

***Index Terms*—Path Planning, Collision Detection, Sorted Axis-Aligned Boundaries, Binary Search**

I. Introduction

PATH planning is a fundamental problem in robotics and autonomous systems, where the goal is to find a collision-free path for a robot from a start to a goal configuration in a given environment. There are four main categories of path planning algorithms that are commonly used to solve path planning problems: artificial potential fields (APF), graph-based, sampling-based and optimization-based algorithms. APF and graph-based algorithms are reactive methods that use a potential function or a graph to guide a robot towards the goal while avoiding obstacles. Sampling-based algorithms are a popular approach to path planning, where random configurations are sampled from the environment and connected to form a roadmap. Optimization-based algorithms formulate the path planning problem as an optimization problem and solve it using techniques such as gradient descent or nonlinear programming.

Collision detection is a crucial step in sampling-based path planning to ensure that the generated path does not intersect with the obstacles in the environment, and it is widely considered to be the major computational bottleneck in sampling-based path planning (see, e.g., [1]). Although the worst-case runtime complexity of individual collision detection is constant, it is far greater than the other steps of the algorithm, such as random point sampling. The brute-force collision detection approach applied to all obstacles compounds the time-consuming nature of the process.

A key observation is that obstacles with right boundaries to the left of an object's left boundary will not collide with it. Leveraging this insight, I conducted binary search on sorted axis-aligned boundaries to eliminate almost all detection of candidate obstacles that will not collide, resulting in a significant acceleration of the collision detection process. I compared the brute-force collision detection with the proposed candidate obstacle filtering collision detection mechanism in the Rapidly-exploring Random Tree(RRT)[2] algorithm, testing the time consumed by initialization of priori information (i.e., sorted axis-aligned boundaries) and collision detection. The experiments demonstrated that, as the number of obstacles increases, the brute-force collision detection time consumption increases rapidly, while the proposed candidate obstacle filtering collision detection mechanism slows down the growth in time consumption. The rest of the report is organized as follows: section II presents the work related to the collision detection algorithm in sampling-based path planning, and section III describes the proposed candidate obstacle filtering mechanism. Experiments and results are placed in Section IV. Discussion and summary are placed in Sections V and VI, respectively.

II. Related Work

There are several approaches to collision detection, including geometric shape representation and occupancy grids. Geometric shape representation involves representing the obstacles and the robot as simple shapes, such as spheres, axis-aligned bounding boxes, or polygons, and checking for intersections between them using algorithms such as the Separating Axis Theorem (SAT)[3] or the Gilbert-Johnson-Keerthi (GJK)[4]

algorithm. On the other hand, occupancy grids divide the environment into a grid of cells and mark each cell as either occupied or free. The robot's trajectory can then be checked against the occupancy grid to detect any collisions.

Some methods use precomputed collision maps or distance fields to accelerate collision detection, while others use knowledge of the robot's kinematics or the environment's geometry to prune collision checks. Karras and Aila [5] presented a fast parallel algorithm for constructing high-quality bounding volume hierarchies (BVHs) for large-scale scenes. Bialkowski et al. [6] proposed the use of safety certificates to bound the probability of collision along a given path, which leads to more efficient collision checking by efficiently pruning the search space. Zesch et al. [7] utilized neural networks to learn a collision detection function for deformable objects.

III. Methodology

In this report, I only consider obstacle detection in static 2D environments and focus on using circle and axis aligned bounding boxes to represent obstacles.

