Robotica: (2019) volume 37, pp. 1677–1694. © Cambridge University Press 2019
doi:10.1017/S0263574719000195

Finding a High-Quality Initial Solution for the RRTs Algorithms in 2D Environments

Jiankun Wang†, Wenzheng Chi‡, Mingjie Shao† and Max Q.-H. Meng†*

†*Department of Electronic Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong SAR, China. E-mails: jkwang@ee.cuhk.edu.hk, mjshao@ee.cuhk.edu.hk*
‡*Robotics and Microsystems Center, School of Mechanical and Electric Engineering, Soochow University, Suzhou, China. E-mail: wzchi@suda.edu.cn*

(Accepted January 23, 2019. First published online: February 21, 2019)

SUMMARY
In this paper, we propose a bioinspired path planning algorithm for finding a high-quality initial solution based on the pipeline of the Rapidly exploring Random Tree (RRT) method by modifying the sampling process. The modification mainly includes controlling the sampling space and using the probabilistic sampling with the two-dimensional Gaussian mixture model. Inspired by the tropism of plants, we use a Gaussian mixture model to imitate the tree's growth in nature. In a 2D environment, we can get an approximate moving point's probabilistic distribution, and the initial path can be found much quickly guided by the probabilistic heuristic. At the same time, only a small number of nodes are generated, which can reduce the memory usage. As a meta-algorithm, it can be applicable to other RRT methods and the performance of underlying algorithm is improved dramatically. We also prove that the probabilistic completeness and the asymptotic optimality depend on the original algorithm (other RRTs). We demonstrate the application of our algorithm in different simulated 2D environments. On these scenarios, our algorithm outperforms the RRT and the RRT* methods on finding the initial solution. When embedded into post-processing algorithms like the Informed RRT*, it also promotes the convergence speed and saves the memory usage.

KEYWORDS: Path planning; Motion planning; Robotics; Mobile robot.

1. Introduction

Robot path planning[1–3] is a basic research domain in robotics, which aims to finding a feasible path for the robot in a given environment if it exists. Recently, growing interest has been placed on finding high-quality paths. The quality of the path can be measured by several criteria according to specific applications, such as the path length, the computational time of the corresponding planner, the clearance to the obstacles, the smoothness of the generated path and the memory usage. In this paper, we take the computational time and memory usage as main criteria. The path length is also discussed on the problem of finding the initial solution.

In general, the artificial potential field (APF), the graph-based and the sampling-based algorithms are usually used to solve path planning problems. The APF method[4] directs the motion of the robot through an APF defined by a function over the whole free configuration space. A feasible path is found by following the direction of the steepest decent of the potential. However, much computational time is required and this method may end up in a local minimum.

* Corresponding author. E-mail: max.meng@cuhk.edu.hk

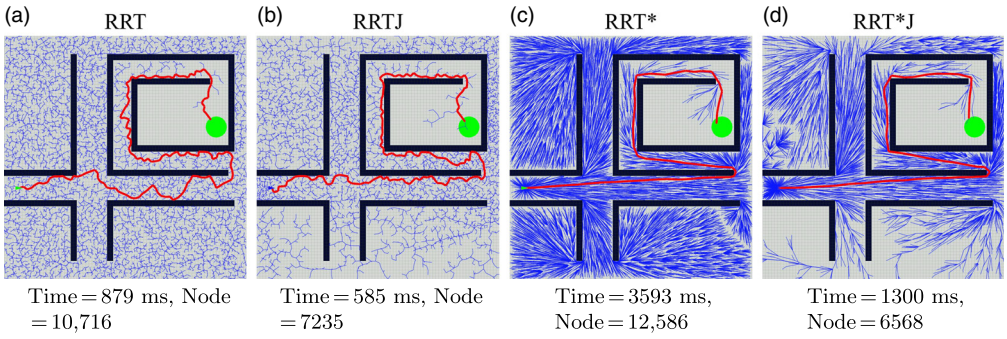


Fig. 1. Experimental results on finding the initial solution in maze environments. The RRTJ and the RRT*J represent improved RRT and RRT* with our meta-algorithm. The small triangle in the left represents the start position, the big green circle in the right represents the goal region, the black rectangle represents the obstacle and the red line represents the initial solution. Time is the execution time of the whole algorithm and Node is the number of node of the tree at last. (a) and (b) show the results of the RRT and the RRTJ, while (c) and (d) show the results of the RRT* and the RRT*J. The number of node can represent the memory usage because different algorithms use almost the same memory usage in other places.

The graph-based algorithms, such as the A*[5] and the D*,[6] are resolution complete and resolution optimal, meaning that they can always find the optimal solution if it exists. But these algorithms have little clearance to obstacles and they do not perform well as the problem scale increases.

The sampling-based algorithms, such as the Rapidly exploring Random Tree (RRT)[7] and the Probabilistic Roadmap (PRM)[8] methods, are very popular because they can avoid the local minimum problem and scale well with problem size. The RRT method is probabilistic complete, which means that the probability of finding a feasible solution approaches to one if it exists with the number of iterations approaching to infinity. But solutions from the RRT algorithms are often far from optimal. Recently, many variants of the RRT methods have been proposed to find the optimal or near-optimal solutions efficiently. The sample biasing,[9–13] the post-processing methods like the shortcutting,[14,15] and the path hybridization[16–19] have been shown very effective at speeding up the convergence rate and finding the optimal solution from a combination of the input paths.

In addition to aforementioned methods, meta-heuristic methods are also used in the path planning problem. Ant Colony algorithm (AC)[20] and Genetic algorithm (GA)[21] are usually applied into the Traveling Salesman Problem, which is a task assignment problem based on the generated path. Inspired by the AC and GA, particle swarm optimization (PSO)[22] is another efficient method, which can optimize the solution in an iterative manner with a swarm of particles. A modified version of PSO[23] is proposed to allow the robot to adjust its moving direction in terms of the movements of the targets and obstacles in the dynamic environment.

Reinforcement learning method[24] achieves the path planning by continuously optimizing the action policy, which is an off-line method. By learning from the demonstrations, inverse reinforcement learning[25] is widely used in the social-aware path planning. They aim at learning different cost functions in different scenarios.

However, to our best knowledge, few research efforts pay attention to the initial solution by using sampling-based methods. In fact, the initial solution has a big impact on the algorithm implementation. A good initial solution can save much computational time and memory usage. The A*-RRT*[16] and the Thete*-RRT[17] methods are hybridization of the graph-based and the sampling-based algorithms. However, like other graph-based algorithms, their scalability is weak on finding the initial solution. Meanwhile, the smoothness is lacking and the clearance to the obstacles is little. More post-processing work is needed to get a high-quality path.

In this paper, we propose a meta-algorithm (as shown in Fig. 1) through controlling the sampling space and using the probabilistic sampling with the Gaussian mixture model. With such a probabilistic heuristic, a feasible path can be found quickly with less memory usage. Our motivation is to balance the exploration and the exploitation. Generally, the RRTs algorithms try to explore the whole map to find an initial solution at first, then the exploitation process is implemented to optimize the initial solution. In this way, the completeness can be probabilistically guaranteed. However, as the exploration area expands, much time and the computational resources are required in the exploitation

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

process. In many cases, such as the home or the environment where a solution is easy to find, uniformly exploring the whole map is not necessary. Motivated by this observation and inspired by the tropism of plants (discussed in Section 3.2), we propose this meta-algorithm. Naturally, it is applicable to the RRTs algorithms like the RRT,[7] the RRT*,[26] and Informed RRT*[14] methods to further improve their performance. Our contributions include:

- a meta-algorithm to find a high-quality initial solution for the RRTs algorithms;
- a tree coverage detection method for the probabilistic sampling process; and
- a probabilistic sampling method based on the Gaussian mixture model.

This paper is organized as follows. We review related existing work in Section 2. Section 3 gives a problem statement and our inspiration about this work. Section 4 introduces the framework of our algorithm, the coverage detection method and the probabilistic sampling process. The probabilistic completeness and the asymptotic optimality of our algorithm are also proved in this section. In Section 5, we discuss our experiments and demonstrate the efficiency of our algorithm. Conclusions and future work are discussed in Section 6.

2. Related Work

Many variants of the RRTs have been proposed in order to find high-quality solutions. These prior related work mainly consists of the sample biasing, the post-processing methods, the hybridization methods, the asymptotically optimal and near-optimal methods, and the anytime RRTs.

Compared to other work, our contributions lie in finding an initial solution efficiently with less memory usage by using the sampling-biasing methods, and the proposed algorithm can be applicable to post-processing, asymptotically optimal methods and other RRTs algorithms.

2.1. Sample biasing

Sample biasing makes use of the improved sampling strategies. Similar to the online optimization, these strategies bias the search to find high-quality solutions during the whole planning process. Urmson and Simmons[9] utilize a heuristic quality function to guide the growth of an RRT. Incorporating the cost of the path provides extra information to an RRT that allows it to operate in more than an exploratory manner. A new sampling strategy based on an estimated feasibility set, named the RG-RRT,[10] is proposed to afford a big improvement in performance for differential constrained systems. Alleviating the sensitivity of the random sampling to a metric that is used to expand the tree, the result of the RG-RRT is a modified Voronoi bias focusing on the nodes that probably promotes exploration given the system dynamics constraints. By using obstacle boundary information, a variant of the RRT[11] is proposed to sample more nodes near the obstacle that is more likely to block the feasible path. The sampling process is biased and the time cost of the whole planning process decreases. Additional methods include, among others, the f-biasing (a heuristic based on the estimate of solution cost guides sparse sampling)[12] and the multiple random restarts[13] methods.

