2020

Extensions of motion planning algorithms

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

EXTENSIONS OF MOTION PLANNING ALGORITHMS

by

XINWEI ZHANG

B.S., Changzhou University, 2018

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

2020

Approved by

First Reader

Roberto Tron, Ph.D.
Assistant Professor of Mechanical Engineering
Assistant Professor of Systems Engineering

Second Reader

Sean B. Andersson, Ph.D.
Professor of Mechanical Engineering
Professor of Systems Engineering

Third Reader

Wenchao Li, Ph.D.
Assistant Professor of Electrical and Computer Engineering
Assistant Professor of Systems Engineering

Acknowledgments

I would like to express my very great appreciation to Professor Roberto Tron, my research supervisors for his academic guidance and enthusiastic encouragement.

I would also like to thank Professor Sean Andersson, and Professor Wenchao Li, for their valuable comments and suggestions for this research work.

Advice and help were also given by Anna Masland, my Master's programs administrator. She has been a great help to me during my graduate studies.

I would like to offer my special thanks to Cynthia Korhonen, Brendan McDermott, and Siyi Fan, for their great help in my thesis revisions and professional guidance in writing and formatting.

Finally, I would like to thank my family for their support and sponsorship. My graduate education would not have been possible without their constant help.

Xinwei Zhang

Master Student

ME Department

EXTENSIONS OF MOTION PLANNING ALGORITHMS

XINWEI ZHANG

ABSTRACT

Sample-based motion planning algorithms can be applied to a broad range of circumstances in motion planning of robotics. Though sample-based algorithms are able to generate collision-free paths without the information of obstacles, they still have two weaknesses: one is that it is challenging to pass through narrow passages, which might result in path generation failures; the other is that a fixed search scope might lead to a waste of computational resource, which would result in low efficiency. In order to limit the search scope and improve the efficiency of paths generation of narrow passages, obstacles in the configuration space can be used to constrain the sampling scope and guide sampling. This thesis develops Obstacle Activation to identify polygonal obstacles that can be used to limit the search scope and Obstacle Exploration to obtain free points in a non-polygonal configuration space. These two methods are combined with several improved sampling-based algorithms to achieve the acceleration of the shortest path and feasible path generation. Finally, through a large number of simulations, it is verified that Obstacle Activation and Obstacle Exploration can effectively improve the speed of sampling-based algorithms significantly on both challenging scenarios with narrow passages and general cases.

Contents

List of Tables

List of Figures

List of Abbreviations

OA	.....	Obstacle Activation
OE	.....	Obstacle Exploration
PRM	.....	Probabilistic Roadmap
RRT	.....	Rapidly-exploring Random Trees
SD	.....	Standard Deviation

Chapter 1

Introduction

1.1 Introduction

During the past decade, robotic sampling-based motion planning algorithms have received increasing attention because they can generate collision-free paths quickly without being tailored to a specific type of environment (e.g., polygon) and they can be easily extended to a high-dimensional space. In particular, the Probabilistic RoadMap (PRM) (Kavraki and Latombe, 1994) and the Rapidly-exploring Random Tree (RRT) (LaValle, 1998) are classic sampling-based algorithms that have formed the basis for many different variations that offer optimality and probabilistic completeness guarantees. Examples of such variations are PRM* (Karaman and Frazzoli, 2011), RRT* (Karaman and Frazzoli, 2011), RRG (Karaman and Frazzoli, 2011) and Informed RRT* (Gammell et al., 2014). These algorithms have found real applications on robotic platforms (Devaurs et al., 2014; Kuwata et al., 2009) and other areas (Wedge and Branicky, 2011; Ju et al., 2011; Santos et al., 2018).

However, flexibility, which is the advantage of the above algorithms, it is also a disadvantage, since they do not make any use of prior knowledge of the map even when this is available. This includes two cases. The first is when the configuration space is much larger than the space to be explored. For example, for a self-driving car, finding a route in a city is the space that must be explored, but if the given configuration space is a map of the entire state, it is excessive for the particular need. However, rather than finding a suitable configuration space in advance, it is better to find a

suitable mechanism that enables the solver to adapt the large map to the reduced spatial need. Second, it is difficult to effectively explore space in environments where the spatial size varies greatly in different directions. For example, narrow passages that are as wide as the body of an unmanned vehicle, or planning a path for a drone in a dense forest, are more difficult for both.

1.2 Thesis overview

According to the geometric properties of obstacles in the configuration space, we propose a series of path planning algorithms guided by obstacle information and combine them with various improved sampling-based algorithms to derive new algorithms. Through testing and data analysis, we show that the proposed algorithms can efficiently find a path through both complex configuration space such as narrow passages and sparse environments with multiple obstacles.

In the second chapter, we propose PRM Trees, which uses the sampling strategy of PRM (Kavraki and Latombe, 1994), but connects samples only to the nearest node of each local tree, as in RRT (LaValle, 1998) and the maximum number of possible connecting local trees is preassigned. it can use free space samples efficiently and it is much faster than a similar algorithm, Multiple RRTs (Clifton et al., 2008), in complex environments. It also forms the basis for the obstacle-based algorithms in Chapter 4.

In the third chapter, we introduce Obstacle Activation method. It is a way to record the obstacles encountered by connecting of any two already existing points in the graph and generate paths by using vertices which belongs to the recorded obstacles. Combining it with Lazy A* (Liu and Arimoto, 1992; Hart et al., 1968), our simulations show that Obstacle Activation can greatly reduce the computation time. We then combine Obstacle Activation with RRT and RRT*, respectively, to

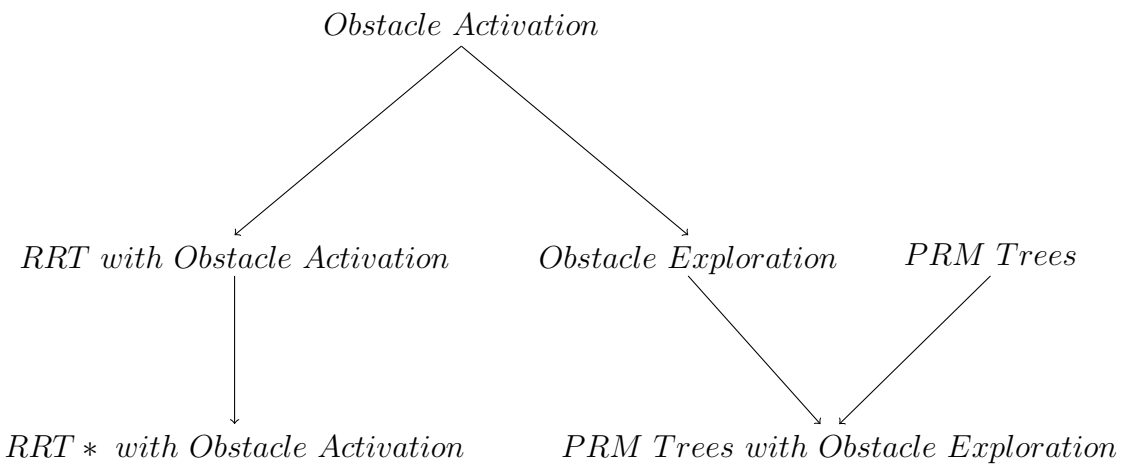


Figure 1·1: All the variations of sampling-based algorithm explored in the thesis.

reduce the computation time for getting the feasible path and the shortest path.

In the fourth chapter, in order to generalize the Obstacle Activation method to the environment of non-polygonal obstacles, we propose Obstacle Exploration to obtain free points around obstacles by exploring the obstacles. Combining it with PRM Trees, its running time is only one tenth of the running time of RRT even when used for navigating through extremely complicated narrow passages.

The sequence of development of the above sampling-based algorithms is shown in Figure 1·1.

In the thesis, all algorithms are implemented in MATLAB. The processor used for simulations is 1.4 GHz Quad-Core Intel Core i5, and the memory is 16 GB 2133 MHz LPDDR3.

1.3 Disclaimer

The results in the thesis are all empirical and limited to the conditions tested.

Chapter 2

PRM Trees

2.1 Related Works

Increasing the speed of path planning has been a major problem for a long time. Research in the late 20th century focused on PRM and its derivative algorithms. Since the beginning of the 21st century, RRT has proven to be computationally efficient. Because it requires no additional knowledge of the environment, RRT and its derivative algorithms have become the new mainstream research direction. The pseudocode of the algorithm is shown in Algorithm 1. $\mathcal{C}$ is the configuration space. The variable M is the number of loop executions. We denote the starting point as q_{Start}, and denote the step size, which is also named as incremental distance, as Δq.

Algorithm 1 RRT

1: **function** RRT(q_0,$\mathcal{C}$,Δq,M)
2: $\mathcal{T}$.Init(q_{Start}) ▷ Initialize the graph $\mathcal{T}$ by the start point q_{Start}.
3: **for** $i = 1$ to M **do**
4: $q_{rand} \leftarrow$ Rand($\mathcal{C}$) ▷ Sample a point, q_{rand} in the configuration space.
5: $q_{near} \leftarrow$ Nearest($q_{rand}, \mathcal{T}$) ▷ Find the nearest node q_{near} of the q_{rand}
6: $q_{new} \leftarrow$ Steer($q_{rand}, q_{near}, \Delta q$) ▷ Get a new node q_{new} by moving one step from q_{near} to q_{target}, with the incremental distance Δq.
7: Extend($\mathcal{T}, q_{near}, q_{new}$) ▷ If the segment between q_{near} and q_{new} is collision-free, add the q_{new} into the graph, and set its backpointer as q_{near}.
8: **end for**
9: **end function**

In order to increase the speed of motion planning, researchers have discovered

that the generation of feasible paths in configuration spaces can be accelerated by adding more trees. For example, RRT-Connect (Kuffner and LaValle, 2000), extends RRT from both the starting point and the end point at the same time. Triple-RRTs (Wang et al., 2010), adds another tree in the map, speeds up the process as well. And, Multiple RRTs (Clifton et al., 2008) extended this type of acceleration to an unlimited number of trees. Through reasonable inference, the efficiency of the algorithm can be improved as the number of trees increases, so increasing the number of trees can effectively enhance the efficiency of the algorithm.

The advantages of Multiple RRTs are as follows:

1. The number of trees can be adaptively changed during the process of the graph generation; and

2. Constantly trying to connect the trees greatly increases the probability of passing through narrow passages, and removing the limitation where RRT slowly expands in the narrow passages step by step.

However, when Multiple RRTs encounters a complex configuration space that must grow a large number of trees, its speed will be greatly reduced because each new node needs to be checked for its connectivity with all the trees, even if they are very far away, in which case the running speed would be greatly reduced. Assuming that the environment being explored is a large maze or a city map, each new node that is sampled uniformly will most likely become a new tree. When a node needs to check its connection with other points repeatedly, the algorithm loses the original advantages of RRT. Therefore, there exists an opportunity to improve Multiple RRTs.

2.2 PRM Trees

To address the drawbacks of Multiple RRTs, a novel sampling-based algorithm, PRM Trees, is proposed as shown in Algorithm 2. It retains the advantages of both PRM

and RRT.

In classical PRM, samples are generated uniformly across the entire environment. The algorithm keeps every sample that is not involved in a collision, and tries to connect to all of its neighboring samples, thus generating a generally loopy graph. The final path is then extracted using A* (Hart et al., 1968), which is time-consuming. On the other hand, one could use RRT or RRT* (Karaman and Frazzoli, 2011)); however, both of them bias the sampling around existing samples by keeping new samples only if they can be connected with a neighbor. Their graph is always maintained in the form of trees, hence removing the need to use A* to find a final path (one can simply follow backpointers toward the root).

PRM Trees uses the sampling strategy of PRM, but connects samples only to the nearest node of each local tree, as in RRT. The maximum number of possible connected local trees can be preassigned, which is different from the Multiple RRTs algorithm. We denote the maximum number as K. The effect of limiting the maximum is that the algorithm grows many separate trees that incrementally merge together as the sampling proceeds. When a node can merge the trees of the start and goal, it means that a feasible path has been found. Like RRT, it can stop automatically when a feasible path is found. A pictorial example of this process in a configuration space with concave polygonal obstacles is shown in Figure 2·1. Compared with the same process of RRT in Figure 2·2, one finds that PRM Trees requires far fewer samples than RRT.

The pseudocode shown in 2 shows how the PRM Trees works. The algorithm first assigns $Tree$ data structures to the start point q_{Start} and the goal q_{Goal}, and records them in Trees.

In lines 5 to 14, the algorithm randomly samples a collision-free point q_{rand} from the configuration space through by the function `RAND` each time and performs `MERGE` to

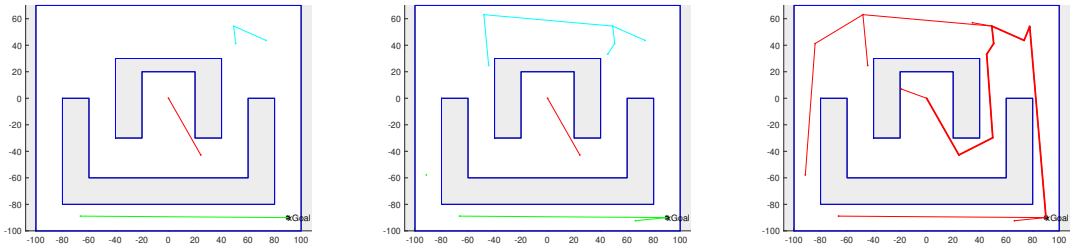
Algorithm 2 PRM Trees

```
1: function BUILD_PRM_TREES(q_Start, q_Goal, K, C)
2:     q_Start.Tree ← 1
3:     q_Goal.Tree ← 2
4:     Trees.Init(q_Start, q_Goal)
5:     while q_Start.Tree ≠ q_Goal.Tree do
6:         q_rand ← RAND(C)
7:         if IsFree(q_rand) then
8:             NumberOfTrees ←COUNTTREES(Trees)
9:             N ←MIN(K, NumberOfTrees)
10:            for i = 1 to N do
11:                Trees ← MERGE(Trees, q_rand)
12:            end for
13:        end if
14:    end while
15:    Path = FIND_PATH(Trees,q_rand)
16:    return Path
17: end function
```

merge with the nearest point of the existing tree. Unlike Multiple RRTs, the number of trees checked is no longer unlimited, but a maximum of K trees are tried. K is often chosen to be a small constant, which can prevent a lot of redundant calculations. By introducing the variable K, the upper limit of the number of trees no longer needs to be limited as in Multiple RRTs so that all collision-free points can be retained. In this case, the efficiency of the algorithm is further improved.

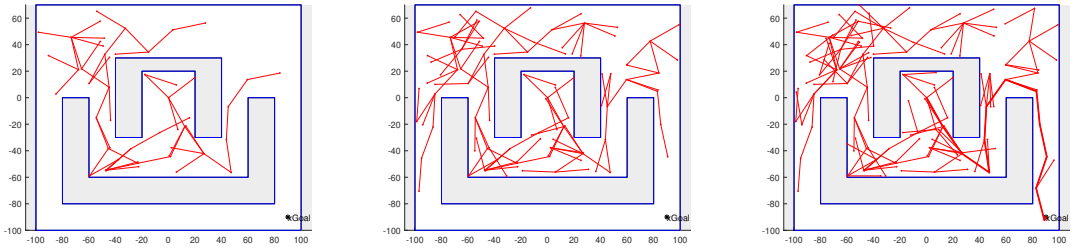
The function `MERGE` selects q_{rand}'s closest node belonging to the unchecked trees each time for the connection checker and repeat N times, where N is the minimum between K and the actual number of trees. The $Tree$ data structures that can be connected are modified to the smallest data structures within them; also, the backpointers are updated. If the q_{rand} cannot be connected to any tree, we create a new tree for the q_{rand} with q_{rand} as the root.