A. Brute Force Collision Detection

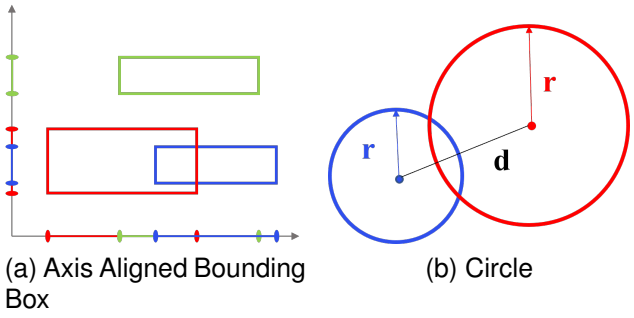


Fig. 1. Illustration of collision detection on simple primitives

For the input object that requires collision detection query, the brute force collision detection traverses all obstacles and determines whether each obstacle intersects the object. An illustration of collision detection on simple primitives is shown in Figure 1. For circle, two circles intersect if the distance between their centers is less than the sum of their radii. For axis-aligned bounding box, each box is defined by n intervals, one for each state dimension. Two boxes intersect only if all their corresponding intervals intersect.

B. Candidate Obstacle Filtering by Sorting Axis-Aligned Boundaries

The candidate obstacle filtering mechanism algorithm is divided into two parts: initialization of the priori information and querying the candidate obstacles that need collision detection. In the following description, we conventions d denotes the number of state dimensions and n denotes the number of obstacles.

During the initialization of the algorithm, the axis-aligned boundaries are first obtained from the obstacles information, specifically, for each state dimension, each obstacle has axis-aligned left and right boundaries (the same concept as axis-aligned bounding boxes), and this information is stored in $obs\_l\_bound^{d\times n}$ and $obs\_r\_bound^{d\times n}$.

Perform ascending sorting on each state dimension of $obs\_l\_bound^{d\times n}$ and $obs\_r\_bound^{d\times n}$ to obtain the sorted boundary data $obs\_l\_bound\_sorted^{d\times n}$, $obs\_r\_bound\_sorted^{d\times n}$, and index information $obs\_l\_bound\_sorted\_idx^{d\times n}$, $obs\_r\_bound\_sorted\_idx^{d\times n}$ (can be obtained by $argsort$). The time consumption for the initialization of the priori information is negligible.

For the input object that requires collision detection query, first get the object's axis-aligned boundaries $obj\_l\_bound^{d\times 1}$, $obj\_r\_bound^{d\times 1}$. For the i-th dimension, use $obj\_l\_bound[i]$ to perform the $binary\_search\_left$ on $obs\_r\_bound\_sorted[i]^{1\times n}$, returns l_start_idx, that is, the index of the first element in $obs\_r\_bound\_sorted[i]^{1\times n}$ that is greater than or equal to $obj\_l\_bound[i]$. $candidate\_l$ is set to an array of elements from l_start_idx to the last one in $obs\_r\_bound\_sorted\_idx[i]^{1\times n}$ (Equivalent to slicing $[l\_start\_idx :]$ in python scripts). Correspondingly, $binary\_search\_right$ is performed on $obs\_l\_bound\_sorted[i]^{1\times n}$ using $obj\_r\_bound[i]$ to get r_end_idx, that is, the index of the first element in $obs\_l\_bound\_sorted[i]^{1\times n}$ that is greater than $obj\_r\_bound[i]$. $candidate\_r$ is set to an array of elements from the first one to $r\_end\_idx - 1$ in $obs\_l\_bound\_sorted\_idx[i]^{1\times n}$ (Equivalent to slicing $[: r\_end\_idx]$ in python scripts). The candidate obstacles in the i-th dimension is the intersection of $candidate\_l$ and $candidate\_r$. The final candidate obstacles is the intersection of the candidate obstacles on all state dimensions.

The demonstration of finding the x-axis candidate obstacle in 2D environment is shown in Figure 2.

In practice, for any state dimension, when computing candidate obstacles, if the number of candidate obstacles in that dimension is less than a threshold (e.g., 5), the candidate obstacles in that dimension can be returned immediately as the result. Compared with performing binary search and intersection operations in the remaining dimensions, the time consumption of performing collision detection for a small number of obstacles is more worthwhile.

IV. Experiments and Results

This section will describe the results of the experiments on the random world and simple maps.

A. Experimental Environment

In this report, the experimental platform is Windows 11 and the processor is AMD Ryzen 7 5800H. The algorithm is implemented in the Python 3.10, and the functions are timed using the Python Profiler $cProfile$.