2.2. Post-processing methods

The shortcutting is a commonly used intuitive post-processing method. Generally, two nonconsecutive configurations are selected randomly along the original path. If they can be connected by a straight line and the connection improves the path quality, the original path segments are replaced by the straight line. Geraerts and Overmars[15] report the partial shortcut method, which can remove redundant motions and decrease the path length by interpolating one degree of freedom at a time.

Another effective post-processing method is the Informed RRT*[14] method. It uses the subset of the states that can improve a solution as a prolate hyperspheroid and uses an admissible ellipsoidal heuristic in the sampling process. A quick convergence rate is reported.

2.3. Hybridization methods

The path hybridization, introduced in ref. [18], takes two or more initial solutions as the input and constructs a higher quality path by comparing the quality of certain subpaths within each solution and the quality of the entire path.

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

The hybridization of the graph-based and the sampling-based algorithms, such as the A*-RRT*,[16] the Thete*-RRT[17] and the RRT#[19] methods, takes advantage of the speed of the graph search and the uniform distribution of the random sampling, and results in a better performance.

2.4. Asymptotically optimal and near-optimal methods

Karaman and Frazzoli[26] show that the cost of the solution returned by the sampling-based algorithms like the RRT and the PRM converges to a suboptimal value. By modifying the connection scheme of a new node to the existing tree, they present the RRT* and the PRM*, which are shown to be asymptotically optimal. As the number of iterations tends to infinity, the probability of finding an optimal solution approaches to one. The fast marching tree (FMT*)[27] is also asymptotically optimal. It combines the features of both the single-query and the multiple-query algorithms and implements a lazy dynamic recursion given a number of probabilistically drawn samples, so it outperforms the RRT* and PRM* methods.

The concept of asymptotically near-optimal means that the current solution converges to within an approximation factor of $(1+\epsilon)$ of the optimal solution with probability of one as the number of iterations approaches to infinity. By this relaxation, the algorithm performance gets improved. The stable Sparse RRT (SST)[28] is an typical example, which does not access the steering function, maintains a sparse set of samples and converges fast to a high-quality path. The lower bound tree-RRT (LBT-RRT)[29] is another asymptotically near-optimal algorithm for fast and high-quality motion planning by using the fast all-pairs r-nearest neighbors (NN) and the lower bound on the cost.

2.5. Anytime RRTs

Anytime path planning means that the algorithm may be terminated anytime, while the time to plan is not known and no specified cost is predetermined. Namely, any solution should be found quickly and a high-quality solution can be reached if time permits. Ferguson and Stentz[30] present a non-uniform search focusing on the area that can potentially improve the quality of the existing paths. It can be combined with other sampling-based planners to perform like an anytime RRT. The RT-RRT*[31] is another variant of the anytime RRTs with an online tree rewiring strategy, which works well under the dynamic differential constraints.

3. Preliminaries

3.1. Problem formulation

In this paper, we only consider the path planning problem in a 2D environment. Let $X \subset \mathbb{R}^2$ be the state space. Let $X_{free} \subset X$ be the free space and $X_{obs} = X \setminus X_{free}$ be the obstacle space. Assume the robot as a point whose path is represented by a curve $\phi(s) : [0, 1] \to \mathbb{R}^2$. Here $\phi(s)$ means the intermediate state in the whole path.

We call $\phi(s)$ a feasible path if the robot can move from a start point x_{init} to the goal region X_{goal} such that

$$\phi(s) : [0, 1] \to X_{free}, \phi(0) = x_{init}, \phi(1) \in X_{goal} \tag{1}$$

where $X_{goal} = \{x \in X_{free} \big| \ ||x - x_{goal}|| \leq r\}$.

Denote the set of all feasible paths by Φ. With a cost function $c : \Phi \to \mathbb{R}_{\geq 0}$, the optimal path planning problem can be formulated as

$$\begin{aligned} \phi^* = \arg\min_{\phi \in \Phi} \{c(\phi) | \phi(0) = x_{init}, \ \phi(1) \in X_{goal}, \\ \forall s \in [0, 1], \ \phi(s) \in X_{free}\}. \end{aligned} \tag{2}$$

3.2. Inspiration: tropism of plants

Tropism of plants is a biological phenomenon, indicating the growth movement of a plant in response to an environment stimulus. Phototropism, one of the main tropisms of plants, is defined as the impact from light stimulus. Most shoots of plant exhibit positive phototropism and grow toward light. When the obstacles block the sunlight, the tree will grow the branches laterally to bypass them. As a result, the lateral growth space of the tree is expanded. When the tree touches the sunlight, it grows branches

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

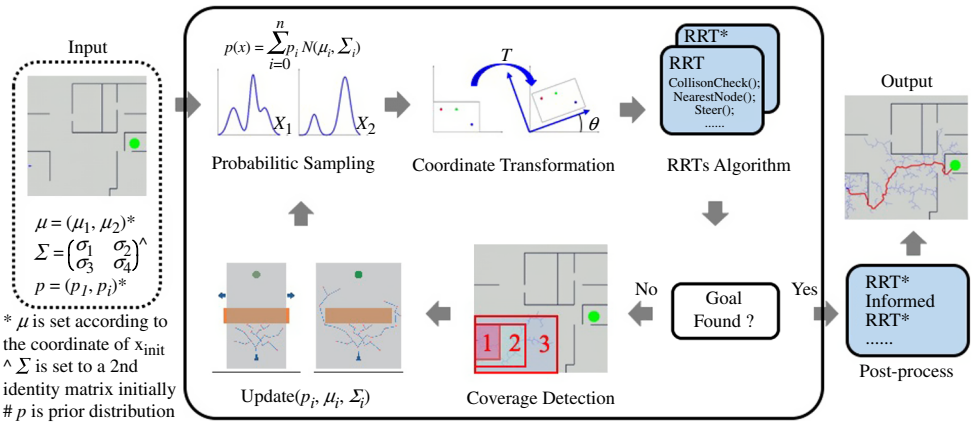


Fig. 2. Algorithm framework. The black box contains the main algorithm. The input is a given map, a two-dimensional mean vector, a 2×2 covariance matrix and a prior distribution. The output is the final result based on the main algorithm and the post-process methods. In the main algorithm, probabilistic sampling represents the sampling method according to the Gaussian mixture model, and the horizontal and vertical coordinates (x_1, x_2) are selected, respectively. Coordinate transformation shows a transformation from the sampling coordinate system to the world coordinate system with a matrix T. RRTs algorithm means that our algorithm can be embedded into them as pre-process and post-process steps. If the goal is found in aforementioned steps, the algorithm steps into post-process for further optimization. If not found, the algorithm implements coverage detection. The red number and the red rectangular show that only the incremental area is calculated. Update(p_i, μ_i, Σ_i) means the value of p_i, μ_i and Σ_i are updated according the result of coverage detection.

vertically and stops the expansion of the lateral growth space. The positive phototropism enables the tree to obtain photosynthetic energy as much as possible and limits the lateral growth space as little as possible.

We use a probabilistic sampling process based on Gaussian mixture model to imitate the phototropism of plants, as shown in Fig. 2. We discuss the details of this probabilistic sampling process in Section 4.2.

4. Algorithm

Figure 2 shows the frame of our algorithm. The input is an environment map, mean value vector μ, covariance matrix Σ and prior distribution p. In the probabilistic sampling, we use a Gaussian mixture model to generate nodes in the sampling coordinate system. By a matrix T, a transformation from the sampling coordinate system to the world coordinate system is achieved. These nodes can be directly used in the RRTs algorithms like the RRT and the RRT*, which are pipelines that our algorithm relies on. In each iteration, the algorithm detects whether the goal is found or not. If not, the coverage detection is implemented and μ_i, Σ_i and p_i are updated, then a new loop begins. Or post-process methods will be used to generate the final result.

The reason why we use the Gaussian mixture model is that we hope the sampling process focuses on the region which potentially contains a solution. As the tree grows, we select the treetop (a node of the tree nearest to the goal position) as the sampling center. Sampling backward can promote the exploitation (rewiring process of the RRTs algorithms) and sampling forward can accelerate the exploration. A major concern of our method is the performance in some cases where a valid path would need to pass close to the environment's border or need to make some complex turns. In fact, with the probabilistic sampling process in our algorithm, the performance is still better than the RRT and the RRT* on finding the initial solution. Because, in our Gaussian mixture model, the probability of sampling nodes in the boundary area of sampling space becomes larger according to the change of p_i in such scenarios. In other words, the probabilistic sampling is automatically adjusted to different scenarios. In the experiment section, we use some challenging environments to demonstrate the efficiency of our algorithm.