`FIND_PATH` in line 15 finds the path to the q_{Start} and q_{Goal} from the backpointers of the last q_{rand} after the fusion of trees by the last node, which can avoid the need



(a) 5 iterations (b) 10 iterations (c) 15 iterations

Figure 2·1: Graph construction of PRM Trees



(a) 50 iterations (b) 100 iterations (c) 133 iterations

Figure 2·2: Graph construction of RRT

to call A*, thereby improving the efficiency of this algorithm.

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	0.1566	0.0480	335.3820	66.3280
PRM Trees ($K = 2$)	0.0563	0.0278	461.8729	126.4650

Table 2.1: Performance comparison of one-hundred trials between RRT and PRM Trees

In order to show that PRM Trees can effectively improve efficiency, the simulation results of finding a path in a maze by Multiple RRTs (our reimplementation) and PRM Trees are presented in Table 2.2, which shows the mean value and the standard deviation of the running time and the path length planned by each algorithm. The proposed algorithm runs in about one-fifth of the previous most direct competitor.

The corresponding feasible paths generated by these two algorithms are shown in Figure 2·3.

At the same time, to analyze how the value of K affects the speed of the algorithm, 100 trials are performed in the above maze shown in Figure 2·3 for different K, and the mean and standard deviation of running time are compared. The overall trend is that the running time increases as the K increases, and the fastest running speed occurs at the minimum value of $K = 2$. At the same time, the algorithm's running speed will decrease when the value of K gradually exceeds the maximum number of local trees that can be established in a map. Eventually, for high K, the results will be the same as those of Multiple RRTs.

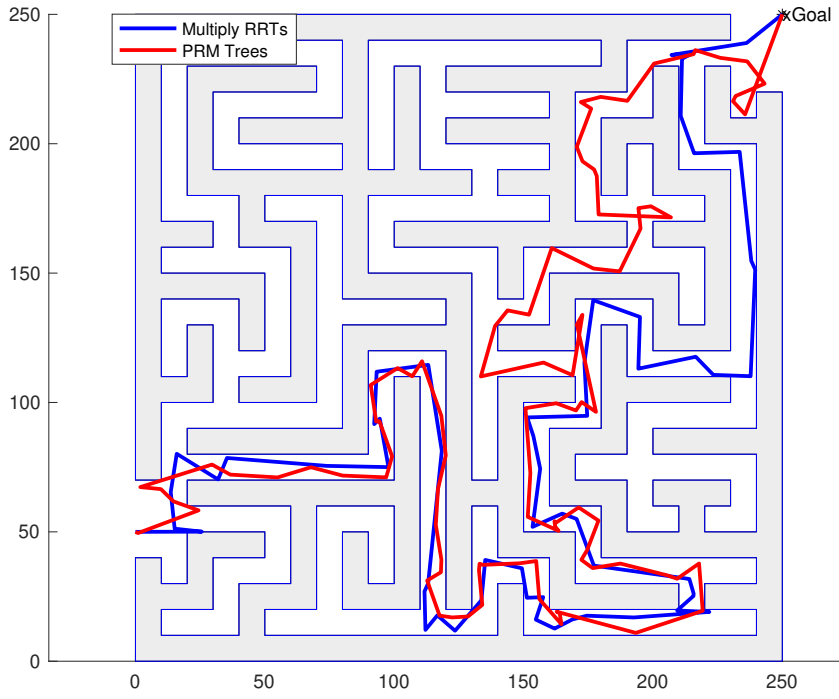


Figure 2·3: Paths generated by multiple RRTs and PRM Trees in a maze

PRM Trees can generate a path through narrow passages faster than RRT can

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
Multiple RRTs	8.4438	1.2057	976.4825	58.7035
PRM Trees ($K = 2$)	1.6205	0.6563	961.2369	50.5322

Table 2.2: Performance comparison of one-hundred trials between Multiple RRTs and PRM Trees in a maze

do in configuration space; a set of comparative experiments was performed. Figure 2·5 shows the path found by PRM Trees and RRT in a 50 x 50 units configuration space with a long narrow passage with a width of 0.2 unit. From the data in Table 2.3, we see that the speed of PRM relative to RRT has been greatly improved. For this configuration space, when PRM Trees samples some points in the area near the entrance of the narrow passage, the probability of finding a path can be increased, and it is not necessary to obtain samples within the passage. However, for RRT, it is necessary to repeatedly extend the nodes in the narrow passage to the area in the remaining part of the narrow passage, the probability of sampling the next node in the narrow passage is small, which makes it difficult for RRT to find feasible paths to pass narrow passages.

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	22.6551	32.0013	196.6483	36.7800
PRM Trees ($K = 2$)	2.1417	2.0403	165.9565	23.3473

Table 2.3: Performance comparison of one-hundred trials between RRT and PRM Trees in a configuration space with a narrow passage

The above cases show that the PRM Trees algorithm has various advantages, but it still has various problems. For example, there are still a large number of detours and nodes, which are unnecessary for the final paths, and eventually reduces the efficiency of our path-finding algorithm. Taking the above narrow passage case for analysis, for human beings, the upper left and lower right parts of the configuration space can be

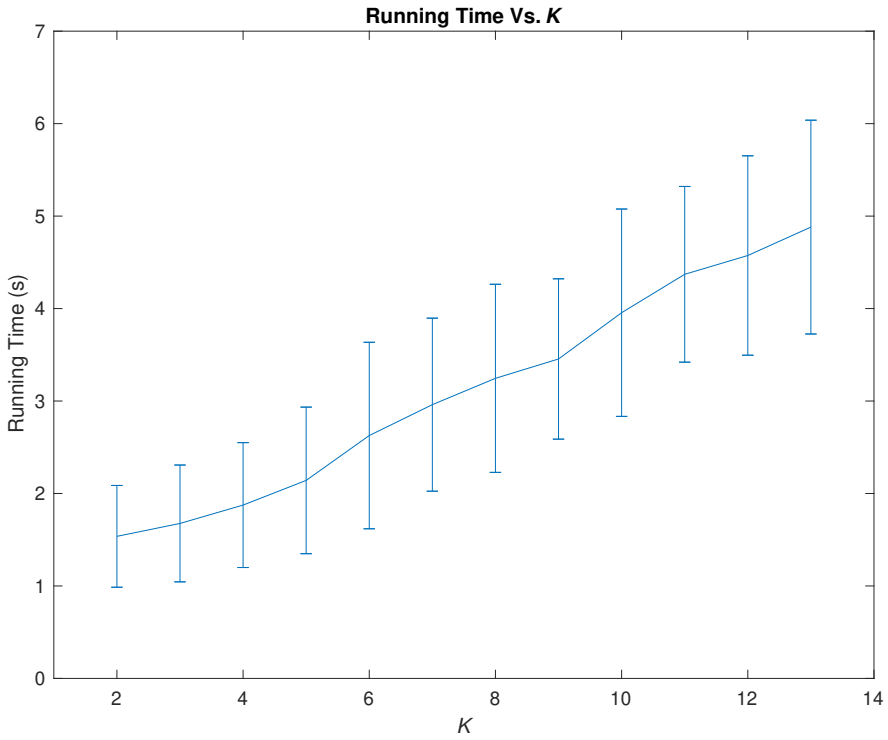


Figure 2·4: Running time Vs K of PRM Trees in a maze

directly ignored. There must be corresponding geometric knowledge behind human intuition that should be added to the algorithm to generalize and solve such problems. For example, for the passage of a straight narrow space, only the ends of the narrow space need to be found to pass through the narrow space, and there is no need for extra exploration of the interior of the narrow space if the geometrical information of obstacles can be utilized. Such a method can significantly improve the speed of the algorithm. To do so, it is necessary to obtain information about the obstacles. We explore how to use the geometric properties of the space in the next chapter (Chapter 3).

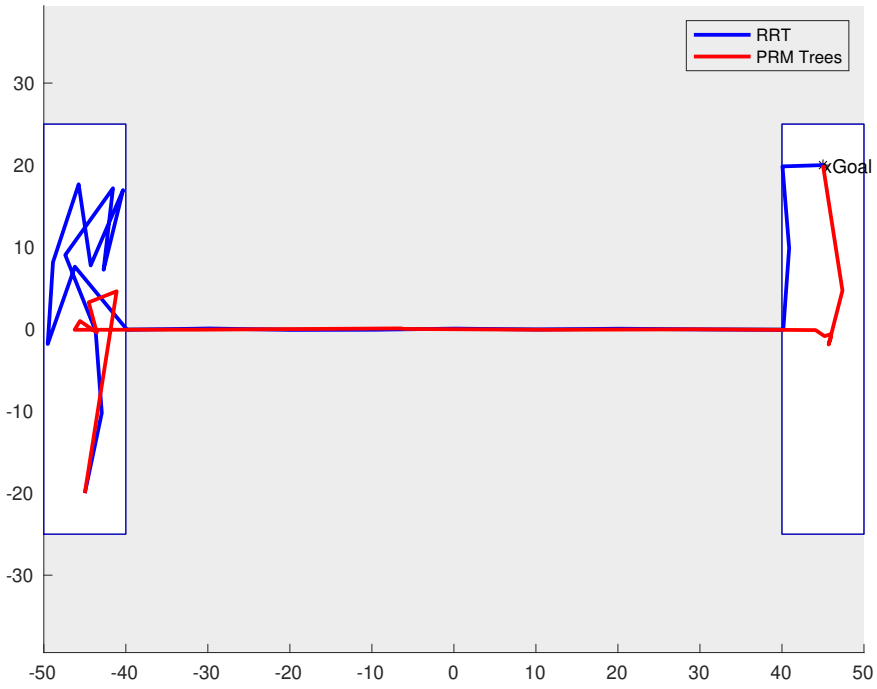


Figure 2·5: Paths generated by RRT and PRM Trees in a configuration space with a narrow passage (width = 0.2 unit)

Chapter 3

Obstacle Activation

3.1 Related work

Methods for limiting the sampling range or guiding path generation have been a significant research direction for sampling-based algorithms. At a higher level, the more complex the mathematical model or logic that is used, the higher the quality of the path that will be obtained and the higher the probability of convergence will be; however, the cost of each step will also be higher. There is a trade-off to consider. Examples of the information regarding obstacles, which has been utilized in motion planning, are as follows:

1. Use of auxiliary knowledge to generate bias to guide or limit the sampling scope.

    - Informed RRT* (Gammell et al., 2014) introduces a method for limiting the sampling scope. When a feasible path is found, it will limit the searching area to an ellipse and continue narrowing the scope until the optimal path is found. However, it does not take into account the relationship between the search area and configuration space. Moreover, before a feasible path can be found, this technique cannot guide the sampling, with a resultant cost in terms of wasted samples. Moreover, a prior limited spatial area does not ensure that the optimal path is included. Assume that the first discovered path of informed RRT* has a large number of detours, but that area occupied by the path is small. The formed search scope may miss

the optimal path without detours outside the search scope. In the most extreme case, an infinite length path can be formed within a limited area.

- TG-RRT* (Qureshi et al., 2015) utilizes both geometric incentres and geometric centroids to add bias to sampling. Its method is still to limit the sampling to a priori geometry by bias. Its idea is similar to Informed RRT* — when the map is too large, it uses a small and safe scope to guide the sampling. The cost of adaptability in a small area is small, but the cost is observed in the speed, which drops in large and complex environments.

2. Generate sampling around obstacles.

    - Retraction-Based RRT (Pan et al., 2010; Liangjun Zhang and Manocha, 2008) tries to produce paths near obstacles to pass through narrow passages, or utilizes obstacles to get a deformed Voronoi diagram to find feasible paths. However, it remains difficult to deal with concave obstacles and has no advantage in sparse environments. This algorithm needs more processing about the environment.

    - UOBPRM (Yeh et al., 2012) has a uniform distribution of the configurations near the obstacle surfaces. It is an improved version of OBPRM (Amato et al., 1998), and it is inspired by both Gaussian sampling (Boor et al., 1999) and Bridge test sampling (Hsu et al., 2003) (these terms are explained in more detail in Chapter 4). But it still does not offer a solution to the kinds of obstacles that are worth processing to find a path in a configuration space.

In fact, with regard to geometric-based path planning algorithms, previous researchers have done a significant amount of research on the geometric problems of motion planning; one such geometric-based path planning algorithm is the visibil-

ity graph (Lozano-Pérez and Wesley, 1979). However, due to the particularity of the sampling-based algorithm, many methods are difficult to apply. Generally, there are three difficulties that make it hard to apply existing methods to sampling-based algorithms:

1. Sampling methods make it challenging to handle available information. Take the commonly used RRT as an example; its steer function to create discrete steps makes it hard to utilize information like a given collision-free point in the configuration space.

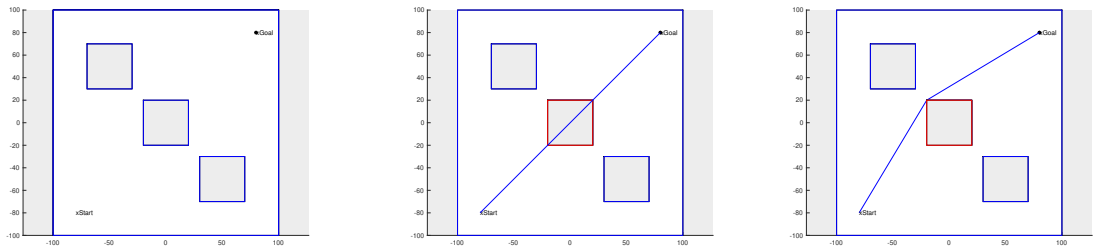
2. It is difficult for sampling-based algorithms to analyze the obtained nodes as a whole, and have to apply the existing methods locally to a single sample. When encountering a non-convex environment, they would have local minima and fail. For example, when performing RRT-A* (Li et al., 2014) in configuration space with a T-shaped obstacle, it would encounter local minimum and get stuck.

3. Some geometric methods are difficult to extend to high-dimensional space. The reason is that the geometric properties of the configuration space become more involved with increasing dimensions. For example, it is easy to identify a convex vertex in two-dimensional space, but not in three-dimensional space or higher, which will be discussed later in this chapter.

3.2 Motivation

To illustrate the basic principle of Obstacle Activation, We start with an intuitive example. Suppose there is a configuration space with several polygons like Figure 3·1 (a). A natural way to find a feasible path for this case is to check whether the starting point and the goal can be connected first, which reveals that the central obstacle is in the way. Then we can try to bypass the obstacle, thus generating a collision-free

path. It is worth noting that during the entire process, only the starting point, the goal, and the area near the middle obstacle are explored. The process does not need to consider other obstacles. This is the idea behind the Obstacle Activation strategy. Obstacles found while trying to connect any two already existing points in the graph are called activated obstacles.