B. Experiments in the random world

The experiments to evaluate the performance of the algorithm were conducted mainly in the random world and the relevant configurations are as follows:

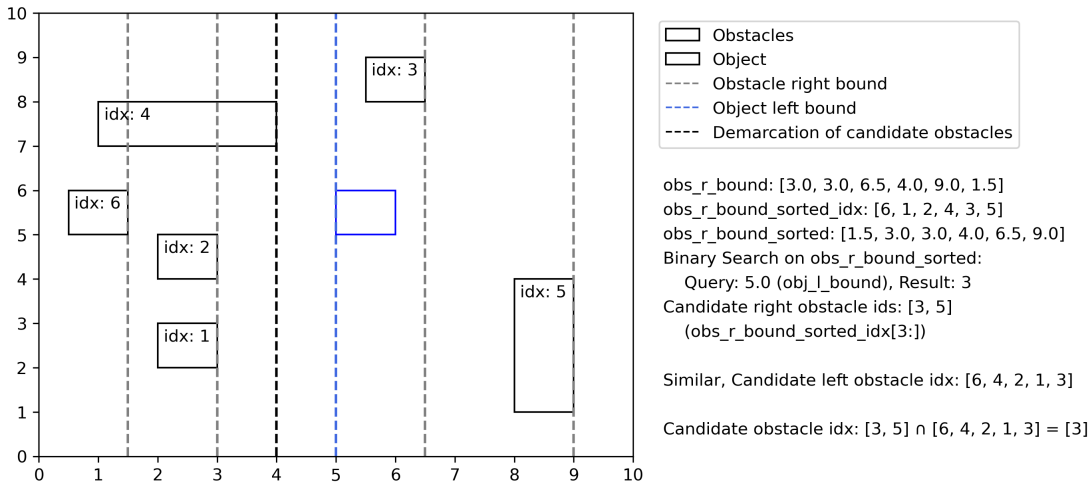
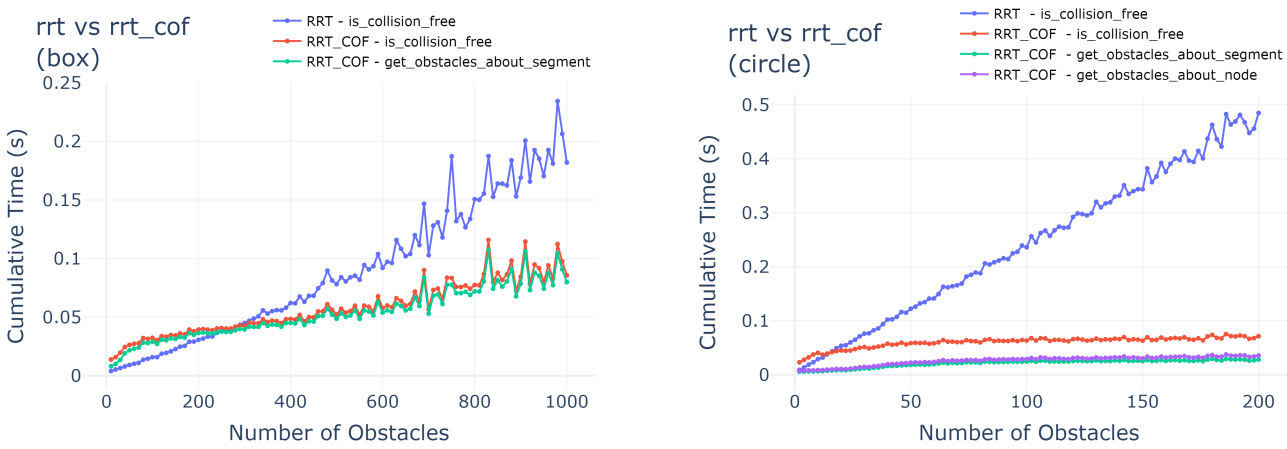


Fig. 2. Illustration of the demonstration of finding the x-axis candidate obstacle in 2D environment