In order to implement the probabilistic sampling, we need two coordinate systems including world coordinate system and sampling coordinate system. First, we sample candidate nodes under the sampling coordinate system (shown in Fig. 2) with Gaussian mixture model. Then we transform the coordinates of these nodes into the world coordinate system using T, where

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

$$T = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \tag{3}$$

In the world coordinates, the x-axis is parallel to the line connecting the start and goal position, then θ denotes the angle between the sampling coordinate system (denoted with black) and the world coordinate system (denoted with blue), as shown in Fig. 2.

The Gaussian mixture model can be formulated with μ, Σ and p, which can be represented as

$$p(x) = \sum_{i=1}^{n} p_i \mathcal{N}(\mu_i, \Sigma_i), \ \sum_{i=1}^{n} p_i = 1 \tag{4}$$

where $\mathcal{N}$ represents a two-dimensional Gaussian distribution and p_i is a weight of ith component. $x = (x_1, x_2)$ is a two-dimensional vector (x_1 and x_2 corresponding to the horizontal and vertical directions, respectively, in the sampling coordinate system), μ_i is a two-dimensional mean vector and Σ_i is a 2×2 covariance matrix.

For the covariance matrix $\Sigma = [\sigma_1 \ \sigma_2; \sigma_3 \ \sigma_4]$, if σ_2 and σ_3 are not zero, the two-dimensional Gaussian distribution will not symmetrically relative to the axes. For simplicity, we assume $\sigma_2 = \sigma_3 = 0$. So Σ becomes

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \tag{5}$$

σ_1 and σ_2 denote the standard deviation of the two-dimensional Gaussian distribution in horizontal and vertical directions, respectively. For implementation, we split the two-dimensional Gaussian mixture model in two one-dimensional models:

$$\begin{aligned} p(x_1) &= \sum_{i=1}^{n} m_i \mathcal{N}(\mu_{1i}, \sigma_{1i}), \ \sum_{i=1}^{n} m_i = 1, \\ p(x_2) &= \sum_{i=1}^{n} n_i \mathcal{N}(\mu_{2i}, \sigma_{2i}), \ \sum_{i=1}^{n} n_i = 1 \end{aligned} \tag{6}$$

which correspond to the horizontal and vertical coordinates of one node x, respectively. The initial setting and the update of μ_i, Σ_i and p_i will be discussed in Section 4.2.

First, we apply our algorithm to the RRT* in Alg. 1 as an example. We construct a tree $\mathcal{T} = (V, E)$ consisting of a vertex set $V \subset X_{free}$ and edges $E \subseteq V \times V$. The key changes are **CoverageDetection**(μ, Σ) and **ProbabilisticSampling**(μ, Σ).

4.1. Coverage detection

The concept of the coverage detection is the key to our algorithm. As mentioned before, our algorithm is based on controlling the sampling space and using the probabilistic sampling. An appropriate control of sampling space is good for finding an initial solution quickly and accelerating the optimization speed, while a bad control slows the exploration speed and has a bad influence on the performance of our algorithm. So the control criteria are essential.

Consider a tree in natural environment; if the nutrition is enough and other external factors do not affect its growth (not blocked by other trees or obstacles), this tree will grow as high as it can. Similar to the RRT, if there is no obstacle in a given sampling space, the tree will grow to the goal region quickly. If not, current sampling space is limiting the growth of the tree and a larger sampling space is needed. The key issue is when and how to adjust the sampling space. An intuitive idea is that the sampling space should be adjusted when the nodes of the tree are too dense. But the detection is hard. First, a criterion to effectively measure the density should be put forward, which needs to consider the current growth of the tree, the size of sampling space and the obstacle area. Secondly, the detection is required to be very fast and does not depend on much computational resource. Slow detection affects the speed of the whole algorithm and using too much computational resource does not scale well in a large map or high-dimensional environment.

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

Algorithm 1: Alg(x_{init}, x_{goal})

```
V ← x_init, E ← ∅, T = (V, E), goal = false;
μ_1 = μ_2 = 0, σ_1 = σ_2 = 1;
GriddingProcess(map);
for i = 1...N do
    if goal == false then
        x_rand = ProbabilisticSampling(μ, Σ);
    else
        x_rand = RandomSampling();
    x_nearest = Nearest(T, x_rand);
    x_new = Steer(x_nearest, x_rand);
    if Line(x_nearest, x_new) ∈ X_free then
        if x_new ∈ X_goal then
            V ← V ∪ {x_new};
            E ← E ∪ {(x_nearest, x_new)};
            return T;
        X_near = NearestNeighbours(T, x_new);
        x_min = x_nearest;
        c_min = Cost(x_min) + Dist(x_min, x_new);
        for x_near ∈ X_near do
            c_new = Cost(x_near) + Dist(x_near, x_new);
            if c_new < c_min then
                if Line(x_near, x_new) ∈ X_free then
                    c_min = c_new, x_min = x_near;
        V ← V ∪ {x_new}, E ← E ∪ {(x_min, x_new)};
        Rewire();
        n = CalGrid(Σ);
    if !goal && Mod(i, counters) == 0 then
        CoverageDetection(μ, Σ);
return T;
```

In terms of these two factors, we introduce the concept of the coverage detection and propose a corresponding algorithm to measure the density effectively and quickly. **GriddingProcess(***map***)** and **CalGrid(**Σ**)** are two pre-processing steps in our coverage detection algorithm.

GriddingProcess(*map***)** means the map is gridded into squares with the side length of k, and

$$k = \frac{\epsilon}{2} \tag{7}$$

where ϵ is the maximum distance between $x_{nearest}$ and x_{new} in **Steer(**$x_{nearest}$, x_{rand}**)**. Using the grid representation, the area of the tree and the sampling space can be compared in the same order of magnitude.

Through hundreds of experiments in different environments, **GriddingProcess(***map***)** can be finished in 0.5–1 ms and takes up less than 1% in the whole computation time, which is negligible compared with the time cost of the algorithm on finding the initial solution.

CalGrid(Σ**)** traverses an area, which is a rectangular bounding box of current tree, and returns the number of current grids occupied by the tree nodes. We use a set N to represent the grid map. The value of each grid is initially set to 0. If a grid is occupied or partly occupied by obstacles, the value becomes -1. If a node is added to the tree and it is enclosed by a grid, then the value of this grid is set to 1. One grid can enclose more than one node, but the maximum value of this grid is 1. If the value of one grid is -1, it can become 1, and if the value of one grid is 1, it also can become -1 again when it is occupied by a moving obstacle. So the number of grids occupied by the tree is

$$n = Count(N, 1) \tag{8}$$

The process of **CalGrid(**Σ**)** is very fast for two reasons. First, we use a rectangular as the bounding box. We only store eight points consisting of maximum and minimum points in horizontal and

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

Algorithm 2: CoverageDetection(μ, Σ)

S_{tree} = CalGrid(Σ);
S_{obs} = CalObsArea(Σ);
$TCR = nk^2/(S_{tree} - S_{obs})$;
Σ = Expand(Σ);
μ = Update(μ);

vertical directions and their updates. The rectangular is not very suitable to be the contour of the tree, but through a large number of experiments we find the density is usually very small and this rectangular bounding box almost brings no influence. Second, it is implemented according to current growth of the tree (shown in Fig. 2). Each time only the incremental area is calculated. It means the incremental area of [rectangular 2–rectangular 1] and [rectangular 3–rectangular 2] will be considered instead of [rectangular 2] and [rectangular 3]. Therefore, the algorithm can be implemented very fast.

In addition, we adopt a "black-box" collision checking algorithm. We do not need to know the shape or position of the obstacles. **CalGrid(Σ)** is only a simple traversal process. Compared to general RRTs algorithm, the further knowledge we need to acquire is that we use an array to store the number of 0 and 1 of current sampling space.

Definition 1 (Node Area). In the square gridded sampling space, suppose the nodes of the tree occupy n grids and the side length of each grid is k, then the node area S_{node} is defined as

$$S_{node} = nk^2 \tag{9}$$

Definition 2 (Tree Area). The area of the rectangle bounding box of the current tree is defined as tree area S_{tree}.

Definition 3 (Tree Coverage Rate). Suppose in the current tree area S_{tree}, the area of the obstacle space is S_{obs}, then we define the Tree Coverage Rate (TCR) as

$$TCR = \frac{S_{node}}{S_{tree} - S_{obs}} = \frac{nk^2}{S_{tree} - S_{obs}} \tag{10}$$

The area of S_{tree} equals the area traversed by **CalGrid(Σ)**. S_{obs} can be obtained through **CalObsArea(Σ)**. Actually, this process is implemented in **CalGrid(Σ)**.

TCR represents the growth condition of the tree under the current two-dimensional Gaussian mixture model. When the program runs for *counters* iterations and X_{goal} is not found, it will call **CoverageDetection(μ, Σ)**. The value of μ and Σ will be updated in **CoverageDetection(μ, Σ)**. **Update(μ)** updates the value of μ_1 and μ_2 while **Expand(Σ)** updates the value of σ_1 and σ_2. The details of the update process are provided in Section 4.2.

4.2. Probabilistic sampling

The probabilistic sampling process is implemented under the sampling coordinate system, with a matrix T the transformation from the sampling coordinate system to the world coordinate system is achieved.

The horizontal coordinate will be generated by μ_1 and σ_1.