(a) (b) (c)

Figure 3·1: The basic ideal behind the Obstacle Activation strategy

Overall, the logic of "connecting points, finding encountered obstacles, and trying new connections through the vertices of these obstacles" can ultimately confine the search scope within the limited area of the activated obstacles. This method is in contrast to the search scope based on hypothesized geometries of algorithms such as informed RRT* and RRT based on cylindrical sampling space (Yu et al., 2019). Although the Obstacle Activation method sometimes explores more space than the above algorithms because a well-designed prior limited spatial area based on repeated trials may include fewer obstacles, it can guarantee that the shortest path will be found.

In the following, we illustrate the implementation of Obstacle Activation in the context of a deterministic A* search and a randomized RRT algorithm.

3.3 Visibility Graph with Obstacle Activation

3.3.1 Visibility Graph and the Lazy A* algorithm

We first verify that the strategy of Obstacle Activation can speed up the path planning algorithm when combined with the strategy with a traditional geometric algorithm rather than a sampling-based algorithm, as it is easier to do so. Another reason is that we will use it as a comparison with sampling-based algorithms.

We first present Obstacle Activation under the assumption that the environment is composed of polygonal obstacles, and their vertices are known. Thus we use the visibility graph (Lozano-Pérez and Wesley, 1979) as the underlying search algorithm.

3.3.2 The basic reduced visibility roadmap algorithm

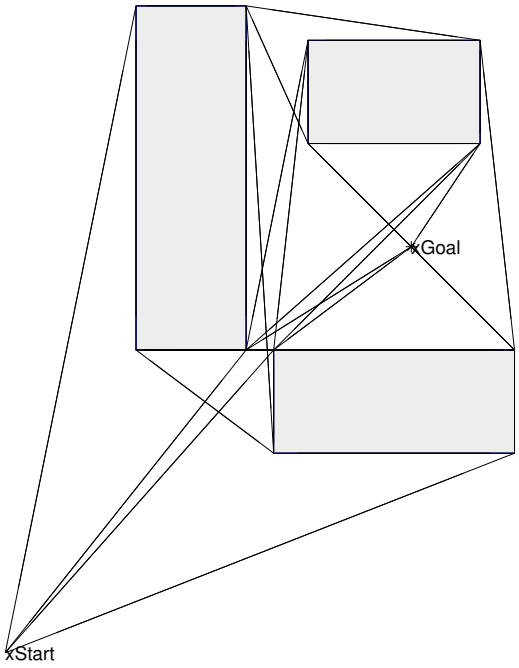


Figure 3·2: Reduced Visibility Map of three convex polygon obstacles

Visibility Map is a way to generate a roadmap by connecting the vertices of all polygonal obstacles in the space and removing the line segments that are in collision with the obstacles. This technique is typically used in the Euclidean plane. Because only the segments which can connect two nodes that are visible to each other remain on the map, and they represent the visibility between pairs of vertices, this kind of map is called a Visibility Map. In fact, for the environment of the obstacles that can be represented by polygons, the shortest path exists in the Visibility Map.

A Reduced Visibility Map (Lee, 1978; Ghosh and Mount, 1991; Pocchiola and Vegter, 1996) is a simplified map based on the Visibility Map that uses geometric principles on tangent lines to extract supporting segments and separating segments, thereby reducing the line segments in the map. Figure 3·2 shows a Reduced Visibility Map constructed in a configuration space with three convex polygons.

One way to find all the supporting segments and separating segments in a Visibility or a Reduced Visibility Map's obstacles is to remove all non-convex vertices of the obstacles and then remove all segments pointing into obstacles. The above method is applicable in two-dimensional space. In high-dimensional space, non-convex vertices cannot be obtained by directly calculating the angle between two edges in advance as in two-dimensional space, so that removing non-convex vertices is meaningless. However, removing all the segments pointing into the obstacle can meet the condition of removing non-convex vertices of different directions; therefore, it can still be used in high-dimensional space. In practice, as long as the line segment is extended a little bit at both ends, the segment collision checker can be performed to rule out the segment which does not meet the conditions above. There is no need to change anything else.

Search via the Lazy A* algorithm

A Reduced Visibility Map is typically used in conjunction with a Lazy A* algorithm to output the shortest path. The A* algorithm is a very classic heuristic algorithm. Different from the depth-first and breadth-first algorithms, it selects the node with the smallest value of $f(n)$ at each node n as the target of the next exploration, where

$$f(n) = h(n) + g(n) \tag{3.1}$$

$g(n)$ is the cost of the path from the start of the node n, and $h(n)$ is a heuristic function that evaluates the cost of the shortest path from the node n to the goal. The A* algorithm outputs the shortest path and stops when $f(x)$ indicates that there cannot exist other paths between the start and the goal.

The traditional Lazy A* algorithm is an A* algorithm applied to a Reduced Visibility Map where the edges from a vertex (other vertices visible from it) are computed only when a node is being expanded (line 8 of Algorithm 3).

3.3.3 Lazy A* with Obstacle Activation (Lazy Visibility Graph).

It is possible to reduce further the computations done by Lazy A* and remove all the vertices of unnecessary obstacles by combining the Reduced Visibility Map with Obstacle Activation. The algorithm first checks the segment between the start and goal points, records the obstacles and their vertices that appear on the segment to initialize the algorithm. Then check the visibility of each pair of vertices and join the queue in the same way for subsequent obstacles, which are found by the collision checker of separating or supporting line and continue to iterate until no more new obstacles are found. The obstacles added to the record are the activated obstacles. In this way, obstacles that do not intersect expanded edges will not be considered during path planning. As a byproduct, the searching scope is limited to the encountered

Algorithm 3 The A* algorithm

Add the starting node n_{start} to $\mathcal{O}$, and set $g(n_{start}) = 0$. ▷ Initialization
while $\mathcal{O} \neq \{\}$ **do**
    Pick n_{best} from $\mathcal{O}$ such that $f(n_{best}) \leq f(n)$ for all $n \in \mathcal{O}$.
    Remove n_{best} from $\mathcal{O}$ and add it to $\mathcal{C}$.
    if $n_{best} = q_{Goal}$ **then**
        break
    end if
    for $x \in$ neighbours of n_{best} that are not in $\mathcal{C}$ **do** ▷ Expand n_{best}
        if $x \notin \mathcal{O}$ **then**
            Set the value of $g(x)$ to $g(n_{best}) + f(n_{best}, x)$.
            Set the backpointer of x to n_{best}.
            Add x to $\mathcal{O}$ with value $f(x)$.
        else if $g(n_{best}) + f(n_{best}, x) < g(x)$ **then**
            Update the value of $g(x)$ to $g(n_{best}) + f(n_{best}, x)$.
            Update the backpointer of x to n_{best}.
        end if
    end for
end while

obstacles and the minimum space between them. The start and the goal are also included, which makes the shape of the scope look like an irregular spindle. When dealing with environments with many obstacles and especially sparse environments, this algorithm is much faster than the basic Lazy A*.

The algorithm is summarized in detail in Algorithm 4; in the pseudocode, $\mathcal{V}$ contains the start, the goal, and the vertices of the activated obstacles. $\mathcal{V}_{new}$ is the vertices of the newly encountered activated obstacles O_{new}. $\mathcal{M}$ records the data of the edges that can be connected, that is, the map. The function `Mapupdate` updates the map by connecting all the vertices in $\mathcal{V}$ and $\mathcal{V}_{new}$. It records the new obstacles encountered by separating and supporting segments in the connection as O_{new}. As for the method of finding the separating and supporting segments, as described above, if a checked line segment is extended a little bit at both ends without entering obstacles, it is a separating or supporting segment. This implementation

ensures that the algorithm can still be used in high-dimensional space. The function `getcorner` is used to extract the vertices of newly encountered obstacles. For 2D cases, extracting vertices is enough for planning. `Astar` is the A* algorithm.

Algorithm 4 Lazy A* with Obstacle Activation

1: $\mathcal{V}$.`Init`(q_{Start}) ▷ Put the q_{Start} into the set of found vertices and start.
2: $\mathcal{V}_{new}$.`Init`(q_{Goal}) ▷ Put the q_{Goal} into the set of newly found vertices and goal.
3: $\mathcal{M} \leftarrow \{\}$ ▷ Initialize the set of all edges.
4: **while** $\mathcal{V}_{new} \neq \{\}$ **do** ▷ Stop looping when there are no newly discovered vertices.
5: $\mathcal{M}, O_{new}, \mathcal{V} \leftarrow$`MapUpdate`($\mathcal{V}_{new}, \mathcal{V}, \mathcal{M}$) ▷ Connect the newly found vertices with all other vertices, update the edge set and found vertices set, and find new obstacles that block the connection.
6: $\mathcal{V}_{new} \leftarrow$`GetCorners`($O_{new}$) ▷ Extract the vertices of newly discovered obstacles. Extracting convex vertices for 2D space.
7: **end while**
8: $\mathcal{P}ath \leftarrow$`Astar`($\mathcal{M}$)
9: **return** $\mathcal{P}ath$

In order to compare the speed of the two algorithms, the basic Lazy A* and Lazy A* with Obstacle Activation were each tested 100 trials in a configuration space with 50 generated square obstacles, as are shown in Figure 3·3. Their respective mean value of running time and path length are shown in Table 3.1. Since both of them are deterministic, their standard deviations are not compared. It can be seen that both algorithms can find the shortest path reliably, and their paths are overlapping. The average running time of Lazy A* with Obstacle Activation is only 18.8% of the average running time of Lazy A* in this case, which supports that Obstacle Activation is an effective method to speed up the process of motion planning and its logic is correct.

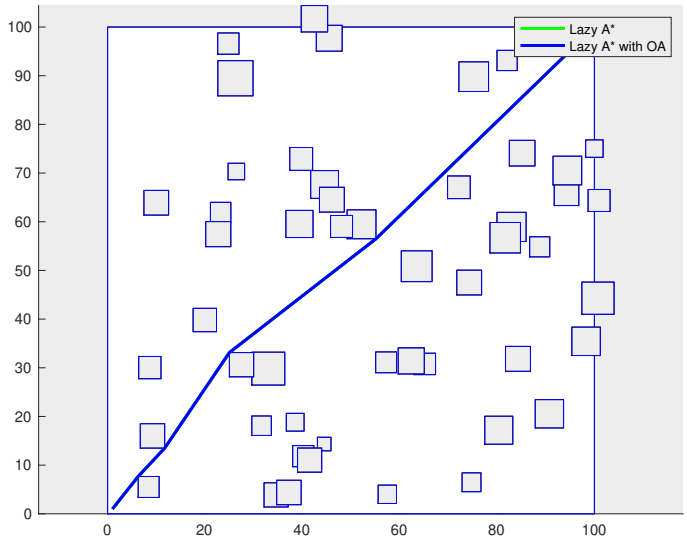


Figure 3·3: Paths generated by the Lazy A* and Lazy A* with Obstacle Activation in a configuration space with 50 square obstacles

Algorithm	Running Time (s)	Path Length (unit)
Lazy A*	15.9027	139.3881
Lazy A* with OA	2.9918	139.3881

Table 3.1: Performance comparison of one-hundred trials between Lazy A* and Lazy A* with Obstacle Activation in a configuration space with 50 square obstacles

3.4 RRT with Obstacle Activation

Compared to geometry-based algorithms, it is possible to explore less space and activate fewer obstacles by introducing randomness. After testing that the Obstacle Activation strategy successfully improves the performance of geometry-based algorithms, the next step is to consider how to combine this idea with sampling-based algorithms. The first attempt is to design an RRT algorithm with Obstacle Activation.

To help the reader in understanding our final algorithm and motivation, we will

introduce it gradually through three versions:

1. RRT with Obstacle Activation (version 1). Each time it selects the nearest node in the graph and the nearest vertex of all the activated obstacles with euclidean distance for a sample in the configuration space. Then it extends the node toward the vertex with a fixed step size.

2. RRT with Obstacle Activation (version 2). The step size is no more fixed when it can attach to the vertex.

3. RRT with Obstacle Activation (version 3). It selects the vertex with the minimum cosine distance.

Algorithm 5 RRT with Obstacle Activation (version 1)

1: $\mathcal{V}$.`Init`(q_{Goal}) ▷ Initialize the set of vertices and goal.
2: $\mathcal{T}$.`Init`(q_{Start}) ▷ Initialize the graph.
3: **for** $i = 1$ to K **do**
4: $q_{rand} \leftarrow$ `Rand`($\mathcal{C}$) ▷ Get a sample in the configuration space.
5: $v_{near} \leftarrow$ `NearestVertex`($q_{rand}, \mathcal{V}$) ▷ Find the nearest vertex of q_{rand}.
6: $q_{near} \leftarrow$ `NearestNode`($q_{rand}, \mathcal{T}$) ▷ Find the nearest node of q_{rand}.
7: $q_{new} \leftarrow$ `Steer` ($v_{near}, \mathcal{T}, \Delta q$) ▷ Move one step from q_{near} to v_{near}.
8: $\mathcal{T}, O_{new} \leftarrow$`Extend`($\mathcal{T}, q_{near}, q_{new}$) ▷ Try to connect the q_{new} and q_{near}. Record the newly encountered obstacles if there is any collision. Update the $\mathcal{T}$.
9: $\mathcal{V}$.`Addcorners`(O_{new}) ▷ Extract vertices of newly encountered obstacles (convex vertices for 2D cases)
10: **end for**

The first version, shown in Algorithm 5, modifies the RRT in the simplest way. On the basis of RRT, an Obstacle Activation module is added, so that it only extends to the q_{Goal} and vertices of the activated obstacle. In order to maintain the bias of searching through the unexplored space, we still choose to uniformly sample a q_{rand} in the configuration space, and choose the q_{near} with the smallest Euclidean distance from q_{rand} as the extended object. Unlike RRT, the steer function will select the

vertex closest to q_{rand} in $\mathcal{V}$ as the target of extension. During the extension process, newly encountered obstacles are continuously recorded in O_{new}, and then their vertices (convex vertices for 2D cases) are added to $\mathcal{V}$.

The new algorithm was tested on the above-mentioned map consisting of 50 square obstacles, and the path generated by the improved algorithm is shown in Figure 3·4. However, the data in Table 3.2 indicate that although the length of the generated path is shorter than the basic RRT, the algorithm obtained by the first improvement is much slower than the basic RRT, which is contrary to expectations. By analyzing the obtained graph $\mathcal{T}$, it is found that because the step size Δq and the vertices of the obstacle are fixed, the position of the newly generated q_{new} is no longer completely random. As a result, the generated $\mathcal{T}$ has a large number of nodes in the same position and repeated edges. The large amount of redundant data caused the speed of the algorithm to drop dramatically. Therefore, it is necessary to prevent the existence of overlapping nodes.

Another problem is that since the step size Δq is fixed, the movement toward the vertices of the obstacle may overextend each step. Imagine the following scenario: If there is no collision between the selected vertices v_{near} and the nearest q_{near}, the next q_{new} will extend to a position farther than v_{near} when the step size Δq is longer than the distance between v_{near} and q_{near}. It is likely to extend into the interior of the obstacle and delay the extension of the path. Moreover, this can also lead to extensions into unrelated obstacles, thereby activating otherwise unwanted obstacles. What is more troublesome is that when there are narrow windows or passages in the map, excessive extension causes the path generation to fail. Therefore, when the step is longer than the distance between v_{near} and q_{new}, v_{near} can be directly used as q_{new}.