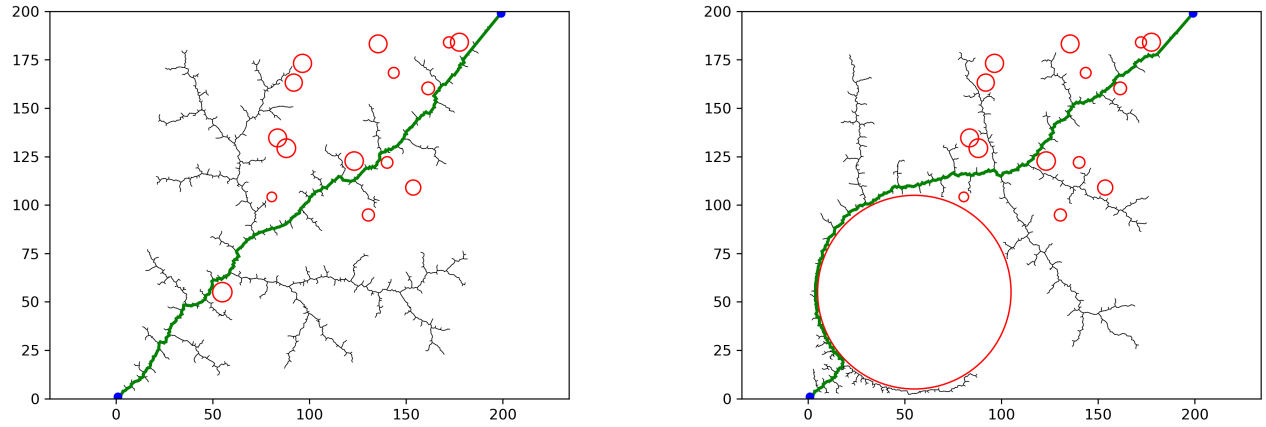
(a) Axis Aligned Bounding Box (b) Circle

Fig. 3. Time consumption statistics of the RRT (brute-force collision detection) and RRT_COF (this report, Candidate Obstacle Filtering) functions for different number of obstacles in the random world problem. The functions $get\_obstacles\_about\_segment$ and $get\_obstacles\_about\_node$ are called inside the function $is\_collision\_free$

1) map config
   - $play\_area = (0, 600, 0, 600)$
   - $rnd\_area = (10, 590, 10, 490)$
   - $box\_obstacle\_constraint = (min\_width, min\_height, max\_width, max\_height) = (2, 6, 2, 6)$
   - $circle\_obstacle\_constraint = (min\_radius, max\_radius) = (2, 6)$
   - $map\_cnt\_per\_obstacle\_num = 10$
   - $num\_box\_obstacles = [10, 20, ..., 1000]$
   - $num\_circle\_obstacles = [2, 4, ..., 200]$
2) robot config
   - $start = (1, 1)$
   - $goal = (599, 599)$
   - $step\_length = 3$
   - $robot\_radius = 1$
3) RRT config
   - $max\_iter = 50000$
   - $goal\_sample\_rate$[1] $= 0.05$

For axis-aligned bounding box and circle, 1000 maps are generated respectively, and each map contains obstacles of only one of the geometry type. For axis-aligned bounding box (circle), the number of obstacles in the map follows the setting of $num\_box\_obstacles$ ($num\_circle\_obstacles$), and for each item in $num\_box\_obstacles$ ($num\_circle\_obstacles$), 10 maps are generated. The algorithm time consumption statistics are obtained by running the algorithm 5 times repeatedly on each map to take the average. Figure 3 shows the experimental results, with axis-aligned bounding box on the left and circle on the right. The horizontal axis of Figure 3 is the number of obstacles in the map, and the vertical axis is the time consumption of the function, which is obtained by averaging the cumulative time of the function over all maps with the same number of obstacles.