For the value of σ_1, we build a relationship between TCR and σ_1. TCR represents the growth condition of the tree under current two-dimensional Gaussian distribution. When TCR is large and no feasible solution is found, it means that the growth of the tree is limited by obstacles around it. So we consider expanding the value of σ_1 to increase the probability of sampling nodes farther away from the center, μ_1, in horizontal direction. For finding the initial solution faster, σ_1 should be more sensitive to TCR at the beginning and become less sensitive to TCR as the probabilistic sampling process goes on. Because on the initial condition, a large number of generated nodes lie on the surrounding of the central line in horizontal direction, which is enough for finding one feasible solution if it exists in such a probabilistic sampling process. In the following steps, we need to control the area of the sampling space. If the area increases quickly, it affects the quality of the initial feasible solution and the convergence speed of following post-process. So the function of σ_1 and TCR, $f(TCR)$, should satisfy:

$$f'(TCR) > 0,\ f''(TCR) < 0. \tag{11}$$

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

Many logarithmic and quadratic functions satisfy the above two properties. Here we use a logarithmic function to construct the relationship between σ_1 and *TCR*.

Initially, the value of μ_1 is 0 for two reasons: (1) The shortest path is the line linking x_{init} and x_{goal} directly as long as there is no obstacle around it, so we set the default value of μ_1 as 0 at the beginning. (2) In the horizontal direction, the probability of growing to different side of the tree should be equal because the tree does not know which side is more beneficial for growing. So the value of μ_1 is still 0. For the purpose of accelerating the exploration speed and escaping from current limited sampling space, μ_1 needs to change. When the current sampling space limits the tree's growth, sampling in the boundary of current space with a large probability is reasonable. Consider the maximum value of horizontal coordinate of current tree is h_{max} and the minimum value is h_{min}. The value of μ_1 will change according to *TCR*. Most of the time, $\mu_1 = 0$ and $\mu_1 = h_{max}$ or h_{min} when the growth of the tree is limited severely. We define a new variable e^{-TCR}. Usually *TCR* is a small number (≤ 0.1) and $TCR \in (0, 1]$, so $e^{-TCR} \in [\frac{1}{e}, 1)$. In fact, e^{-TCR} approaches to 1 as usual.

Therefore, the Gaussian mixture model in horizontal direction is formulated as

$$p(x_1) = m_1\mathcal{N}(0, \sigma_1) + m_2\mathcal{N}(h_{max}, \sigma_1) + m_3\mathcal{N}(h_{min}, \sigma_1) \tag{12}$$

where $m_1 = e^{-TCR}$ and $m_2 = m_3 = \frac{1-e^{-TCR}}{2}$.

The vertical coordinate will be generated by μ_2 and σ_2. For the current tree, the vertical coordinate of the node nearest to x_{goal} in the vertical direction is considered as n_v (called treetop), which reflects the growth condition to x_{goal}. So the value of n_v is always changing until one feasible solution is found or the tree reaches the bounder of the current map.

In order to avoid ending up in a local minimum and getting stuck in some extreme environments (like the trap environment and the long corridor not leading to the goal region), sometimes the sampling center in the vertical direction should be opposite to the growth tendency of current tree. At this time, we have $n_{vo} = 2x_{init}.y - n_v$. y represents the vertical coordinate. For example, $x_{init}.y$ is the vertical coordinate of x_{init}. $\mu_2 = n_v$ or n_{vo}.

For the value of σ_2, it is a function about μ_2. This function $f(\mu_2)$ can be explicitly formulated as

$$f(\mu_2) = \frac{\mu_2 - x_{init}.y}{3} \tag{13}$$

It is obvious that $f(\mu_2)$ is a monotonically increasing function and the maximum depends on the map. The denominator 3 comes from the "3σ principle" in Gaussian distribution.

We still use e^{-TCR} to guide the probabilistic process. When e^{-TCR} is larger, it means current sampling space limits the tree's growth and the probability to sample in the opposite direction of the tree's growth tendency will increase. So the Gaussian mixture model in the vertical direction is formulated:

$$p(x_2) = n_1\mathcal{N}(n_v, \sigma_2) + n_2\mathcal{N}(n_{vo}, \sigma_2) \tag{14}$$

where $n_1 = e^{-TCR}$ and $n_2 = \frac{1-e^{-TCR}}{2}$.

Through μ_2 and σ_2, the probabilistic sampling process in the vertical direction depends on the current growth condition of the tree in vertical direction. At most time, $\mu_2 = n_v$. Naturally, sampling nodes around the treetop are very beneficial for the growth of this tree. According to the property of the Gaussian distribution, we know that 99.73% nodes will be located in the range $(-3\sigma_2, 3\sigma_2)$. The reason why we do that includes: (1) Sampling in the backward direction of μ_2 solves the "False Treetop" problem, which means that current treetop cannot continuously grow and the true treetop does not grow faster than the false one at this moment. (2) Sampling in the forward direction of μ_2 can make sure the treetop get enough space to grow. (3) Continuously increasing value of σ_2 can help find the initial feasible solution quickly.

4.3. Analysis of probabilistic completeness and asymptotic optimality

The probabilistic completeness is naturally guaranteed. According to the property of the Gaussian distribution, each point can be selected with different probability in theory. The points close to center are more likely to be selected, while the points far away from center are less likely to be selected. In addition, as a meta-algorithm, when the initial solution is found, it does not have an effect on the main algorithm any more. In the following process, the probabilistic completeness depends on the algorithm that our meta-algorithm is embedded in.

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

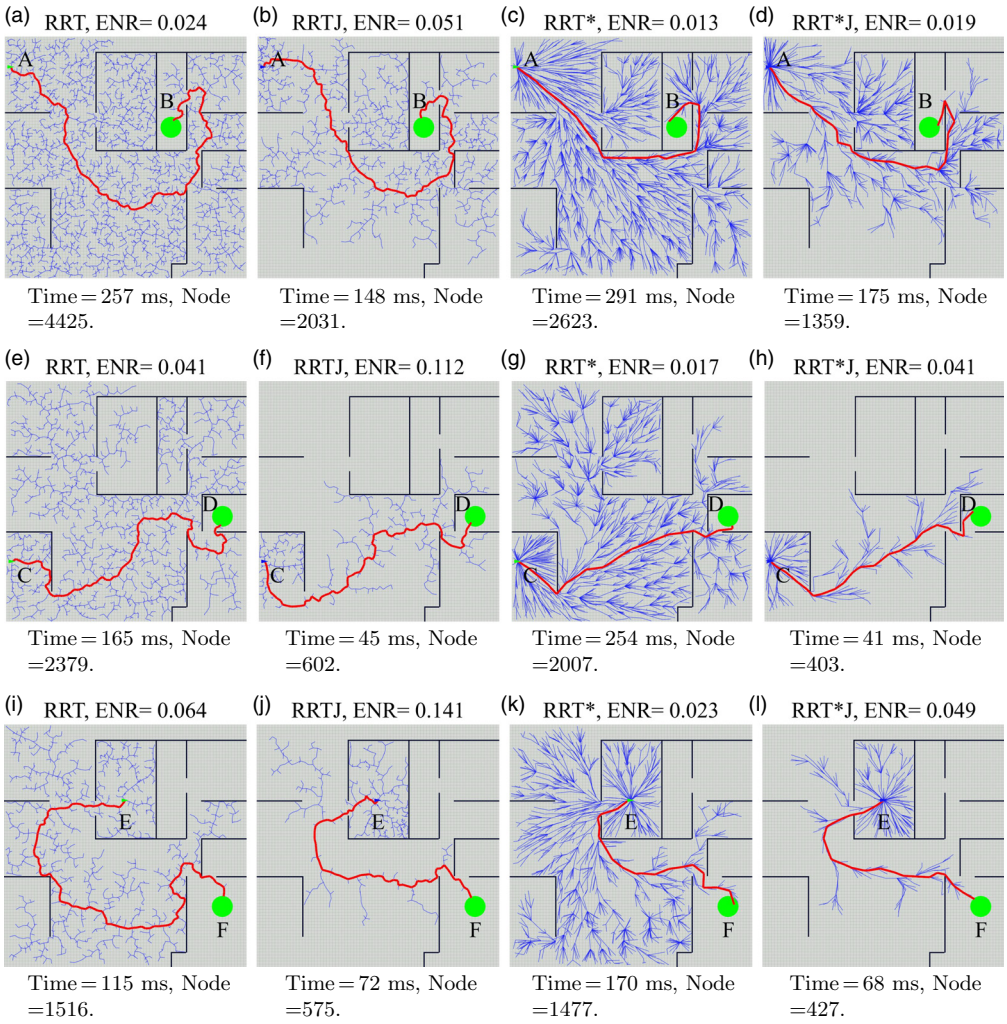


Fig. 3. Demonstration of experimental results in three scenarios with four algorithms. The RRTJ and the RRT*J represent improved RRT and RRT* with our meta-algorithm. The small triangle in the left represents the start position, the big circle in the right represents the goal region, the rectangle represents the obstacle and the bold line represents the initial solution. The *ENR* means the effective node rate, reflecting the utilization of sampled nodes. Time is the execution time of the whole algorithm and Node is the number of node of the tree at last.

Obviously, the asymptotic optimality depends on the algorithm that our meta-algorithm is embedded in because our algorithm only aims to find a high-quality initial solution.