We address these concerns in the second version of RRT with Obstacle Activation is shown in Algorithm 6. The algorithm still keeps using a point q_{rand} randomly

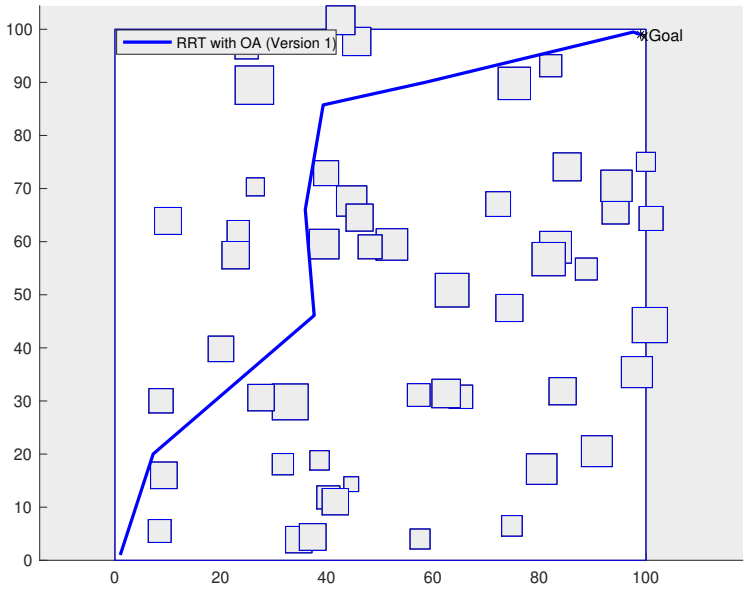


Figure 3·4: The path generated by the RRT with Obstacle Activation (version 1) in a configuration space with 50 square obstacles

sampled from the configuration space to select the nearest q_{near} and v_{near}. When the distance between q_{near} and v_{near} is greater than the incremental distance Δq, q_{new} is still moved by the length of Δq to v_{near}. Unlike the previous version of the algorithm, if q_{new} coincides with an existing node, it will not be extended at this position. When the distance between q_{near} and v_{near} is less than or equal to the incremental distance Δq, directly select v_{near} as the next extended q_{new}. After that, the same strategy is still adopted, trying to extend from q_{near} to q_{new} and update $\mathcal{T}$. If the extension

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	0.1638	0.0761	181.4359	16.8389
RRT with OA (version 1)	0.3837	0.6446	169.9586	23.0249

Table 3.2: Performance comparison of one-hundred trials between RRT and RRT with Obstacle Activation (version 1) in a configuration space with 50 square obstacles

fails, the newly encountered obstacles are recorded, and the vertices of the newly encountered obstacles are added to $\mathcal{V}$.

If v_{near} can be extended, remove this vertex from the vertices set $\mathcal{V}$. This operation can prevent a vertex from being overused, as well as prevent the occurrence of loops. It is an advantage of the RRT to generate graphs without loops, which can avoid the use of algorithms such as A* and save computation time and resources.

Algorithm 6 RRT with Obstacle Activation (version 2)

1: $\mathcal{V}$.`Init`(q_{Goal}) ▷ Initialize the set of vertices and goal.
2: $\mathcal{T}$.`Init`(q_{Start}) ▷ Initialize the graph.
3: **for** $i = 1$ to K **do**
4: $q_{rand} \leftarrow$ `Rand`($\mathcal{C}$) ▷ Get a sample in the configuration space.
5: $q_{near} \leftarrow$ `NearestNode`($q_{rand}, \mathcal{T}$) ▷ Find the nearest node of q_{rand}.
6: $v_{near} \leftarrow$ `NearestVertex`($q_{rand}, \mathcal{V}$) ▷ Find the nearest vertex of q_{rand}.
7: **if** `Distance`(q_{near},v_{near}) $> \Delta q$ **then**
8: $q_{new} \leftarrow$ `Steer` ($v_{near}, \mathcal{T}, \Delta q$) ▷ Move one step from q_{near} to v_{near}.
9: **if** `Ismember`($q_{new}, \mathcal{T}$) **then**
10: **continue** ▷ Avoid overlapping nodes.
11: **end if**
12: **else**
13: $q_{new} \leftarrow v_{near}$ ▷ Assign v_{near} to q_{new}.
14: **end if**
15: $\mathcal{T}, O_{new} \leftarrow$`Extend`($\mathcal{T}, q_{near}, q_{new}$) ▷ Try to connect the q_{new} and q_{near}. Record the newly encountered obstacles if there is any collision. Update the $\mathcal{T}$.
16: **if** `ismember`($q_{Goal}, \mathcal{T}$) **then**
17: **break** ▷ Break when q_{Goal} is in the graph.
18: **end if**
19: $\mathcal{V}$.`Addcorners`(O_{new}) ▷ Extract vertices of newly encountered obstacles (convex vertices for 2D cases)
20: **end for**

The algorithm was tested on the above map of 50 square obstacles. The improved algorithm no longer has the problem of overlapping nodes, and the speed of the algorithm has been greatly improved. The data in Table 3.3 shows that the speed of the algorithm has exceeded RRT, and the generated path length is shorter than RRT. This verifies that the original idea about the Obstacle Activation is correct.

The path it generates is shown in Figure 3·5. It can be seen from the figure that some nodes are directly attached to the vertices of the obstacle. Moreover, the path is very streamlined, which can avoid many obstacles from being activated. It is one of the reasons for the increased speed.

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	0.1638	0.0761	181.4359	16.8389
RRT with OA (version 2)	0.0881	0.0219	147.9340	10.1336

Table 3.3: Performance comparison of one-hundred trials between RRT and RRT with Obstacle Activation (version 2) in a configuration space with 50 square obstacles

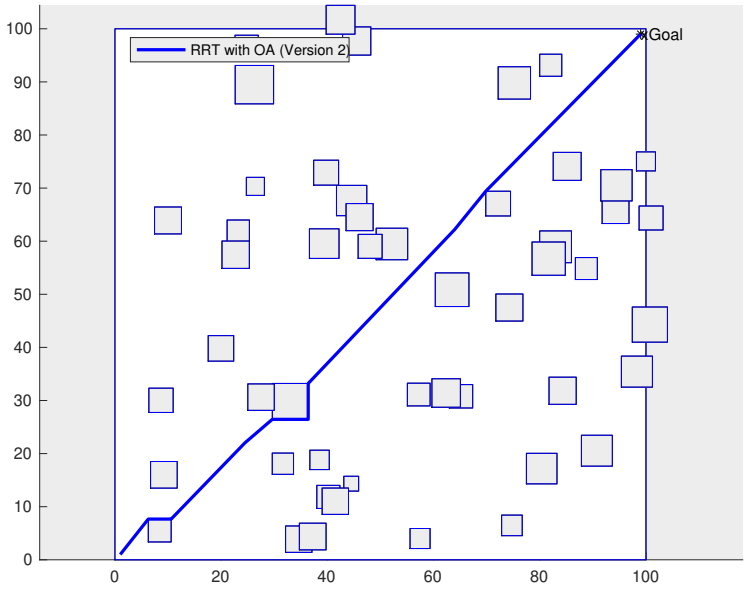


Figure 3·5: The path generated by RRT with Obstacle Activation (version 2) in a configuration space with 50 square obstacles

However, after testing on more maps, new problems appeared. The second version of RRT with Obstacle Activation encounters local minima in the case of a concave polygon, and thus cannot find the path. As is shown in Figure 3·6, when the path

extends to a vertex of the concave polygon, the graph can no longer grow. In order to maintain the bias toward unexplored areas, the sampling methods that RRT and the improved algorithms currently used are to select the nearest vertex or node of the uniformly sampled q_{rand} as the extension target. Considered in a different light, they follow the rules of the Voronoi diagram (Aurenhammer, 1991).

The Voronoi diagram is defined for a set of points, which are called sites. A Voronoi region is the set of points closest to a particular site. The Voronoi diagram is the set of points equidistant from two nearest sites; its edges divide the free space into regions that are closest to different sites (Choset et al., 2005). In the 2D case, the Voronoi diagram is a collection of line segments.

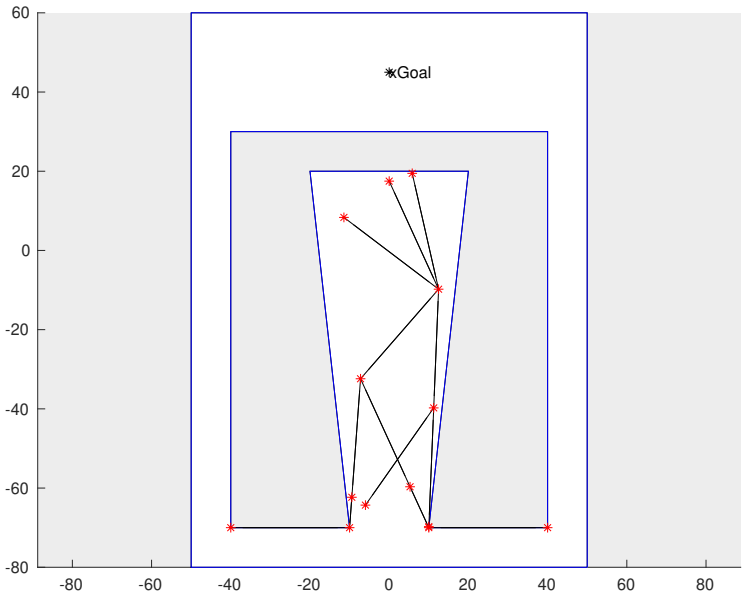


Figure 3·6: Graph generated by the RRT with Obstacle Activation (version 2) in a configuration space with a concave obstacle

Although the Voronoi diagram is not actually used during path planning, it can help us to analyze this problem. The Voronoi diagram (in magenta) of the convex vertices of the obstacles that have been activated in the map and the Voronoi diagram

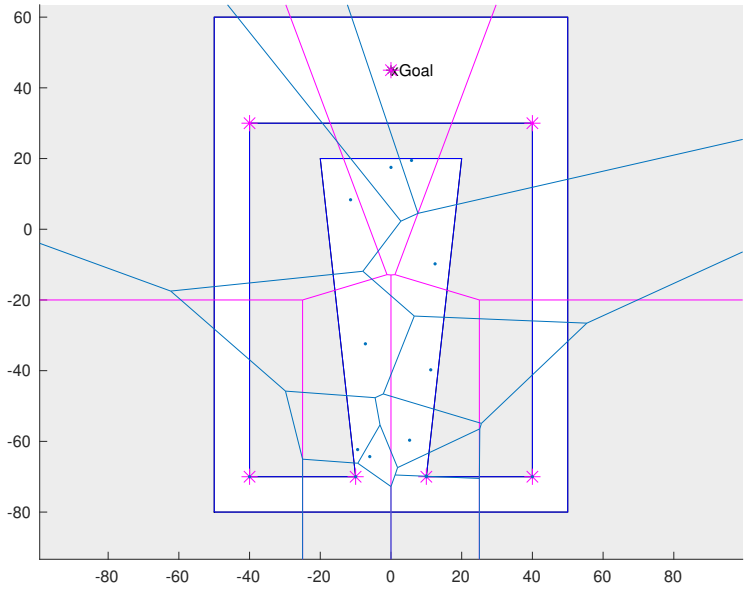


Figure 3·7: Voronoi diagram (in magenta) of the convex vertices of the obstacles that have been activated in the map and the Voronoi diagram (in blue) of the existing nodes

(in blue) of the existing nodes are shown in Figure 3·7. It can be found that the node and the Voronoi region corresponding to its nearest vertex are highly overlapping. In such a case, each randomly generated q_{rand} will always choose the vertex closest to it and the nodes within its Voronoi region. Also, the vertices of the obstacles are fixed and will not change when no new obstacles are found. This algorithm can only continuously try to extend the existing graph to the vertices that have been included in the graph, preventing the graph from growing. To solve this problem, we propose two solutions:

1. From the set of activated vertices, directly delete the vertices that have been included in the graph. When a vertex is deleted, its corresponding Voronoi region will disappear and subsumed by the surrounding vertices that are not included in the graph. In this way, the problem of local minima can be solved. For a

feasible path planning algorithm, deleting vertices can improve its efficiency. However, such an operation causes the loss of ability to rewire the graph and finding an asymptotically optimal path. We will elaborate on the problem in the next section.

2. Do not select the vertex with the minimum Euclidean distance to q_{rand}. A better approach is to select the vertex with the minimum cosine distance from q_{near} as the extension direction. The possibility of all nodes extending toward the vertices of all activated obstacles is preserved. This method not only preserves the bias of outward expansion but also allows the algorithm to avoid local minima. In addition, it also provides the possibility of developing algorithms for finding asymptotically optimal paths. Again, this will be discussed further in the next section.

The cosine distance is derived from the cosine similarity. Cosine similarity is a measure of similarity between two non-zero vectors by their inner-product and magnitude, which represents the cosine of the angle between them (Singhal, 2001). Given two non-zero vectors, $\mathbf{A}$ and $\mathbf{B}$, their cosine similarity, $\cos(\mathbf{A}, \mathbf{B})$, is represented as

$$\cos(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|} = \frac{\sum_{i=1}^{d} A_i B_i}{\sqrt{\sum_{i=1}^{d} A_i^2} \sqrt{\sum_{i=1}^{d} B_i^2}} \tag{3.2}$$

and the corresponding cosine distance of $\mathbf{A}$ and $\mathbf{B}$, $d_c(\mathbf{A}, \mathbf{B})$, is defined as

$$d_c(\mathbf{A}, \mathbf{B}) = 1 - \cos(\mathbf{A}, \mathbf{B}) \tag{3.3}$$

where A_i and B_i are components of vector $\mathbf{A}$ and $\mathbf{B}$ respectively.

In the next algorithm, the new v_{target} will be derived by the following formulas,

$$\mathbf{A} = q_{rand} - q_{near} \tag{3.4}$$

$$\mathbf{B}_j = v_j - q_{near} \tag{3.5}$$

$$d_c(\mathbf{A}, \mathbf{B}_j) = 1 - \cos(\mathbf{A}, \mathbf{B}_j) = 1 - \frac{\mathbf{A} \cdot \mathbf{B}_j}{\|\mathbf{A}\| \cdot \|\mathbf{B}_j\|} \tag{3.6}$$

$$J = \arg\min_j \{d_c(\mathbf{A}, \mathbf{B}_j)\} = \arg\max_j \{\cos(\mathbf{A}, \mathbf{B}_j)\} \tag{3.7}$$

$$v_{target} = v_J \tag{3.8}$$

So far, the obtained information and knowledge are enough to form a new algorithm, which is summarized in Algorithm 7. This is the final version of the algorithm combining RRT and Obstacle Activation. This algorithm can find a feasible path in an environment with known polygon obstacle vertices, and it is also suitable for high-dimensional non-convex environments.

Based on the second version of the algorithm, the method of searching v_{target} was changed from finding the vertex with the minimum Euclidean distance to finding the vertex with the lowest cosine distance (line 6). At the same time, the vertices added to the graph are also deleted because only the feasible path needs to be found (lines 19 to 21).

Performance Testing

In order to test the performance of the algorithm, we applied it on various maps, and RRT was used as a comparison.