[1]the probability that the goal will be selected when random point sampling is performed



(a) Map01 (b) Map02

Fig. 4. Visualization of solved scene maps.
(a) Map01: number of iterations is 2041, time consumption of function $is\_collision\_free$: RRT 0.046 s, RRT_COF(this report) 0.039 s, 15.2% speedup.
(b) Map02: number of iterations is 8446, time consumption of function $is\_collision\_free$: RRT 0.127 s, RRT_COF(this report) 0.091 s, 28.3% speedup.

Meanwhile, the average number of obstacles for collision detection per iteration was recorded, and this value was less than 10 for the proposed candidate obstacle filtering mechanism on all maps, and less than 1 for maps with more than 200 obstacles.

The experimental results demonstrate that as the number of obstacles on the map increases, the time consumption of the brute force collision detection increases rapidly, while the collision detection time consumption of the proposed candidate obstacle filtering mechanism increases slowly, which results in a significant performance improvement. It is attributed to the fact that the candidate obstacle filtering mechanism optimizes the number of obstacles to be detected from $O(n)$ to $O(1)$, while the additional time overhead is negligible. Comparing the time consumption of the algorithm under maps with different obstacle types shows that the more complex the obstacle (corresponding to the longer time for individual obstacle collision detection), the larger the performance improvement brought by the proposed candidate obstacle filtering mechanism.

C. Experiments in the simple map

Intuitively, the more collision detection queries (equivalently, the more iterations), the more significant the performance improvement from the proposed candidate obstacle filtering mechanism. To test this conjecture, I conducted a simple test on two small maps ($Map01$, $Map02$), which differ only in the radius of the obstacle with the circle center coordinates at $(55, 55)$, which are 5 and 50, respectively. The larger obstacle radius makes the RRT algorithm "go around", which leads to an increase in the number of iterations. The solved scene maps are visualized in Figure 4. The proposed candidate obstacle filtering mechanism results in 15.2% and 28.3% improvement in collision detection performance on $Map01$ and $Map02$, respectively. This experiment confirms the conjecture mentioned above.

V. Discussion

A. High-dimensional Space

The proposed candidate obstacle filtering mechanism naturally adapts to high-dimensional spaces, which do not break down due to the increase in the number of dimensions.

B. Dynamic Environment

The proposed candidate obstacle filtering mechanism still works properly when faced with a dynamic environment, i.e., an obstacle addition and removal event occurs. When an obstacle is added, the axis aligned boundary information can be updated by first performing a binary search to find the position of the new obstacle's axis aligned boundary in the array and then inserting it. The complexity of the binary search is $O(logn)$. To avoid mass movement of array elements, the array can be saved in chunks so that only the elements within the chunks are moved. The operation of removing an obstacle is similar to the inverse process of adding. Collision detection queries are not affected.

VI. Conclusion

The performance of obstacle detection greatly affects the performance of sampling-based path planning algorithms such as RRT. In this report, the Candidate Obstacle Filtering (COF) mechanism based on sorted axis-aligned boundaries and binary search is proposed, which is simple and efficient. COF mechanism optimizes the number of obstacles to be detected from $O(n)$ to $O(1)$, while the additional time overhead is negligible, which leads to performance improvement. The experiments show that the performance improvement brought by the proposed COF mechanism becomes larger as the obstacle complexity (individual obstacle collision detection time), the number of obstacles, and the number of collision detection executions(equivalent to number of iterations) increase.

REFERENCES

[1] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. doi: 10.1017/CBO9780511546877.

[2] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[3] Gino Van Den Bergen. *Collision detection in interactive 3D environments*. CRC Press, 2003.

[4] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988. doi: 10.1109/56.2083.

[5] Tero Karras and Timo Aila. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 89–99, 2013.

[6] Joshua Bialkowski, Michael Otte, Sertac Karaman, and Emilio Frazzoli. Efficient collision checking in sampling-based motion planning via safety certificates. *The International Journal of Robotics Research*, 35(7):767–796, 2016.

[7] Ryan S Zesch, Bethany R Witemeyer, Ziyan Xiong, David IW Levin, and Shinjiro Sueda. Neural collision detection for deformable objects. *arXiv preprint arXiv:2202.02309*, 2022.