5. Experiments and Results

In this paper, we adopt Windows 10.0 on an Intel i5-4590 with 8GB of RAM as our experimental platform. We use Visual Studio 2015 with openFrameworks as the Integrated Development Environment (IDE). As shown in Fig. 3, the goal region is a circle with the radius of 35 pixels. The home environment (shown in Fig. 3) is designed according to some general house floor plans. We intend to see the performance of our algorithm in such a practical environment. Different start and goal positions (Scenario $A \rightarrow B$, Scenario $C \rightarrow D$ and Scenario $E \rightarrow F$) are selected to test the efficiency and robustness (the performance under different configurations). In scenario $A \rightarrow B$, the tree needs to make a complex turn and grows backward when approaching the goal region. In scenario $C \rightarrow D$, the tree needs to make simple turns at the beginning and at the end. In scenario $E \rightarrow F$, the tree is required to grow backward to escape from a small trap at first. The maze environment (shown in Fig. 1) is used to test the generality of our algorithm.

Except for the aforementioned evaluations, we further analyze the algorithm based on the fundamental tasks that a path planner is required to do: exploration and exploitation. The RRTs algorithms

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

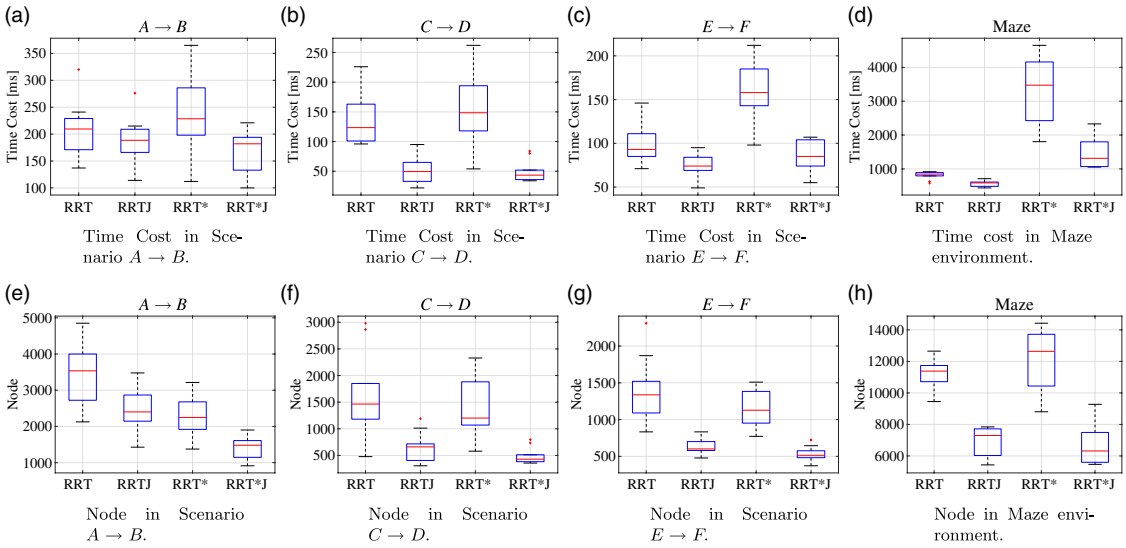


Fig. 4. Experimental results on finding the initial solution. The RRTJ and the RRT*J represent improved RRT and RRT* with our meta-algorithm. (d) and (h) show the results on maze environment, and the others show the results in the home environment with different configurations. For any scenario, the time cost and the number of node are provided. In each figure, we use a box plot to describe the distribution of the experimental results.

sample many nodes in the exploration process, some of which make up the initial solution in the exploitation process. However, many nodes contribute a little or nothing to the solution. It is a key issue to balance the exploration and the exploitation. In order to explain this issue more clearly, we define a new evaluation metric, **Effective Node Rate**, notated as *ENR*.

Definition 4 (Effective Node Rate) When an initial solution is found, suppose the tree contains M nodes in total and the initial solution is composed of m nodes, the **ENR** is defined as

$$ENR = \frac{m}{M} \tag{15}$$

The *ENR* can reflect the utilization of nodes sampled in the exploration process. If a large number of nodes are sampled and the initial solution only contains a few nodes, we can say that the utilization is very low. Herein, we use the *ENR* to measure the balance between the exploration and the exploitation. Obviously, a large value of the *ENR* means a good balance between the exploration and the exploitation.

We validate our algorithms incorporating with the RRT, the RRT* and the Informed RRT* methods in simulation environments. First, we compare our algorithm with the RRT and the RRT* on finding the initial solution in terms of the time cost, the memory usage and the *ENR*. Second, we apply our algorithm into the Informed RRT* to see the influence of our proposed algorithm in convergence rate of finding the optimal solution.

In addition, we also test our algorithm in maze environments where a feasible solution is in a narrow aisle and with some complex turns. The experiments in the dynamic environment are also provided.

5.1. Finding the initial solution

In each experiment, we create a map with the size of 800 × 800 pixels. We run each path planning problem 50 times and use box plot (including maximum, the first quartile, median, the third quartile and minimum) to graphically depict the distribution of experimental data.

The RRT* and the Informed RRT* both use the same sampling method to find an initial solution. This sampling method may cost much time and memory usage. Our algorithm is also able to find the initial solution if it exists and can reduce the time cost and the memory usage using the probabilistic sampling.

Figure 4 shows the experiment results on finding the initial solution, where the RRTJ and the RRT*J mean that our algorithm, as a meta-algorithm, is applied to the RRT and the RRT*,

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

Table I. *ENR* in four different scenarios.

	RRT	**RRTJ**	**RRT***	**RRT*J**
$A \rightarrow B$	0.030	**0.041**	0.013	**0.019**
$C \rightarrow D$	0.044	**0.107**	0.017	**0.047**
$E \rightarrow F$	0.066	**0.126**	0.022	**0.049**

respectively. In home environment (Scenarios $A \rightarrow B$, $C \rightarrow D$ and $E \rightarrow F$), incorporating with our algorithm can save 34% and 51% in the time cost, and 49% and 54% in the memory usage on average compared with the RRT and the RRT*. Here the memory usage is equivalent to the number of nodes because two pre-processing steps, **GriddingProcess(***map***)** and **CalGrid(**Σ**)**, only use a very small number of memory at the beginning and other steps in our algorithm do not use the memory any more. In addition, in the box plot, the box of the RRTJ and the RRT*J is much smaller than that of the RRT and the RRT*, especially in Scenarios $C \rightarrow D$ and $E \rightarrow F$, which means the results from the RRTJ and the RRT*J have smaller standard deviation and good stability.

The key of our algorithm is that the algorithm can automatically choose the most potentially beneficial space for the tree's growth. The "potential" is measured by the probability. Because in our algorithm, the search process first will focus on the straight line linking x_{init} and x_{goal}. When the growth of the tree is limited by the obstacles between x_{init} and x_{goal}, the probabilistic sampling space will be expanded in the horizontal direction to bypass the obstacles. And in the vertical direction, the probabilistic sampling space moves with the treetop. Compared to the global sampling of the RRT*, the probabilistic sampling method can avoid many invalid nodes and shrink the search space, so it can find a high-quality path using less time cost and less memory usage. Even for some complicated environments, the probabilistic sampling method can expand the sampling space to the whole map gradually if the initial solution is hard to be found. The length of the initial solution obtained by two algorithms is similar because we do a tradeoff between the exploration and the exploitation. If the probabilistic sampling space expands slowly, the exploration is limited and the probabilistic completeness is not guaranteed at the beginning stage. So we want to speed the exploration and make the most of explored space in the exploitation process at the same time. As a result, the sampling space containing the initial solution is overlarge and the length of the initial solution obtained by our algorithm is similar to that of the RRT*.

As for Scenario $A \rightarrow B$, the performances of the RRTJ and the RRT*J are not improved significantly compared with the RRT and the RRT* in the time cost and the memory usage. Because Scenario $A \rightarrow B$ is challenging for our algorithm, where the tree needs a backward growth when approaching the goal region. However, the probabilistic sampling provides forward and backward sampling in the vertical direction and the results are still better than that of the RRT and the RRT*. In our future work, we intend to develop some methods for heuristically biasing the RRT growth[9] to further improve the performance of our algorithm in such challenging environments.

Table I shows the *ENR* of the RRT, the RRTJ, the RRT* and the RRT*J in three scenarios. In each scenario, the RRTJ and the RRT*J have a higher *ENR* than the RRT and the RRT*. As we all know, the RRT and the RRT* may sample many unnecessary nodes due to their randomness, and these nodes contribute a little to the initial solution and take much computational resource and memory usage. So a high *ENR* indicates that the algorithm maintains a better balance between the exploration and the exploitation, and more information from the exploration is utilized by the exploitation. Under this condition, the algorithm's performance is improved naturally. In other words, the high *ENR* of the RRTJ and the RRT*J proves that our probabilistic sampling method is effective in balancing the exploration and the exploitation, which are two fundamental work in path planning algorithms.

As mentioned before, we also test our algorithm in maze environments, as shown in Fig. 1. In such maze environments, an initial solution is hard to be found in that it needs to pass through the environment's border, grow backward and make some complex turns.