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	22.6551	32.0013	196.6483	36.7800
RRT with OA (version 3)	0.0132	0.0068	121.1472	0.0999

Table 3.4: Performance comparison of one-hundred trials between RRT and RRT with Obstacle Activation (version 3) in a configuration space with a narrow Passage (width = 0.2 unit)

Figure 3·8 shows an environment with a long narrow passage. Because RRT with

Algorithm 7 RRT with Obstacle Activation (version 3)

$\mathcal{V}$.Init(q_{Goal}) ▷ Initialize the set of vertices and goal.
$\mathcal{T}$.Init(q_{Start}) ▷ Initialize the graph.
for $i = 1$ to K **do**
    $q_{rand} \leftarrow$ Rand($\mathcal{C}$) ▷ Get a sample (q_{rand}) in the configuration space.
    $q_{near} \leftarrow$ NearestNode($q_{rand}, \mathcal{T}$) ▷ Find the nearest node(q_{near}) of q_{rand}.
    $v_{target} \leftarrow$ Argmincosdis($q_{near}, q_{rand}, \mathcal{V}$) ▷ Find the vertex with the lowest cosine distance.
    if Distance(q_{near},v_{target}) $> \Delta q$ **then**
        $q_{new} \leftarrow$ Steer ($v_{target}, \mathcal{T}, \Delta q$) ▷ Move one step from q_{near} to v_{target}.
        if Ismember($q_{new}, \mathcal{T}$) **then**
            continue ▷ Avoid overlapping nodes.
        end if
    else
        $q_{new} \leftarrow v_{target}$ ▷ Assign v_{target} to q_{new}.
    end if
    $\mathcal{T}, O_{new} \leftarrow$Extend($\mathcal{T}, q_{near}, q_{new}$) ▷ Try to connect the q_{new} and q_{near}. Record the newly encountered obstacles if there is any collision. Update the $\mathcal{T}$.
    if ismember($q_{Goal}, \mathcal{T}$) **then**
        break ▷ Break when q_{Goal} is in the graph.
    end if
    if ismember($v_{target}, \mathcal{V}$) **then**
        $\mathcal{V}$.Remove(v_{target}) ▷ Remove v_{target} from $\mathcal{V}$ if v_{target} can be added into $\mathcal{T}$.
    end if
    $\mathcal{V}$.Addcorners(O_{new}) ▷ Extract vertices of newly encountered obstacles (convex vertices for 2D cases)
end for

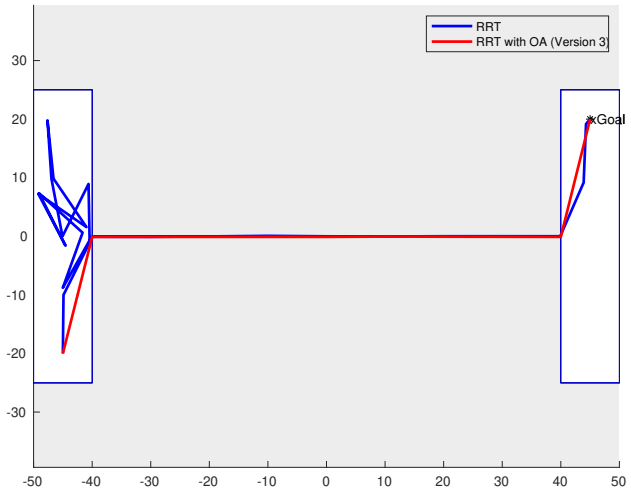


Figure 3·8: Paths generated by RRT and RRT with Obstacle Activation (version 3) in a configuration space with a narrow Passage (width = 0.2 unit)

Obstacle Activation can directly find the corners at the entrances of the passage as target corners, it can find a path through this narrow passage very quickly (shown in 3.4), and the found path is almost the optimal solution which length is the shortest. This effective method was mentioned in the introduction of PRM Trees.

As for the maze shown in Figure 3·9, although there are only three obstacles in total, they occupy almost the entire map and are activated in subsequent explorations. In this case, RRT with Obstacle Activation has lost its advantage of extensions to only a part of obstacles. As can be seen from Table 3.5, its performance is just slightly better than RRT's.

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	5.3395	1.6680	888.9713	26.0839
RRT with OA (version 3)	5.0775	1.2941	863.5558	24.8530

Table 3.5: Performance comparison of one-hundred trials between RRT and RRT with Obstacle Activation (version 3) in a maze

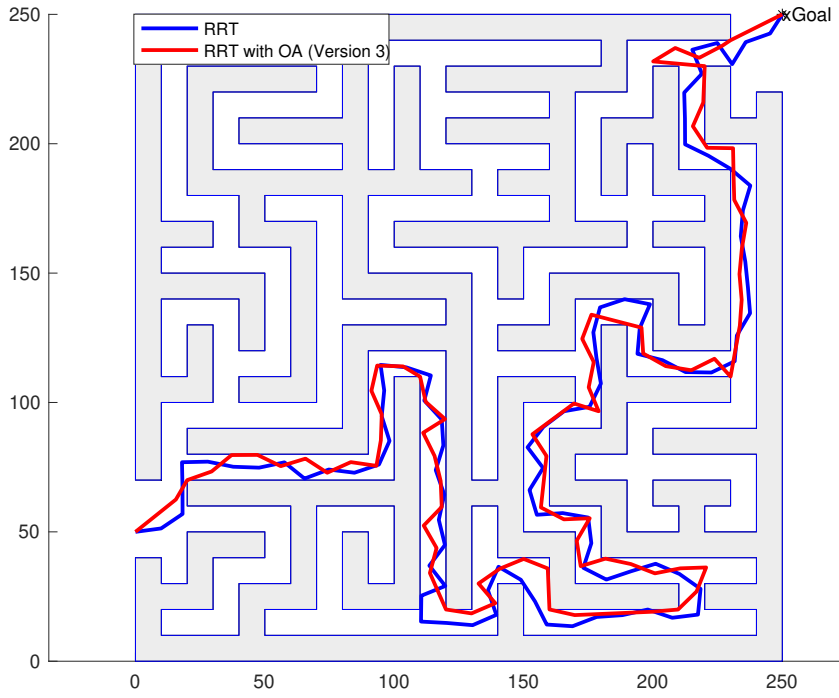


Figure 3·9: Paths generated by RRT and RRT with Obstacle Activation (version 3) in a maze

In addition, the incremental distance Δq of RRT with Obstacle Activation also has a huge impact on its speed. The paths generated by the different Δq in this environment with 50 square obstacles are shown in Figure 3·10. Different incremental distances are used for the comparison. The mean value and the standard deviation of the running time and the path length are shown in Table 3.6.

The same test is performed in a complex environment with a composition of several concave obstacles, as is shown in Figure 3·11. Obstacles in this environment are all activated. The average running time is shown in Table 3.7. The algorithm does not have the problem of falling into local minima any more.

What is interesting is that no matter what the environment and the percentage of obstacles that are activated, the larger the step size, the shorter the average running

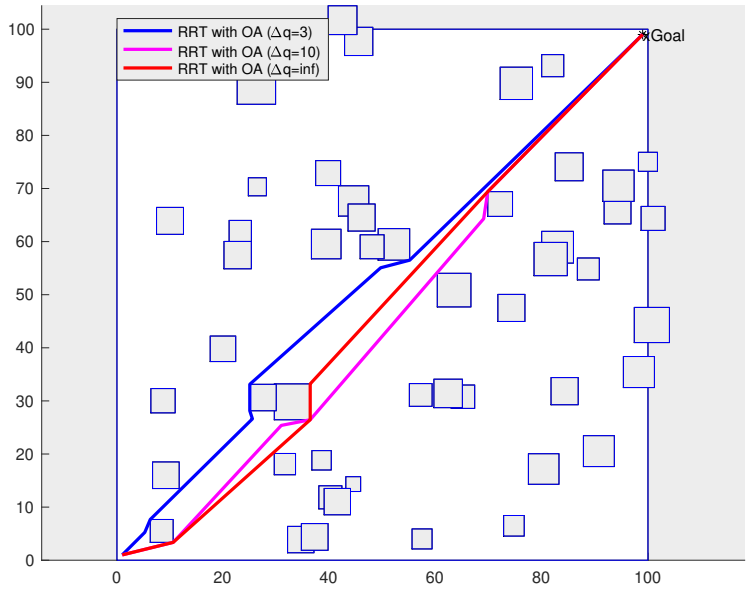


Figure 3·10: Paths generated by RRT with Obstacle Activation (version 3) with different Δq in a configuration space with 50 square obstacles

Algorithm	Δq	Running Time (s)		Path Length (unit)	
		Mean	SD	Mean	SD
RRT	10	0.1638	0.0761	181.4359	16.8389
RRT with OA (version 3)	3	0.2513	0.0594	145.0572	3.1417
	10	0.1016	0.0376	153.5393	16.2686
	15	0.0693	0.0291	155.0420	20.6101
	$+\infty$	0.0382	0.0217	148.8421	11.9095

Table 3.6: Performance comparison of one-hundred trials between RRT and RRT with Obstacle Activation (version 3) in a configuration space with 50 square obstacles

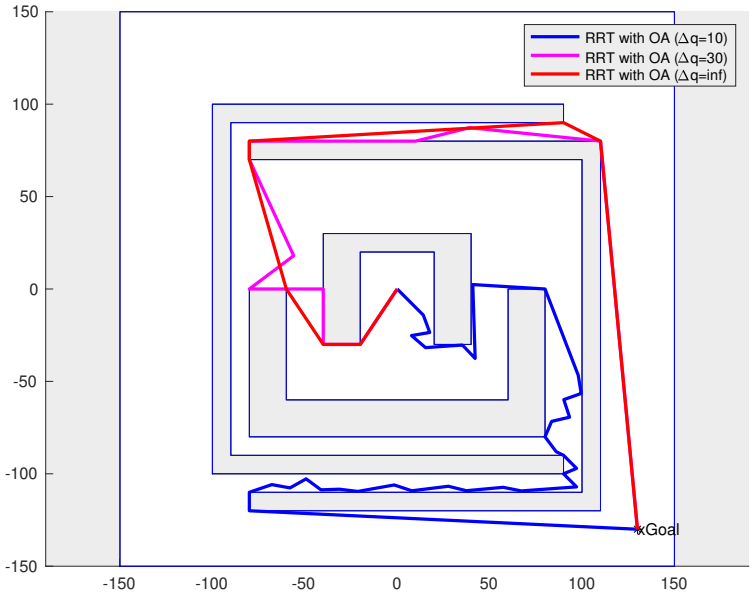


Figure 3·11: Paths generated by RRT with Obstacle Activation (version 3) with different Δq in a configuration space with multiple concave polygonal obstacles

		Running Time (s)		Path Length (unit)	
Algorithm	Δq	Mean	SD	Mean	SD
RRT	30	6.3067	4.2347	997.7250	240.7269
RRT with OA (version 3)	10	3.6596	0.8093	715.6147	57.6539
	30	1.2588	0.6682	742.8147	78.7765
	50	0.6791	0.3971	746.1234	144.5360
	$+\infty$	0.2904	0.1506	671.3924	138.1695

Table 3.7: Performance comparison of one-hundred trials between RRT and RRT with Obstacle Activation (version 3) in a configuration space with multiple concave polygonal obstacles

time. When the step length is longer, it also means that more line segments are allowed to be directly attached to the obstacles; for this reason, the efficiency of the algorithm will be higher. When the incremental distance far exceeds the distance between the vertices of the obstacles, all the line segments are connected to the vertices of the obstacles, and the speed of the algorithm is the fastest. Based on the above results, an infinitely long incremental distance is always worth choosing.

It is also worth noting that the line segment collision detector used in this thesis is parallelized. The line segments are first uniformly discretized, and then the points which are in collision with obstacles are detected simultaneously by means of parallel calculation. Therefore, regardless of the length of the line segment, it has little effect on the execution time. This also means that longer line segments have more points for the check of collision at the same time, which makes use of the collision checker more efficient. As a final note, The line segment detector uses the Matlab function `inpolygon` (MATLAB, 2018), which is based on the algorithm introduced in (Hormann and Agathos, 2001).

3.5 RRT* with Obstacle Activation

After proposing a way to accelerate feasible-path sampling-based algorithms, our next task is to rewire the graph to find the shortest path to achieve functionality similar to RRT*. This section introduces the method RRT* with Obstacle Activation. In the previous section, vertices cannot be deleted from the set of activated vertices $\mathcal{V}$, if we want to maintain the ability to rewire the graph and asymptotically obtain the optimal path.

Additionally, since the area of the Voronoi region occupied by each node attached to the vertex is different, selecting the node with the shortest Euclidean distance from the sampled q_{rand} will make each node have a different probability of being

selected. The probability of some nodes being selected may be much greater than the probability of others being selected. For algorithms that may need to check all nodes to find the shortest path, maintaining that kind of bias is unnecessary. As a result, the sampling method of the nodes and vertices is determined to be randomly selected from the two sets $\mathcal{V}$ and $\mathcal{T}$. At the same time, it is also necessary to avoid repeatedly checking the connectivity of the same pair of nodes.

Therefore, required is a data structure to store whether there are collisions between the nodes and the vertices of activated obstacles. Here, a data structure similar to the adjacency matrix is used, and it is named connection information matrix $\mathcal{I}$. With respect to a standard adjacency matrix, the connection information matrix needs to be dynamically updated, and each element in the matrix has three states instead of two, namely:

$$\begin{aligned} &\textit{null}\text{: no information} \\ &1\text{: collision-free} \\ &0\text{: collision.} \end{aligned} \tag{3.9}$$

Its rows correspond to all nodes in $\mathcal{T}$, and its columns correspond to all vertices in $\mathcal{V}$ and goal. Each element $\mathcal{I}_{(i,j)}$ contains the attributes of the edge between node i and vertex j. Since $\mathcal{T}$ and $\mathcal{V}$ are dynamically expanded, the number of rows and columns of the matrix also changes accordingly. Each time $\mathcal{I}$ is updated, we need to consider that a pair of points may correspond to two positions of the matrix. Both of them need to be found and updated at the same time. In order to avoid redundancy caused by self-loop, when a vertex is added to the set of nodes, the value corresponding to the same point on the row and column in $\mathcal{I}$ is set to 0.

The pseudocode of Algorithm 8 summarizes this algorithm. Firstly use q_{Goal} to initialize $\mathcal{T}$, the set of vertices and goal. Use q_{Start} to initialize graph, $\mathcal{V}$. Then set

Algorithm 8 RRT* with Obstacle Activation

1: $\mathcal{V}$.Init(q_{Goal}) ▷ Initialize the set of vertices and goal.
2: $\mathcal{T}$.Init(q_{Start}) ▷ Initialize the graph.
3: $\mathcal{I}(q_{selected}, v_{target}) = null$ ▷ Initialize the connection information matrix.
4: **for** $i = 1$ to K **do**
5: Randomly select a pair of node $q_{selected}$ and vertex v_{target} from $\mathcal{I}$. These two points satisfy that $\mathcal{I}(q_{selected}, v_{target}) \neq 0$.
6: **if** $\neg$ ismember($v_{target}, \mathcal{T}$) **then**
7: Try to connect the $q_{selected}$ and v_{target}.
8: Record the newly encountered obstacles if there is any collision.
9: Update $\mathcal{T}$ and $\mathcal{I}$.
10: **else**
11: **if** $\mathcal{I}(q_{selected}, v_{target}) = null$ **then**
12: Try to connect the $q_{selected}$ and v_{target}.
13: Record the newly encountered obstacles O_{new} if there is any collision.
14: Update $\mathcal{I}$.
15: **end if**
16: **if** $\mathcal{I}(q_{selected}, v_{target})$ =1 **then**
17: **if** Distance($q_{selected}$,v_{target})+Cost($q_{selected}$) <Cost(v_{target}) **then**
18: Rewire $\mathcal{T}$.
19: **else**
20: **continue**
21: **end if**
22: **end if**
23: **end if**
24: Add vertices of newly encountered obstacles O_{new} into $\mathcal{V}$ (Add convex vertices for 2D cases).
25: Expand and update the $\mathcal{I}$ since the new vertices are added into $\mathcal{V}$.
26: **if** $\neg$ ismember($q_{Goal}, \mathcal{T}$) **then**
27: Feasible Path is found.
28: **end if**
29: **end for**

their corresponding values in the connection information matrix $\mathcal{I}$ to *null*. Now, the algorithm is initialized. The following operations will be run K times.

First, randomly select a pair of node $q_{selected}$ and vertex v_{target} from $\mathcal{I}$, and these two points satisfies that $\mathcal{I}(q_{selected}, v_{target}) \neq 0$. Also avoid the situation where the backpointer of v_{target} is exactly $q_{selected}$. If this v_{target} has not been recorded in $\mathcal{T}$, then try to connect v_{target} and $q_{selected}$ and record the new obstacles encountered. Record the result of this connection in $\mathcal{I}$. If successfully connected, record this v_{target} into $\mathcal{T}$ and $\mathcal{I}$. If this v_{target} is already in $\mathcal{T}$ and if $\mathcal{I}(q_{selected}, v_{target}) = null$, then try to connect v_{target} and $q_{selected}$ and record the new obstacles encountered. Record the result of this connection in $\mathcal{I}$. If successfully connected, update $\mathcal{I}$ for this v_{target}. Then if $\mathcal{I}(q_{selected}, v_{target})$ =1, check if the new connection can make the distance from v_{target} to the starting point shorter. If yes, rewire $\mathcal{T}$, if not, continue. Finally, the vertices of the new obstacles encountered are added to $\mathcal{V}$, and $\mathcal{I}$ is extended to record these vertices. When q_{Goal} is added to $\mathcal{T}$, a feasible path has been found.

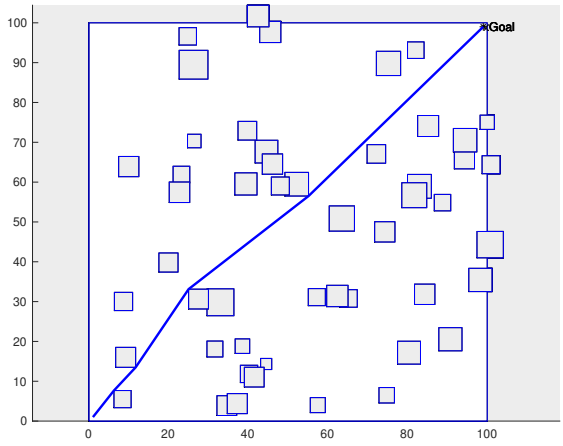


Figure 3·12: The shortest path in a configuration space with 50 square obstacles

To test the performance of this algorithm, compare it to Lazy A* and Lazy A* with Obstacle Activation. Measure the time it takes for each of them to find the shortest

Performance Comparison of Running One-Hundred Trials			
	Lazy A*	Lazy A* with OA	RRT* with OA
Running Time (s)	15.9027	2.9918	2.0170
Path Length (unit)	139.3881	139.3881	139.3881

Table 3.8: Performance comparison of Lazy A*, Lazy A* with Obstacle Activation and RRT* with Obstacle Activation for shortest path planning in a configuration space with 50 square obstacles

Performance Comparison of Running One-Hundred Trials			
	Lazy A*	Lazy A* with OA	RRT* with OA
Running Time (s)	0.2245	0.2495	0.2543
Path Length (unit)	574.7090	574.7090	574.7090

Table 3.9: Performance comparison of Lazy A*, Lazy A* with Obstacle Activation and RRT* with Obstacle Activation for shortest path planning in a configuration space with multiple concave polygonal obstacles

Performance Comparison of Running One-Hundred Trials			
	RRT	RRT with OA	RRT* with OA
Δq	10	$+\infty$	None
Running Time (s)	0.1638	0.0382	0.2407
Path Length (unit)	181.4359	148.8421	158.1886

Table 3.10: Performance comparison of RRT, RRT with Obstacle Activation and RRT* with Obstacle Activation for feasible path planning in a configuration space with 50 square obstacles

Performance Comparison of Running One-Hundred Trials			
	RRT	RRT with OA	RRT* with OA
Δq	30	$+\infty$	None
Running Time (s)	6.3067	0.2904	0.1901
Path Length (unit)	997.7250	671.3924	623.0441

Table 3.11: Performance comparison of RRT, RRT with Obstacle Activation and RRT* with Obstacle Activation for feasible path planning in a configuration space with multiple concave polygonal obstacles

path. For RRT* with Obstacle Activation, record the running time when the path length no longer changes, which means that the shortest path is found. From Table

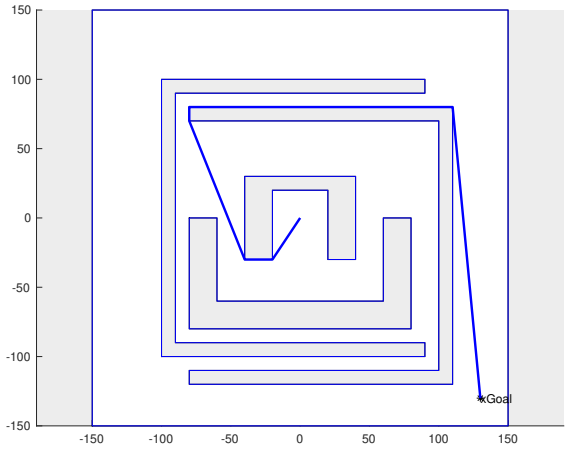


Figure 3·13: The shortest path in a configuration space with multiple concave polygonal obstacles

3.8, we observe that in a relatively sparse environment with 50 obstacles in Figure 3·12, RRT* with Obstacle Activation is faster than the other two algorithms. Because it neither activates all obstacles nor traverses all vertices of all activated obstacles, the completely random approach makes it the fastest. However, in an environment with four concave obstacles shown in Figure 3·13, it is indeed the slowest, as recorded in Table 3.9. In an environment where all obstacles are activated, the rewiring of RRT* with Obstacle Activation makes it necessary to repeatedly check whether some edges can be rewired for a short distance. For the other two algorithms, these edges need only be checked once.

We also compare RRT* with Obstacle Activation to RRT and RRT with Obstacle Activation, and compare the average running time of each of them to generate a feasible path. To make the comparison fair, we stop RRT* with Obstacle Activation as soon as a feasible path is found. Table 3.10 shows that in the sparse environment with many obstacles of Figure 3·12, RRT* with Obstacle Activation takes the longest time. This is because it attempts to rewire repeatedly before finding a feasible path, thereby delaying time. The reason why RRT with Obstacle Activation is the fastest

among the three is that under such sparse and no concave obstacles, the goal and obstacles between the goal are more likely to be its extension target. The selected node is also affected by its bias, and it is easier to go to the unexplored area. The strategy of uniformly sampling a pair of connectable nodes makes RRT* with Obstacle Activation no longer has a tendency to move to the unexplored space, which seems to be a disadvantage. Nevertheless, without such a bias is advantageous in an environment with concave obstacles. Table 3.11 shows that in the environment with four concave obstacles, as in Figure 3·13, RRT* with Obstacle Activation becomes the algorithm with the lowest average running time and the shortest average path.

From the above results and analysis, whether it is to obtain the shortest path or a feasible path, the overall performance of this algorithm is excellent. A limitation of all the algorithms described above is that they are still based on polygonal obstacles, and it is assumed that they can directly obtain the vertices of the obstacles. When we lose this information, the advantages no longer exist. We offer a solution to this situation in the next chapter.

Chapter 4

Obstacle Exploration

4.1 Obstacle Exploration

In order to generalize the idea of Obstacle Activation to more environments where the vertices of obstacles cannot be obtained directly, we can extract the vertices of obstacles through a variety of other algorithms.

This approach has been used for a long time. For example, OBPRM (Amato et al., 1998) emits rays evenly from a point inside obstacles and obtains collision-free points around the obstacle through a binary search. Bridge Test (Hsu et al., 2003) finds the points in the narrow passage through the way that the ends of a line segment are inside the obstacle, and the midpoint is outside the obstacle. Additionally, UOBPRM (Yeh et al., 2012) finds nodes by looking for discontinuities inside and outside obstacles within a line segment. They obtain vertices in advance in the entire configuration space or around all obstacles. The path is then obtained using a sampling-based algorithm. However, the search scope involves the entire configuration space because these methods cannot predict which obstacles need to be activated. As discussed in Chapter 3, obstacles that block the connection of points that exist in the graph are obstacles that need to be bypassed. Only the points around these obstacles are useful. However, in the absence of this information, all obstacles have to be solved, which not only increases the amount of calculation to get the vertices but also brings redundancy into the construction of the roadmap. So it is necessary to perform the path planning and to generate the vertices online.

Therefore, we need to solve the problem of finding the information of the "activated obstacles" without knowing the shape of the obstacles itself. In this chapter, we propose a solution based on the collision points. When a collision point is found, it must be inside the boundary of its corresponding obstacle. If a tree is used to extend based on this collision point, then all its child nodes will inherit the information of this obstacle. We arrive at the desired solution by recording the collision points on the segments during extensions of the graph, learning the shape of the obstacles by using RRT and generating points without collisions around the obstacles. We call, collision-free points generated in this way around the obstacles are called Artificial Vertices. The process of exploring obstacles using the structure of trees is called Obstacle Exploration.

In fact, there are a series of Retraction-Based RRT (Pan et al., 2010; Liangjun Zhang and Manocha, 2008) that explore the entire configuration space first, and then utilize obstacles to construct a deformed Voronoi diagram to find feasible paths. However, the exploration of the entire space is still a disadvantage. Their search scope is not limited to obstacles that need to be explored. These methods, therefore, suffer from decreased efficiency.

4.2 PRM Trees with Obstacle Exploration

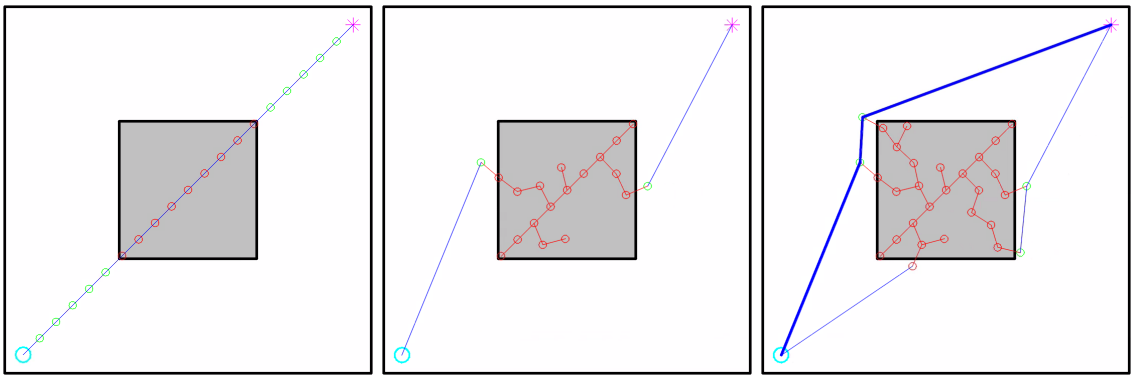
In this section, we focus on the algorithm that can be used to queue these artificial vertices into the graph.

In comparison with using the vertices of polygonal obstacles directly, the position of the points generated by Obstacle Exploration is no longer deterministic. It is impossible to know when the nodes necessary for the path have been generated. The problem then becomes choosing the algorithm to best deal with the artificial vertices that are generated in real-time.

If we were to utilize Obstacle Exploration with PRM, since all the samples appear around the obstacle, when the radius of the local planner is too small, the points around the obstacles that are far away would not be connected. If the radius is too large, the amount of calculation will increase sharply. So PRM is not a good choice. If we choose RRT, it is necessary to answer a question: What weight should be arranged to decide whether the next step is to extend the RRT inside the obstacle or the graph outside the obstacle? To avoid these problems, we propose to combine Obstacle Exploration with PRM Trees. Compared with RRT and PRM, it has the following two advantages:

1. Its local planner searches in the entire configuration space, so we will avoid problems like not being able to determine the radius of the local planner.

2. The connection of each newly generated artificial vertex will take into account all the artificial vertices that have been generated; therefore, the problem that RRT cannot determine the weight is avoided.



(a) Connection and collision detection (b) Explore obstacles and connect free points (c) Feasible path extraction

Figure 4·1: Graph construction of the composite sampling-based system

The pseudocode for our proposed algorithm combining PRM Trees with Obstacle Exploration is summarized by Algorithm 9. It is initialized by connecting the start

Algorithm 9 PRM Trees with Obstacle Exploration for sparse environments

1: **function** PRM_TREES_WITH_OE($q_{Start}, q_{Goal}, K, \Delta c$)
2: $q_{Start}.Tree \leftarrow 1$
3: $q_{Goal}.Tree \leftarrow 2$
4: $Trees$.Init(q_{Start}, q_{Goal})
5: $Collisions \leftarrow$ CONNECT(q_{Start}, q_{Goal}) ▷ Record collisions between start and goal.
6: **while** $q_{Start}.Tree \neq q_{Goal}.Tree$ **do**
7: $q_{new}, Collisions \leftarrow$ OERRT($Collisions, \Delta c$) ▷ Get a new free points generated by Obstacle Exploration.
8: $NumberOfTrees \leftarrow$COUNTTREES($Trees$)
9: $N \leftarrow$MIN($K, NumberOfTrees$)
10: **for** $i = 1$ to N **do**
11: $Trees, collsions \leftarrow$ MERGE($Trees, q_{rand}$) ▷ Try to merge trees by the q_{new}. Record the collisions.
12: **end for**
13: **end while**
14: $Path =$ FIND_PATH($Trees$,q_{new})
15: **return** $Path$
16: **end function**

Algorithm 10 Obstacle Exploration for sparse environments

1: **function** OERRT($Collisions, \Delta c$)
2: **while** 1 **do**
3: $c_{rand} \leftarrow$ Rand($\mathcal{C}$) ▷ Get a sample in the configuration space.
4: $c_{near} \leftarrow$ Nearestnode($q_{rand}, Collisions$) ▷ Find the nearest node of c_{rand}.
5: $c_{new} \leftarrow$ Steer ($c_{near}, Collisions, \Delta c$) ▷ Move Δc from c_{near} to c_{rand}.
6: **if** IsCollision(c_{new}) **then**
7: $Collisions \leftarrow$ Addnode($c_{new}, Collisions.$) ▷ If c_{new} collides, add it to $Collisions$.
8: **else**
9: $q_{free} \leftarrow c_{new}$ ▷ If c_{new} is free, assign it to q_{new}.
10: **return** ($q_{new}, Collisions$) ▷ Return q_{free} and the updated $Collisions$.
11: **end if**
12: **end while**
13: **end function**

and the goal, recording the collision points encountered by the local planner and then using `OERRT` to explore obstacles based on these points. The main difference from the PRM Trees shown in the previous chapter is that the newly generated collision-free sample becomes the artificial vertex generated by the RRT inside the obstacles. Every time an artificial vertex is obtained, it tries to connect with the nearest points in N trees and to merge all connected trees. All collisions are recorded. If it is an isolated point, then a new tree is established. A sketch of the algorithmic process is shown in Figure 4·1.