We first compare the RRTJ with the RRT, and two examples are shown in Fig. 1(a) and (b). Without any rewiring process, the initial solutions obtained by the RRT and the RRTJ, red line in Fig. 1(a) and (b) linking the start position and the goal region, both seem far from optimal. The statistical experimental results in Fig. 4(d) and (h) show that the RRTJ can save 31% in the time cost and 39% in the memory usage compared with the RRT. By observing the data distribution,

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

Table II. The experimental results of the IRRT* and the IRRT*J.

Scenario	Planner	Time (s)	Node	Length
$A \rightarrow B$	IRRT*	0.890 ± 0.231	5483 ± 1041	986 ± 12
	IRRT*J	$\mathbf{0.700 \pm 0.223}$	$\mathbf{4718 \pm 1174}$	$\mathbf{990 \pm 9}$
$C \rightarrow D$	IRRT*	0.521 ± 0.157	3486 ± 695	795 ± 3
	IRRT*J	$\mathbf{0.314 \pm 0.067}$	$\mathbf{2208 \pm 281}$	$\mathbf{795 \pm 4}$
$E \rightarrow F$	IRRT*	0.425 ± 0.080	2958 ± 554	710 ± 8
	IRRT*J	$\mathbf{0.378 \pm 0.165}$	$\mathbf{2475 \pm 1015}$	$\mathbf{701 \pm 16}$

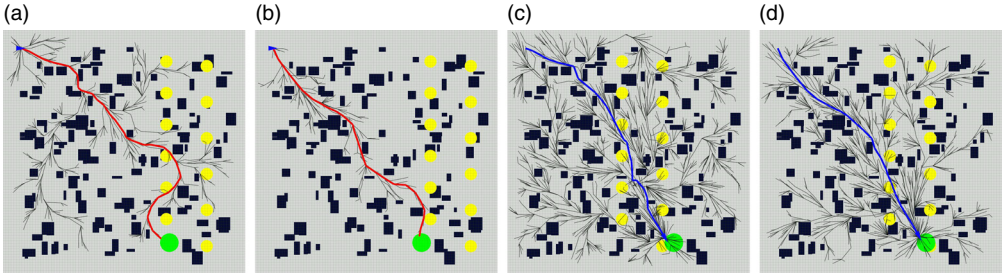


Fig. 5. Demonstration of RT-RRT* and RT-RRT*J. The triangle in the left top denotes a holonomic robot, the big green disk denotes the goal region and the small yellow disk denotes dynamic obstacle moving from right to left with the velocity $v = 2$ pixel/frame. The red and blue path represent the initial path founded by the path planner and the final path which the robot goes through, respectively.

we see that the mean value in the number of node of the RRTJ is much smaller than that of the RRT. Probabilistic sampling focuses on the "potential" region containing a feasible solution and this advantage is more obvious in complex environments like the maze environment. In fact, after so many iterations (14,122 iterations for the RRTJ and 16,237 iterations for the RRT on average), the area used for **CoverageDetection**(μ, Σ) is nearly equal to the whole map and the execution speed of our algorithm is still fast. It proves that our algorithm scales well even in complex environments.

Secondly, we compare the RRT* with the RRT*J, and two examples are shown in Fig. 1(c) and (d). The biggest difference between the RRT* and the RRT is that the RRT* has a rewiring process, and we see the initial solutions in Fig. 1(c) and (d) from the RRT* and the RRT*J are better than that from the RRT and the RRTJ shown in Fig. 1(a) and (b). However, the time cost of the RRT* and the RRT*J is higher, as shown in Fig. 4(d). As illustrated in Fig. 4(d) and (h), compared with the RRT*, the RRT*J can save 57% in the time cost and 45% in the memory usage. This improvement is larger than the improvement from the RRT to the RRTJ. It is worth mentioning that the standard deviations of the RRT*J both in the time cost and in the memory usage are much smaller than that of the RRT*. This is because the probabilistic sampling of our algorithm does not change too much for the same map and the results of different experiments are close. It further proves that our algorithm can help those algorithms with the property of asymptotic optimality improve much better. To test the generality, we apply our algorithm to another state-of-art algorithm, the Informed RRT*, to see the performance in the following section.

*5.2. Applied to the informed RRT**

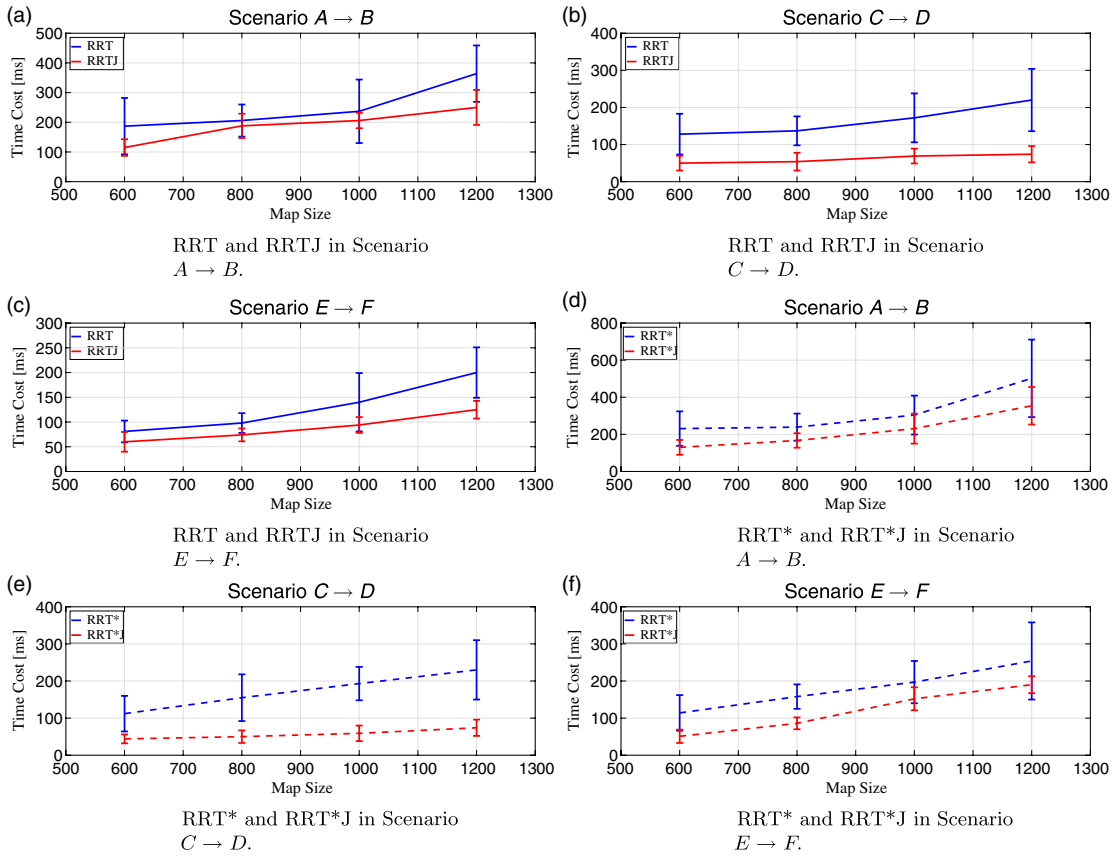
In each experiment, we create a map with the size of 800×800 pixels. We run each path planning problem 50 times and use a data table to show the experimental results.

As a meta-algorithm to find a high-quality initial path, our algorithm can be embedded into the post-processing methods like the Informed RRT*. Table II shows the experimental results of the Informed RRT* (notated as IRRT*) and the Informed RRT* incorporating our algorithm (notated as IRRT*J). In the home environment, the IRRT*J can save 24% in the time cost and 22% in the memory usage on average.

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

Table III. The experimental results of the RT-RRT* and the RT-RRT*J.

Scenario	Planner	Time (s)	Node	Length
	RT-RRT*	20226 ± 932	2776 ± 219	932 ± 21
	RT-RRT*J	$\mathbf{14358 \pm 1476}$	$\mathbf{2153 \pm 111}$	$\mathbf{903 \pm 7}$



RRT and RRTJ in Scenario $A \rightarrow B$.

RRT and RRTJ in Scenario $C \rightarrow D$.

RRT and RRTJ in Scenario $E \rightarrow F$.

RRT* and RRT*J in Scenario $A \rightarrow B$.

RRT* and RRT*J in Scenario $C \rightarrow D$.

RRT* and RRT*J in Scenario $E \rightarrow F$.

Fig. 6. Time cost on finding the initial solution in home environments with four different sizes. The RRTJ and the RRT*J represent improved RRT and RRT* with our meta-algorithm. In each figure, the time cost varies with the size of map, and the mean value and the standard deviation of corresponding statistics data are provided.

Through these experiments, we see that a high-quality initial solution has a big impact on the final optimal solution. Using our algorithm to search the initial solution, the planner has a priority to search the potential area that contains the solutions. Due to a focused exploration in these potential areas, the following exploitation process can be improved intuitively because it pays more attention to the aforementioned potential areas instead of treating any place of the map equally.