Algorithm 10 is the pseudo-code of Obstacle Exploration RRT in a sparse environment. In order to distinguish the nodes inside the obstacle from the nodes outside the obstacle, we use c to represent the nodes used by Obstacle Exploration RRT. The set of all the collision points are recorded in $Collisions$. At a high level, the operations performed are as follows: randomly sample a point c_{rand} in the configuration space each time, select the nearest collision point c_{near} and, move the incremental distance Δc to get a new node c_{new}. If the new node c_{new} is a collision point, record it, and repeat. If the new node c_{new} goes out of the obstacle, return it with the updated $Collisions$. Since it is assumed that the distance between obstacles is larger than the incremental distance Δc in a sparse environment, it is not necessary to check the node-to-node connection.

Figures 4·2 and 4·3 show the paths, graphs and collision points generated by the algorithm in different configuration spaces. Table 4.1 and Table 4.2 show the performance of different algorithms in the corresponding environments.

From Table 4.1 and Figure 4·4, we can find that PRM Trees with Obstacle Exploration is much faster than RRT, faster than PRM Trees, and slower than RRT with Obstacle Activation in a configuration space with multiple concave polygonal obstacles as shown in Figures 4·2. The latter is due to the fact that PRM Trees with

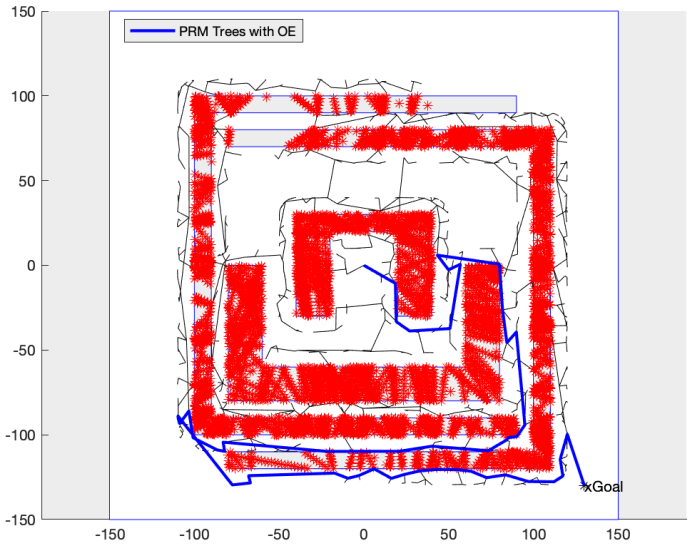


Figure 4·2: Path, graph and collision points generated by PRM Trees with Obstacle Exploration in a configuration space with multiple concave polygonal obstacles

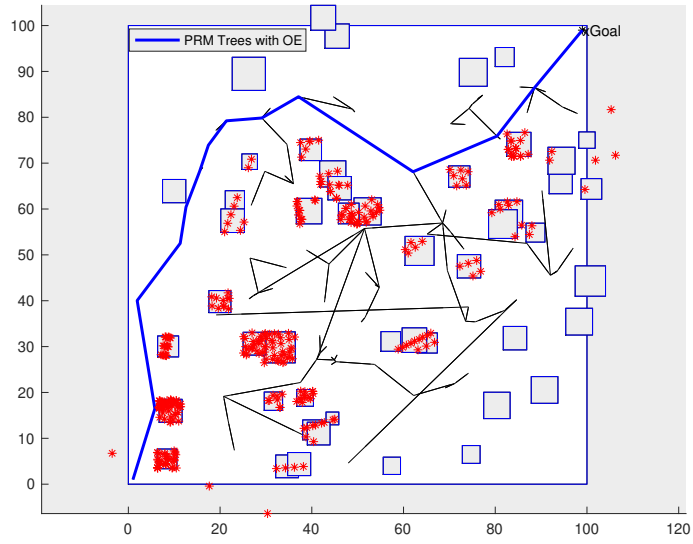


Figure 4·3: Path, graph and collision points generated by PRM Trees with Obstacle Exploration in a configuration space with 50 square obstacles

Obstacle Exploration lacks the advantage of using obstacle vertices directly. Hence, it is slower than RRT with Obstacle Activation. We can find that the new algorithm is faster than RRT and PRM Trees, which shows that although Obstacle Exploration increases the computation of obtaining each step extension, this penalty is not enough to reduce the overall speed of the algorithm. As previously mentioned, what we need to find is the balance between the cost of each step and the global cost of the entire algorithm. The Obstacle Exploration can bring efficiency improvement in more complex environments.

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	6.3067	4.2347	997.7250	240.7269
RRT with OA (version 3)	0.2904	0.1506	671.3924	138.1695
PRM Trees (K=2)	1.8551	0.8225	1066.3854	303.2903
PRM Trees with OE (K=2)	1.1563	0.4764	835.3611	120.8856

Table 4.1: Performance comparison of RRT, RRT with Obstacle Activation, PRM Trees and PRM Trees with Obstacle Exploration in a configuration space with multiple concave polygonal obstacles (one-hundred trials)

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	0.1638	0.0761	181.4359	16.8389
RRT with OA	0.0382	0.0217	148.8421	11.9095
PRM Trees (K=2)	0.0724	0.0451	231.6787	55.0465
PRM Trees with OE (K=2)	0.0903	0.0603	188.3978	27.6413

Table 4.2: Performance comparison of RRT, RRT with Obstacle Activation, PRM Trees and PRM Trees with Obstacle Exploration in a configuration space with 50 square obstacles (one-hundred trials)

From Table 4.2 and Figure 4·5, it can be seen that PRM Trees with Obstacle Exploration is not as fast as PRM Trees when passing a relatively sparse environment as shown in Figure 4·3, which is not surprising. After all, the sparser the environment, the simpler the algorithm can be. In such an excessively sparse environment, the

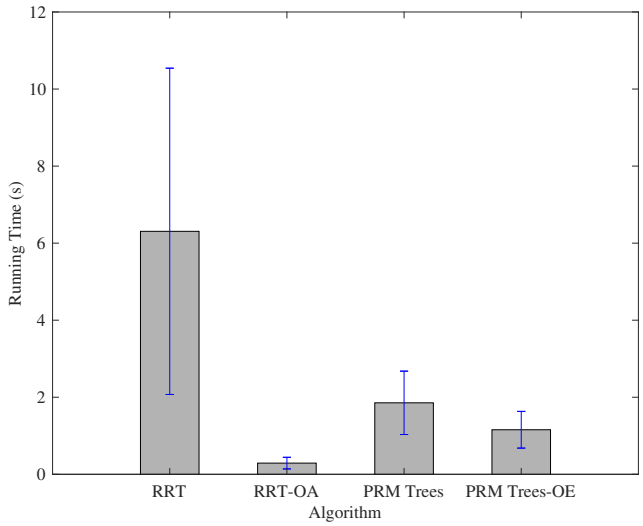


Figure 4·4: Average running time of RRT, RRT with Obstacle Activation, PRM Trees and PRM Trees with Obstacle Exploration in a configuration space with multiple concave polygonal obstacles

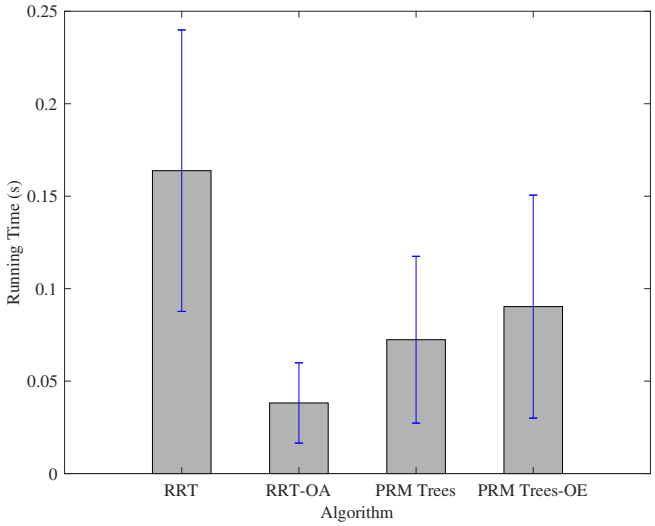


Figure 4·5: Average running time of RRT, RRT with Obstacle Activation, PRM Trees and PRM Trees with Obstacle Exploration in a configuration space with 50 square obstacles

speed of direct sampling will be faster than the speed of finding free points around obstacles. However, Obstacle Exploration reduces the average path length, which is still beneficial.

Although this chapter shows that Obstacle Exploration can bring advantages to sampling-based algorithms, the current Obstacle Exploration still has disadvantages. We can also see the exaggerated number of collisions in Figure 4·2. Since the dense collision detector used by the local planner of PRM Trees can generate too many collision points at a time, the collisions left in the obstacle will increase sharply, incurring into high memory usage. Furthermore, when dealing with complex environments in which dimensions of the space in different directions vary greatly, such as through narrow passages, the currently used Obstacle Exploration methods cannot effectively handle them. These and other shortcomings of the Obstacle Exploration method, and possible solutions, are discussed in detail in the next section.

4.3 Obstacle Exploration for complex environments

It is worth noting that for the Obstacle Exploration process, it is still necessary to use a local planner with dense collision detection because dense collision detection can effectively handle complex situations such as narrow passages. However, if only the endpoints of the Obstacle Exploration local planner are retained and the increment distance is long, it is possible to enter from one obstacle to another. As a result, obstacles that do not need to be explored may be solved, and points without collisions between obstacles may be missed, which is undesirable.

In order to avoid the above problems, when the Obstacle Exploration RRT is extended beyond the obstacle, the following two candidates of artificial vertices output are practical.

1. The first free point obtained by the dense collision detector of the local planner.

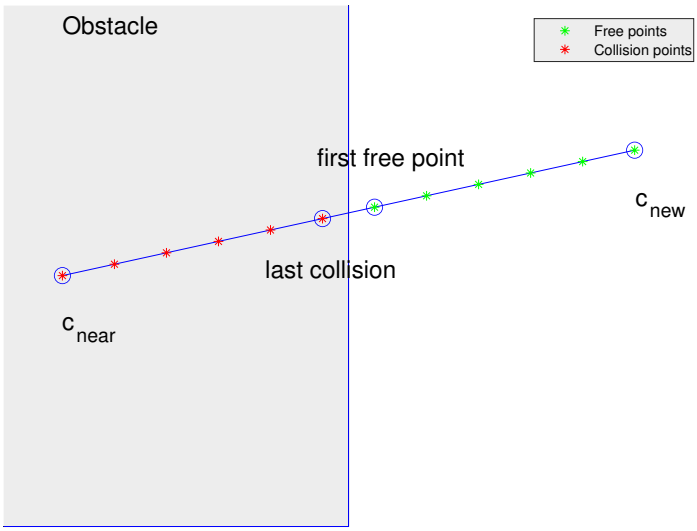


Figure 4·6: Dense collision detector of the local planner of Obstacle Exploration

As shown in Figure 4·6. The advantage of the first free point is the ability to detect narrow passage because the free point near the obstacle is more likely to appear at the entrance of the narrow channel. The disadvantage is that the connection speed in sparse environments is reduced because the around area of points near an obstacle is occupied more by the obstacle. Figure 4·7 shows all the collision points (in red) and all the first free points (in green) recorded by Obstacle Exploration after 1500 iterations.

2. Randomize one of the free points obtained by the dense collision detector of the local planner. The artificial vertices thus obtained will be distributed within a certain range around the obstacle, as shown in Figure 4·8. The advantage of random free points is that the probability of each point being connected is increased, thereby accelerating the path generation in sparse environments.

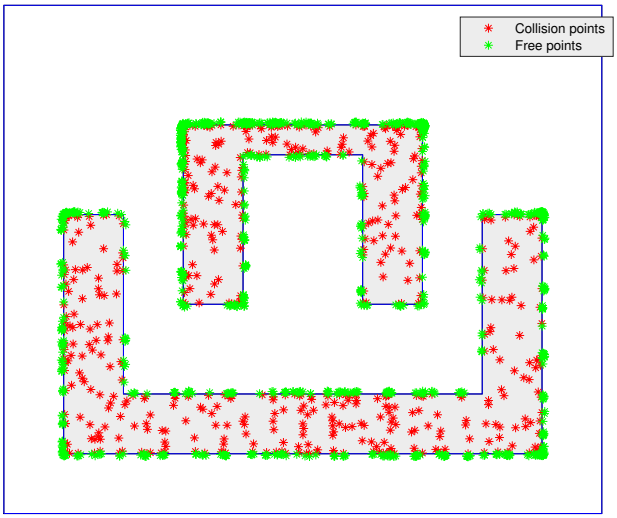


Figure 4·7: Collision points and first free points recorded by Obstacle Exploration after 1500 iterations

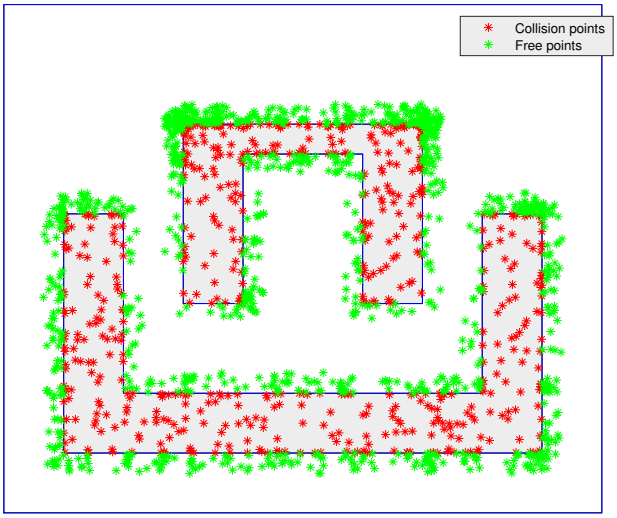


Figure 4·8: Collision points and random free points recorded by Obstacle Exploration after 1500 iterations

Another improvement is that in order to increase the efficiency of exploring obstacles, when the local planner generates a free point, the upstream collision point of the first free point is selected as the next node, as shown in Figure 4·6. This selection can increase the speed through narrow spaces. It also avoids entering other obstacles. The reason for not keeping all collision points is to reduce memory consumption, improving performance when the configuration space is large and complex.

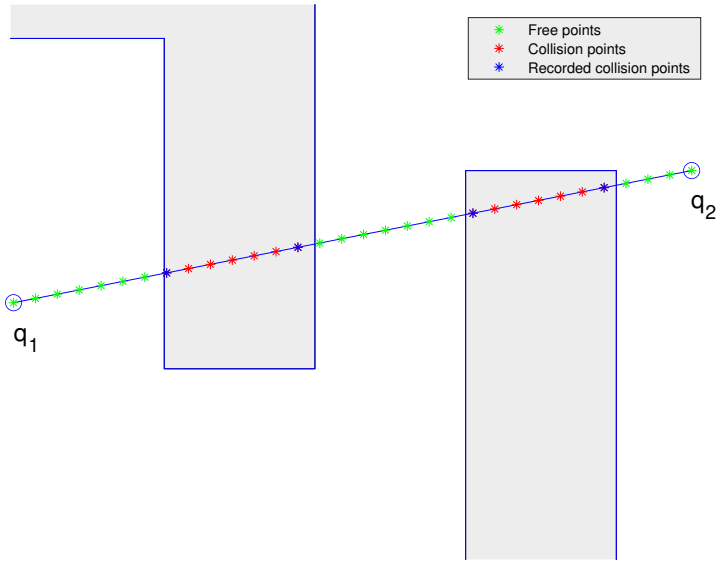


Figure 4·9: Dense collision detector of the local planner of PRM Trees

We use similar ideas for the local planners used to connect free points. In the connection process of PRM Trees, it is not necessary to keep all the points of the dense collision detector. But if we only keep the first collision point encountered, the exploration efficiency is decreased. The final choice is to keep all collision points adjacent to the free points, as shown in Figure 4·9. To put it another way, they are the collision points almost on the surface of all obstacles. They can retain the information of the obstacles well, and do not reduce the efficiency too much.

Figure 4·10 shows the path and all collision points obtained by the PRM Trees

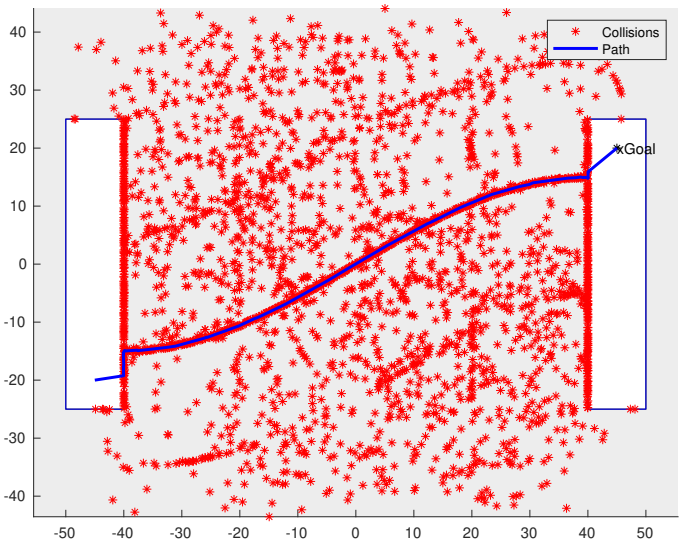


Figure 4·10: Path and collision points generated by PRM Trees with Obstacle Exploration in a configuration space with a sinusoidal narrow passage

	Running Time (s)		Path Length (unit)	
Algorithm	Mean	SD	Mean	SD
RRT	8.2525	20.5739	113.1604	0.6785
PRM Trees with OE (K=2)	0.8145	0.3823	122.3738	14.2629

Table 4.3: Performance comparison of RRT and PRM Trees with Obstacle Exploration in a configuration space with a sinusoidal narrow passage (one-hundred trials)

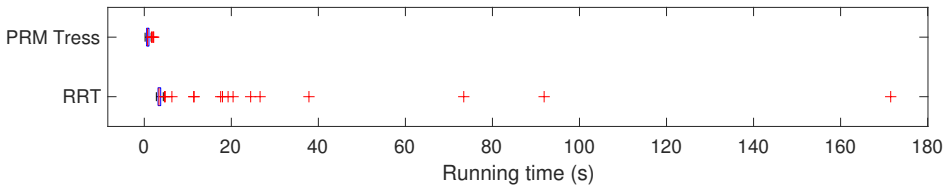


Figure 4·11: Running time of RRT and PRM Trees with Obstacle Exploration in a configuration space with a sinusoidal narrow passage

with the Obstacle Exploration algorithm after improving the local planner in an environment with a narrow sinusoidal passage. It can be seen that the collision points obtained in such an extremely complicated environment still do not fill the entire configuration space. It can be found in Table 4.3 and Figure 4·11 that in such an environment, the time taken by PRM Trees with Obstacle Exploration to find a path is only one-tenth of the time consumed by RRT, which is a great speedup. We can also find that the standard deviation of RRT, in this case, is too large. The reason is that RRT has some very high outliers.

PRM Trees with Obstacle Exploration without the improved local planner are almost unsuccessful at finding the path in such an environment, so it is not shown in the comparison.

Chapter 5

Conclusions

5.1 Summary of the Thesis

In this thesis, we addressed the problem of lack of constraints of search scope and the inefficiency of passing through narrow passages for sampling-based motion planning algorithms. One of the main contributions of our work is to utilize the polygonal obstacles in the configuration space to limit searching space and to generalize the method to non-polygonal configurations, which improve the performance of sampling-based algorithms in sparse environments and narrow passages.

We first developed an efficient sampling algorithm PRM Trees and showed that the relationship between the running time and the number of checked trees. We also found that a relatively small number of the checked trees resulted in better performance.

We introduced the method of Obstacle Activation to show how to selectively utilize polygonal obstacles encountered during path generation in the configuration space to narrow the scope of searching to increase the speed of finding a path for sampling-based algorithms. Based on Obstacle Activation, we developed the method of Obstacle Exploration, which can be utilized in a configuration space without the information of obstacles. Finally, we combined the PRM Trees algorithm and the Obstacle Exploration and created an efficient composite, sampling-based path planning algorithm, PRM Trees with Obstacle Exploration. By adjustment of the local planner, the composite algorithm was able to adapt to both sparse environments and

environments with narrow passages. We also noted that PRM Trees with Obstacle Exploration has good potential for generalized use in high-dimensional, non-convex spaces, which is a great advantage.

For a polygonal configuration space that its vertices of obstacles can be directly obtained, RRT with Obstacle Activation, which performed with the cosine distance and infinite step size, has the best performance of generating feasible paths in a sparse environment. However, if the obstacles are concave, RRT* with Obstacle Activation, which uniformly samples a pair of vertices, can generate a feasible path at a higher speed. At the same time, it can also get the shortest path through rewiring.

In a configuration space that cannot directly obtain obstacle information, PRM Trees with Obstacle Exploration can generate feasible paths. In particular, the local planner of Obstacle Exploration that only performs collision detection on a single sample can generate paths quickly in a sparse environment. In a space with a narrow passage, PRM Trees with Obstacle Exploration equipped with an improved local planner is much more effective.

5.1.1 Solutions to rotation

Although the methods we discussed in the thesis have many advantages, how to deal with space with a topological structure for them is still a problem worthy of discussion.

In an environment that allows rotation whose range is much larger than 0 to 2π, how to record and connect points becomes a problem. The discrete methods of PRM and RRT make them avoid this problem. For PRM, the local planner of PRM connects points in a small local range by controlling the radius. For RRT, it has incremental distance, and its extension is relative.

However, the connection range of PRM Trees is almost global, which leads to multiple choices when connecting in a space with topological properties. Because for any given two points, there are countless ways to connect them by the local

planner. Trying only the shortest connection may cause the failure of path generation. Fortunately, with Obstacle Exploration, this problem can be solved, because Obstacle Exploration itself is RRT. Obstacle Exploration can record the absolute coordinates of each point and, therefore, the overlapping coordinate axes can be expanded.

Admittedly, due to the rotation, obstacles may be unbounded, PRM Trees with Obstacle Exploration loses its ability to narrow the search scope. This is not a problem only faced by PRM Trees with Obstacle Exploration, but other algorithms that speed up path generation by narrowing the search scope also will face the same problem, such as informed RRT*.

5.1.2 Avenues for incorporating machine learning

Current research is still focused on sampling-based algorithms. As the research progresses, there must be more geometric knowledge and machine learning algorithms that can be used to promote the development of sampling-based algorithms.

Not all the traditional machine learning algorithms are suitable for the improvement of sampling-based algorithms, such as neural networks based on grids. A map that can be discretized into a grid means that a comprehensive understanding of the map can be obtained, which is impractical in many scenarios. Another disadvantage is that the neural network still needs a large number of samples to train a model. Compared with neural networks with grids, knowledge of clustering algorithms that can be used locally is more helpful for sampling-based algorithms. A principal component analysis used by SR-RRT (Lee et al., 2012) is a good example; it can increase the probability of passing through narrow passages by analyzing the local environment. However, each successful extension of the algorithm requires multiple uses of the collision detector, which is a disadvantage. As in a sparse environment, excessive use of collision detectors can lead to penalties, and speed can drop dramatically.

Admittedly, analyzing the local environment can avoid the expensive process of

getting information from the entire environment and significantly improve efficiency. Only considering the local environment also makes it difficult for sampling-based algorithms to obtain the optimal solution, which results in unnecessary, redundant information in the obtained graph. There should be a balance between the local environment and the global environment, but it is hard to achieve. For example, in Chapter 4, the reason for choosing the combination of PRM Trees and Obstacle Exploration rather than the combination of RRT and Obstacle Exploration is to avoid the problem of determining the weight of which tree to extend, which is a very complicated queuing theory problem.

5.2 Future work

During the completion of the thesis, there are many ideas that I would like to try and methods that I want to develop in the future. Future work concerns better methods or new proposals to try different approaches. Most of the ideas concern Obstacle Exploration and then how to generalize the idea to actual robots. The following idea is a list of possible future research directions:

1. The decision mechanisms for allocating Obstacle Exploration and the graph extension: The RRT with Obstacle Exploration, as mentioned above, has already been tried, and its performance was satisfactory. However, the decision-making method currently being used is not intelligent enough. Additionally, how to allocate the weight to extend the collision points or the free points will require more study about queuing theory and machine learning. Furthermore, we expect that machine learning algorithms will complement the decision-maker in the future.

2. Generalization of Obstacle Exploration to some actual implementations with kinematic and dynamic constraints: generalizing the idea to a robotic arm is

not difficult, but for a nonholonomic system, like a car or a tractor-trailer, it becomes a rather tricky problem. Obstacle Exploration will probably lose its ability to complete a narrow search scope in these cases.

3. Improvement of Obstacle Exploration: In Chapter 4, we discussed what kind of local planner can effectively improve the efficiency of Obstacle Exploration. Among the many issues, an important one is to use a more advanced function to decide how to explore obstacles. A lot of methods in the design of the local planner have been tried, which include using different distance equations to select extended nodes. We also tried to introduce more uncertainty into the local planner. For example, the inverse ratio of distance is used to determine the probability, and then nodes are randomly selected according to the probability. It is interesting that, in some cases, a well-defined distance function does improve the efficiency of the Obstacle Exploration. However, the parameters of the distance function are relevant to the corresponding configuration space, so it seems necessary, even if complicated, to develop a set of adaptive parameters and distance functions that apply to general situations.

References

Amato, N. M., Bayazit, O. B., Dale, L. K., Jones, C., and Vallejo, D. (1998). Obprm: An obstacle-based prm for 3d workspaces. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: The Algorithmic Perspective: The Algorithmic Perspective*, WAFR '98, page 155–168, USA. A. K. Peters, Ltd.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405.

Boor, V., Overmars, M. H., and van der Stappen, A. F. (1999). The gaussian sampling strategy for probabilistic roadmap planners. *In proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, 2:1018–1023 vol.2.

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2005). *Principles of Robot Motion: theory, algorithms, and implementation*. MIT Press.

Clifton, M., Paul, G., Kwok, N., Liu, D., and Wang, D. (2008). Evaluating performance of multiple rrts. In *2008 IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, pages 564–569.

Devaurs, D., Siméon, T., and Cortés, J. (2014). A multi-tree extension of the transition-based rrt: Application to ordering-and-pathfinding problems in continuous cost spaces. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2991–2996.

Gammell, J. D., Srinivasa, S. S., and Barfoot, T. D. (2014). Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004.

Ghosh, S. and Mount, D. (1991). An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hormann, K. and Agathos, A. (2001). The point in polygon problem for arbitrary polygons. *Computational Geometry: Theory and Applications*, 20(3):131–144.

Hsu, D., Tingting Jiang, Reif, J., and Zheng Sun (2003). The bridge test for sampling narrow passages with probabilistic roadmap planners. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 3, pages 4420–4426 vol.3.

Ju, T., Liu, S., Yang, J., and Sun, D. (2011). Apply rrt-based path planning to robotic manipulation of biological cells with optical tweezer. In *2011 IEEE International Conference on Mechatronics and Automation*, pages 221–226.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894.

Kavraki, L. and Latombe, J. . (1994). Randomized preprocessing of configuration for fast path planning. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 2138–2145 vol.3.

Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2.

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., and How, J. P. (2009). Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University.

Lee, D.-T. (1978). *Proximity and Reachability in the Plane.* PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA. AAI7913526.

Lee, J., Kwon, O., Zhang, L., and Yoon, S.-e. (2012). Sr-rrt: Selective retraction-based rrt planner. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2543–2550.

Li, J., Liu, S., Zhang, B., and Zhao, X. (2014). Rrt-a* motion planning algorithm for non-holonomic mobile robot. *2014 Proceedings of the SICE Annual Conference (SICE)*, pages 1833–1838.

Liangjun Zhang and Manocha, D. (2008). An efficient retraction-based rrt planner. In *2008 IEEE International Conference on Robotics and Automation*, pages 3743–3750.

Liu, Y. and Arimoto, S. (1992). Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *The International Journal of Robotics Research*, 11:376 – 382.

Lozano-Pérez, T. and Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22:560–570.

MATLAB (2018). *9.7.0.1190202 (R2019b)*. The MathWorks Inc., Natick, Massachusetts.

Pan, J., Zhang, L., and Manocha, D. (2010). Retraction-based rrt planner for articulated models. In *2010 IEEE International Conference on Robotics and Automation*, pages 2529–2536.

Pocchiola, M. and Vegter, G. (1996). Topologically sweeping visibility complexes via pseudotriangulations. *Discrete & Computational Geometry*, 16(4):419–453.

Qureshi, A. H., Mumtaz, S., Ayaz, Y., Hasan, O., Muhammad, M. S., and Mahmood, M. T. (2015). Triangular geometrized sampling heuristics for fast optimal motion planning. *International Journal of Advanced Robotic Systems*, 12(2):10.

Santos, L., Ferraz, N., Neves dos Santos, F., Mendes, J., Morais, R., Costa, P., and Reis, R. (2018). Path planning aware of soil compaction for steep slope vineyards. In *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 250–255.

Singhal, A. (2001). Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43.

Wang, W., Xu, X., Li, Y., Song, J., and He, H. (2010). Triple rrts: An effective method for path planning in narrow passages. *Advanced Robotics*, 24(7):943–962.

Wedge, N. A. and Branicky, M. S. (2011). Using path-length localized rrt-like search to solve challenging planning problems. In *2011 IEEE International Conference on Robotics and Automation*, pages 3713–3718.

Yeh, H., Thomas, S., Eppstein, D., and Amato, N. M. (2012). Uobprm: A uniformly distributed obstacle-based prm. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2655–2662.

Yu, F., Chen, Y., Zhang, X., and Xin, X. (2019). Rapidly-exploring random trees based on cylindrical sampling space. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 539–544.

CURRICULUM VITAE