5.3. Simulation experiments in the dynamic environment

In the dynamic environment, a famous variant of the RRTs algorithm is Real-time RRT* (RT-RRT*).[31] The RT-RRT* uses the RRT* to find the initial solution. On the basis of the RT-RRT*, the RT-RRT*J uses our proposed algorithm to find the initial solution and other settings are the same with the RT-RRT*. Then, we carry on two sets of experiments with respect to moving obstacles in the dynamic environment to evaluate the performance of our algorithm. For each algorithm, 50 trials are conducted.

The map size is 800×800 pixels. The maximum velocity of the robot is 3 pixel/frame. As shown in Fig. 5, when the robot begins to move, its position changes and the path planner needs to update the path simultaneously. We use the time cost, the number of node and final path length as the metric to evaluate the performance of each algorithm, as shown in Table III.

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

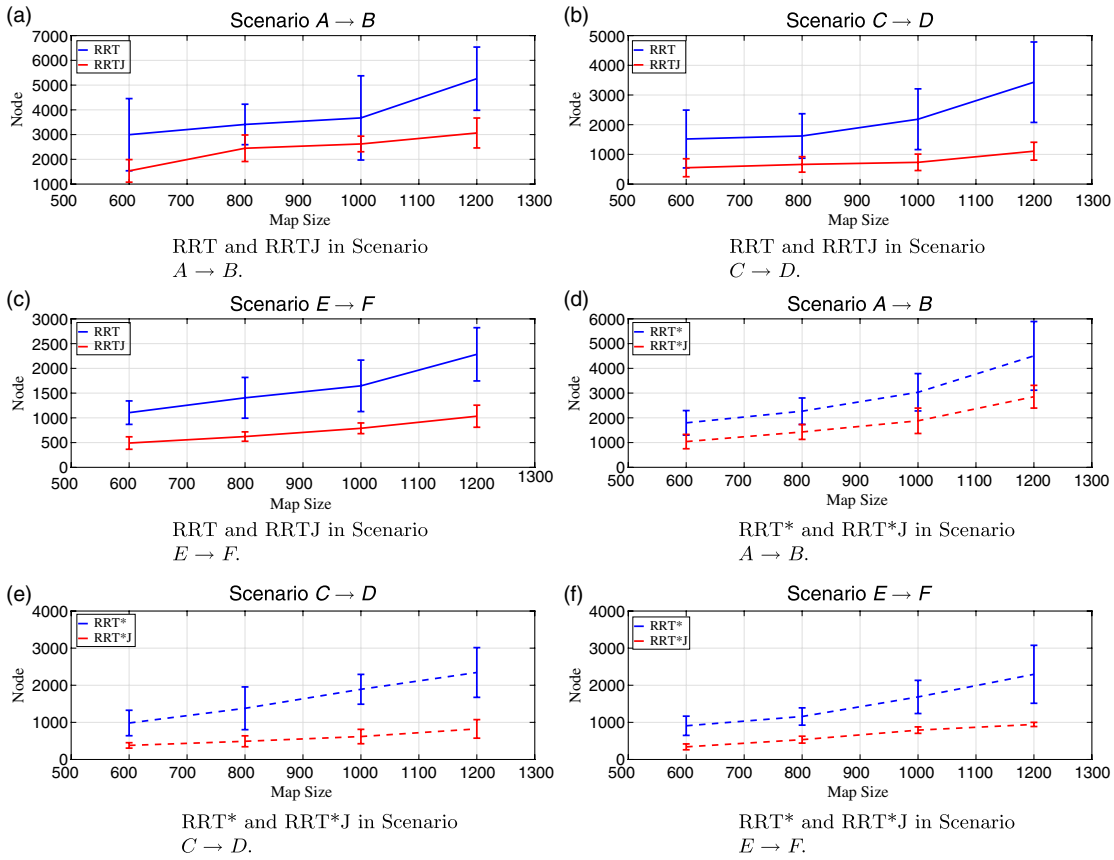


Fig. 7. The number of node on finding the initial solution in home environments with four different sizes. The RRTJ and the RRT*J represent improved RRT and RRT* with our meta-algorithm. In each figure, the number of node varies with the size of map, and the mean value and the standard deviation of corresponding statistics data are provided.

In the time cost, the number of node and final path length, the RT-RRT*J has a significant improvement compared with the RT-RRT*. Meanwhile, the RT-RRT*J has a smaller standard deviation, which shows the stability of the RT-RRT*J. Therefore, we can say that the RT-RRT*J achieves a better performance in the dynamic environment than the RT-RRT*.

5.4. Discussion

To further analyze the performance on finding the initial solution of our algorithm and test its scalability and generality, we implement our algorithm in the home environments of four different sizes (600×600, 800×800, 1000×1000 and 1200×1200 pixels). In each scenario, the algorithm is repeatedly implemented for 50 times. In the home environment, three scenarios represent three different path planning conditions.

For the experimental settings in the home environments, the size of the obstacle scales with the size of the map, while other parameters, such as the step size in **Steer**($x_{nearest}$, x_{rand}), the rewire radius in rewire process and so on, remain unchanged. We apply our algorithm to the RRT and the RRT* to see the performance including the time cost, the memory usage and the *ENR*.

Figures 6 and 7 show the experimental results in the four maps with different sizes. In general, first, the time cost and the number of node increase gradually as the map size increases. It is easily understood for the increment of the computation on a larger map. Secondly, the time cost of the RRT is usually less than that of the RRT* while the number of the node of the RRT* is usually less than that of the RRT, since the RRT* uses **NearestNeighbours**($\mathcal{T}$, x_{new}) instead of a single nearest node in each iteration and adds a rewiring process to optimize the whole tree. These two processes need extra time cost and cut off many unnecessary nodes.

In Fig. 6, we see that the time cost of the RRTJ and the RRT*J is always less than that of the RRT and the RRT*, especially in Scenario $C \rightarrow D$ shown in Fig. 6(b) and (e), and the standard deviation

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

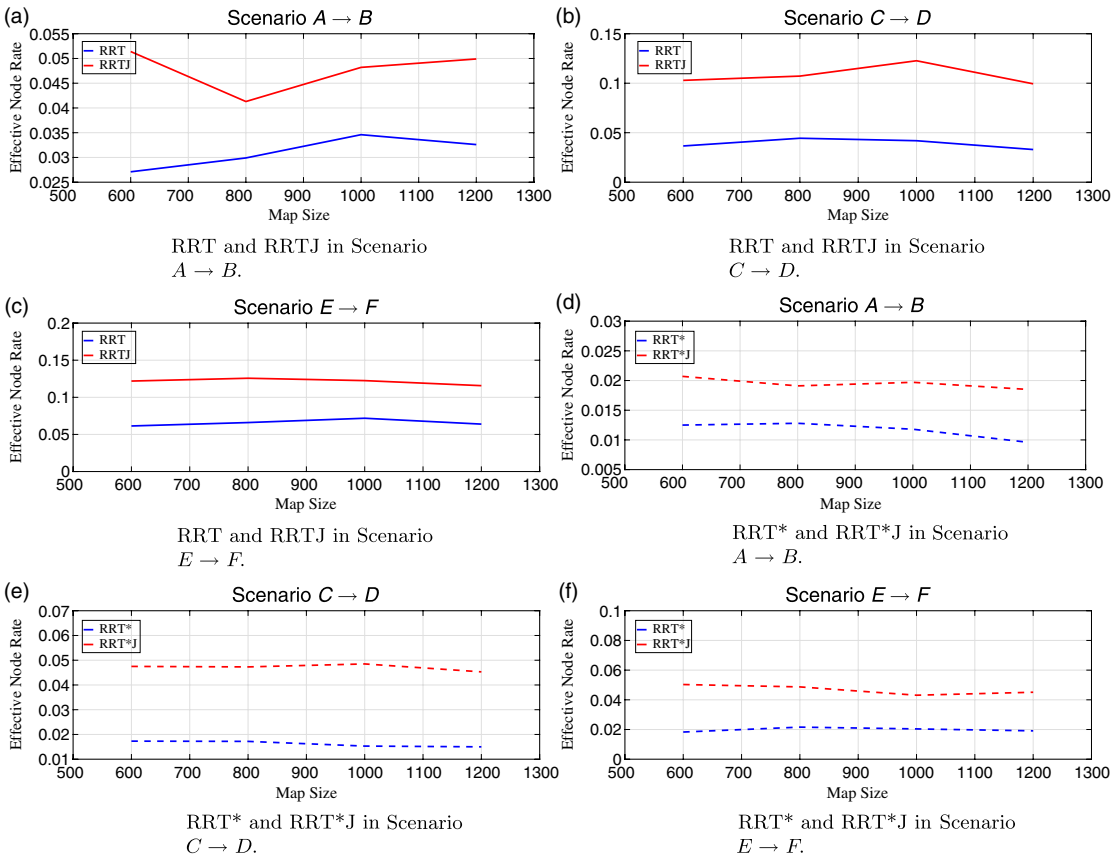


Fig. 8. *ENR* on finding the initial solution in home environments with four different sizes. The RRTJ and the RRT*J represent improved RRT and RRT* with our meta-algorithm. In each figure, the *ENR* varies with the size of map, and the mean value of corresponding statistics data is provided.

of the RRTJ and the RRT*J is also much smaller in most cases. Further more, as we all know, the RRT and the RRT* scale well with the map size. By comparison, the RRTJ and the RRT*J also scale well with the map size, which means that the execution speed of our algorithm does not decrease too much as the map size increases gradually.

In Fig. 7, the number of node used by the RRTJ and the RRT*J is obviously smaller than that of the RRT and the RRT*, and the standard deviation of the RRTJ and the RRT*J is still smaller. Compared to the RRT and the RRT*, the memory usage of the RRTJ and the RRT*J increases more smoothly and seems less sensitive to the map size.

From the experimental results in different map sizes, we see that the time cost and the number of nodes both scale well with the map size. This shows that our algorithm can perform better than the RRT and the RRT* and scale well with the map size when finding the initial solution under different scenarios. In short, our algorithm has good scalability and generality.

Figure 8 shows the *ENR* of the RRT, the RRTJ, the RRT* and the RRT*J in three scenarios. In each scenario, RRTJ and RRT*J have a higher *ENR* than the RRT and the RRT*, which means that our algorithm achieves a better balance between the exploration and the exploitation. As the map size increases, the *ENR* of the RRTJ and the RRT*J changes a little. Again it shows that our algorithm has good scalability and generality. Due to the good balance between the exploration and the exploitation, our algorithm obtains an improved performance in the time cost, the memory usage and the *ENR*.

6. Conclusions and Future Work

In this paper, a bioinspired path planning algorithm is proposed to find a high-quality initial solution for the RRTs algorithm on the basis of the RRT method. We modify the sampling process through controlling the sampling space and using the probabilistic sampling with the two-dimensional Gaussian mixture model. Using the Gaussian mixture model to imitate the trees' growth, we can

https://doi.org/10.1017/S0263574719000195 Published online by Cambridge University Press

quickly obtain an initial path with a small number of nodes. We apply our algorithm to two fundamental sampling-based algorithms, the RRT and the RRT*, to see the improvement. We test our algorithm in different environments, and the experimental results show that our algorithm brings a significant improvement in the time cost, the memory usage and the *ENR*. In addition, we implement our algorithm on the same map with different sizes. The results show that our algorithm scales and generalizes well. As a meta-algorithm, we apply our algorithm to the Informed-RRT*, and the results show that the convergence rate is promoted further and much memory usage is saved.

A limitation of our algorithm is that it only applies to 2D environments. In our future work, we will improve our algorithm to apply 3D or high-dimensional environments. We have verified that our algorithm scales well in 2D environments (the coverage detection and the probabilistic sampling are little affected by the problem size), and its variant is also very promising to be applied well in high-dimensional environments. In fact, the sampling-based path planning algorithms compose of two primitive operations: the collision detection (CD) and the NN search.[32] The CD[33] and the NN[34] are also considered as the main bottleneck. To an extent, our algorithm improves the NN search by controlling the sampling space and using the probabilistic sampling. In our future work, we will work on the further adjustment on the NN to improve the sampling-based algorithms.

Acknowledgement

This project is partially supported by the Hong Kong ITC ITSP Tier 2 grant #ITS/105/18FP and Hong Kong RGC GRF grant #14200618.

References

1. A. R. Willms and S. X. Yang, "An efficient dynamic system for real-time robot-path planning," *IEEE Trans. Syst. Man Cybern. B Cybern.* **36**(4), 755–766 (2006).
2. J. Yang, Z. Qu, J. Wang and K. Conrad, "Comparison of optimal solutions to real-time path planning for a mobile vehicle," *IEEE Trans. Syst. Man Cybern. A Syst. Hum.* **40**(4), 721–731 (2010).
3. H. Li, S. X. Yang and M. L. Seto, "Neural-network-based path planning for a multirobot system with moving obstacles," *IEEE Trans. Syst. Man Cybern. C Appl. Rev.* **39**(4), 410–419 (2009).
4. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Rob. Res.* **5**(1), 90–98 (1986).
5. P. E. Hart, N. J. Nilsson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.* **4**(2), 100–107 (1968).
6. A. Stentz, "Optimal and Efficient Path Planning for Partially-known Environments," *Proceedings of the IEEE International Conference on Robotics and Automation, 1994*, IEEE, San Diego, CA, USA (1994) pp. 3310–3317.
7. S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning," *Int. J. Rob. Res.* **20**(5), 378–400 (2001).
8. L. E. Kavraki, P. Svestka, J.-C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.* **12**(4), 566–580 (1996).
9. C. Urmson and R. Simmons, "Approaches for Heuristically Biasing RRT Growth," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003*, IROS 2003, Las Vegas, NV, USA vol. 2, IEEE (2003) pp. 1178–1183.
10. A. Shkolnik, M. Walter and R. Tedrake, "Reachability-guided Sampling for Planning Under Differential Constraints," *IEEE International Conference on Robotics and Automation, 2009*, ICRA'09, IEEE, Kobe, Japan (2009) pp. 2859–2865.
11. J. Wang, X. Li and M. Q.-H. Meng, "An Improved RRT Algorithm Incorporating Obstacle Boundary Information," *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, IEEE, Qingdao, China (2016) pp. 625–630.
12. S. Kiesel, E. Burns and W. Ruml, "Abstraction-guided Sampling for Motion Planning," Proceedings of the Fifth Annual Symposium on Combinatorial Search, *SoCS*, Ontario, Canada (2012).
13. N. A. Wedge and M. S. Branicky, "On Heavy-tailed Runtimes and Restarts in Rapidly-exploring Random Trees," *Twenty-Third AAAI Conference on Artificial Intelligence*, Chicago, IL, USA (2008) pp. 127–133.
14. J. D. Gammell, S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, (IROS 2014), IEEE, Chicago, IL, USA (2014) pp. 2997–3004.
15. R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *Int. J. Rob. Res.* **26**(8), 845–863 (2007).
16. M. Brunner, B. Brüggemann and D. Schulz, "Hierarchical Rough Terrain Motion Planning Using an Optimal Sampling-based Method," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Karlsruhe, Germany (2013) pp. 5539–5544.

17. L. Palmieri, S. Koenig and K. O. Arras, "RRT-based Nonholonomic Motion Planning Using Any-angle Path Biasing," *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Stockholm, Sweden (2016) pp. 2775–2781.
18. B. Raveh, A. Enosh and D. Halperin, "A little more, a lot better: Improving path quality by a path-merging algorithm," *IEEE Trans. Rob.* **27**(2), 365–371 (2011).
19. O. Arslan and P. Tsiotras, "Use of Relaxation Methods in Sampling-based Algorithms for Optimal Motion Planning," *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Karlsruhe, Germany (2013) pp. 2421–2428.
20. M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans Evol Comput* **1**(1), 53–66 (1997).
21. J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gucht, "Genetic Algorithms for the Traveling Salesman Problem," *Proceedings of the first International Conference on Genetic Algorithms and their Applications* Pittsburgh, PA, USA (1985) pp. 160–168.
22. E. Ferrante, A. E. Turgut, C. Huepe, A. Stranieri, C. Pinciroli and M. Dorigo, "Self-organized flocking with a mobile robot swarm: A novel motion control method," *Adapt. Behav.* **20**(6), 460–477 (2012).
23. M. S. Güzel, M. Kara and M. S. Beyazkılıç, "An adaptive framework for mobile robot navigation," *Adapt. Behav.* **25**(1), 30–39 (2017).
24. T. Dean, K. Basye and J. Shewchuk, "Reinforcement Learning for Planning and Control," **In**: *Machine Learning Methods for Planning*, S. Minton (ed.) (Elsevier, San Francisco, CA, USA, 1993) pp. 67–92.
25. H. Kretzschmar, M. Spies, C. Sprunk and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *Int. J. Rob. Res.* **35**(11), 1289–1307 (2016).
26. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Rob. Res.* **30**(7), 846–894 (2011).
27. L. Janson, E. Schmerling, A. Clark and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *Int. J. Rob. Res.* **34**(7), 883–921 (2015).
28. Y. Li, Z. Littlefield and K. E. Bekris, "Sparse Methods for Efficient Asymptotically Optimal Kinodynamic Planning," **In**: *Algorithmic Foundations of Robotics XI* H. L. Akin, N. M. Amato, V. Isler, A. F. van der Stappen (eds.) (Springer, Istanbul, Turkey, 2015) pp. 263–282.
29. O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality motion planning," *IEEE Trans. Rob.* **32**(3), 473–483 (2016).
30. D. Ferguson and A. Stentz, "Anytime RRTs," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Beijing, China (2006) pp. 5369–5375.
31. K. Naderi, J. Rajamäki and P. Hämäläinen, "RT-RRT*: A Real-time Path Planning Algorithm Based on RRT," *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, ACM, New York, NY, USA (2015), pp. 113–118.
32. M. Kleinbort, O. Salzman and D. Halperin, "Efficient High-quality Motion Planning by Fast All-pairs r-nearest-neighbors," *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, Seattle, WA, USA (2015) pp. 2985–2990.
33. S. M. LaValle, *Planning Algorithms* (Cambridge University Press, New York, NY, USA, 2006).
34. J. Bialkowski, S. Karaman, M. Otte and E. Frazzoli, "Efficient Collision Checking in Sampling-based Motion Planning," **In**: *Algorithmic Foundations of Robotics X* (Springer, Boston, MA, USA, 2013) pp. 365–380.