# Sampling-based algorithms for optimal motion planning

Md Rayhanul Islam
Ibne Farabi Shihab

May 5, 2023

## 1 Introduction

The path-planning technique has gained significant momentum in robotics to plan the trajectories of robots navigating dynamic and uncertain environments. This technique finds valid configurations capable of guiding a robot from its initial position to a predetermined goal, provided that the robot possesses a comprehensive description of the environment, dynamics, and the initial and goal configurations. However, finding a set of configurations from initial to goal configurations is computationally expensive. Even a relatively simple problem, such as the piano movers problem, has been deemed PSPACE-hard [Rei79]. Furthermore, the very initial works in motion planning, such as [KB+91, BLP85], highly depend on constructing obstacle regions explicitly [LK01], which imposes an enormous computational burden when navigating the vast expanse of high-dimensional space.

The sampling-based motion planning algorithm comes out as a solution to alleviate the computational burden of constructing the obstacle region. The core principle of this approach depends on the random generation of points within the configuration space, subsequently connecting them via a graph, provided that the ensuing path remains unobstructed by obstacles. Two prominent sampling-based motion planning algorithms, $RRT$ and $PRM$, have risen to prominence. While both algorithms share the commonality of generating random points within the obstacle-free configuration space, their methods of connecting these points and forming graphs diverge. The $PRM$ algorithm, for instance, assembles collision-free trajectories and traces a path from the initial configuration to the goal configuration by navigating the graph, known as the roadmap. On the other hand, the $RRT$ algorithm employs an incremental approach expanding a tree structure from an initial configuration towards a goal configuration by integrating randomly generated samples into the tree while avoiding obstacles [KL00].

Although both $RRT$ and $PRM$ save computational resources by avoiding the construction of the obstacle region, they are not asymptotically optimal. Apart from this, RRT is highly dependent on sampling, and biased sampling toward certain regions of the configuration space can lead to longer planning times. In addition, PRM requires more memory as it maintains a roadmap. To solve this problem, [KF11] proposes a variant of $RRT$ and $PRM$, named $RRT^*$ and $PRM^*$ respectively, which is probabilistically complete, asymptotically optimal and computationally efficient. This study implements the $RRT^*$ and $PRM^*$ and compares results with existing $RRT$ and $PRM$ along with complexity analysis followed by discussion.

### 1.1 Problem configurations and assumptions

In addressing motion planning challenges, configuration spaces represent all possible transformations applicable to a robot without encountering obstacles. Unlike discrete planning, actions remain implicit as sampling-based motion planning algorithms randomly generate points within the configuration space, connecting them through a tree or graph. The concept of time is continuous, as the robot's motion is continuous. Furthermore, it is assumed that there are no sudden changes in the robots' acceleration. Equipped with sensors capable of accurately detecting static obstacles, the robot navigates an environment where the obstacle regions persist unchanged over time.

### 1.2 Problem formulation

Consider $X = (0, 1)^d$ be the configuration space with dimension d=2. Let $X_{obs}$ be the obstacle region, and $X_{free} = X \setminus X_{obs}$ be the obstacle-free region. The initial condition $x_{init} \in X_{free}$ and the goal condition $X_{goal} \in X_{free}$ as only one goal is considered here, however, $X_{goal} \subset X_{free}$ if there exist multiple goal configurations. A path planning problem is defined as a triplet $(X_{free}, x_{init}, X_{goal})$. A path is a continuous function, $\tau : [0, 1] \to X$. A path is a collision-free path if $\tau(i) \in X_{free}$, for all $i \in [0, 1]$. A path is a feasible path if it is a collision-free path

along with $\tau(0) = x_{init}$, and $\tau(1) = X_{goal}$. Let $\Sigma$ is the set of all paths, and $c : \Sigma \to \mathbb{R}_{\geq 0}$ be the cost function to assign a strictly positive cost to all collision-free paths. For any two paths, $\tau_1, \tau_2 \in \Sigma$ such that $\tau_1(1) = \tau_2(0)$, then the concatenation of $\tau_1$, and $\tau_2$ is defined as $\tau' : \tau_1 \frown \tau_2$ such that $\tau'(0) = \tau_1(0)$ and $\tau'(1) = \tau_2(1)$ and $\tau' \in \Sigma$. The set of all paths $\Sigma$ is closed under concatenation.

**Problem:** Given a path planning problem, $(X_{free}, x_{init}, X_{goal})$, and a cost function, $c : \Sigma \to \mathbb{R}_{\geq 0}$, find an optimum feasible path, $\tau^\star$ such that $c(\tau^\star) = Inf\{c(\tau) : \tau \text{ is feasible}\}$. If no such path exists, report a failure.

# 2 Solution Approaches

## 2.1 $PRM^*$

Unlike the PRM algorithm, which considers the fixed length radius for computing the neighbors of a new vertex in the roadmap. The proposed dynamic radius based $PRM^\star$ considers radius as the function of number of samples.

If radius, $r$ and the number of samples, n, then $r(n) = \gamma_{PRM} * (\frac{log(n)}{n})^{\frac{1}{n}}$, where

- $\gamma_{PRM} > \gamma^\star_{PRM} = 2(1 + \frac{1}{d})^{\frac{1}{d}} * \frac{\mu(X_{free})}{\zeta_d}$.
- d be the dimension of the c-space.
- $\mu(X_{free})$ be the Lebesgue measure of the obstacle-free space.
- $\zeta_d$ is the unit ball in the d-dimensional space.

Another variation of $PRM^*$ uses the k-nearest neighbors value as $k(n) = k_{PRM} * log(n)$, where $k_{PRM} = 2e$ is the valid choice, and e represents a natural logarithmic constant [KF11].

*The main difference between PRM, and $PRM^*$ algorithms is that $PRM^*$ connects to all vertices residing within the radius value or k-nearest neighbors.* The proposed $PRM^*$ is provided in Algorithm 1.

**Algorithm 1** $PRM^*$

**Require:** $n \geq 0$
1: $V \leftarrow x_init \cup \{SampleFree_i\}_{i=1,\ldots,n}; E \leftarrow \{\}$
2: **for each** $v \in V$ **do do**
3: $U \leftarrow Near(G = (V, E), v, \gamma_{PRM}(\frac{log(n)}{n})^{\frac{1}{n}}) \setminus \{v\}$
4: **for each** $u \in U$ **do do**
5: **if** CollisionFree(v,u) **then**
6: $E \leftarrow E \cup (v, u), (u, v)$
7: **end if**
8: **end for**
9: **end for**
10: **return** G=(V,E)

## 2.2 $RRT^*$

The $RRT^*$ is deemed as the extension of the RRG algorithm (The detail of RRG is available in [KF11]). *Unlike RRG, the $RRT^*$ algorithm removes the redundant edges not part of the shortest path.*

The $RRT^*$ algorithm can use the radius, or k-nearest neighbors approach to compute the neighbors of a particular new vertex. The dynamic radius based $RRT^*$ uses $r(card(V)) = min(\gamma_{RRT} * (\frac{log(card(V)}{card(V)})^{\frac{1}{n}}, \eta)$ to compute radius, where $\eta$ is predefined value, and $\gamma_{RRT} = 2(1 + \frac{1}{d})^{\frac{1}{d}} * \frac{\mu(X_{free})}{\zeta_d}$ (Explanation available in $PRM^*$). The k-nearest neighbors based $RRT^*$ uses $k(card(V)) = k_{RRG} * log(card(V))$, and $k_{RRG} = 2e$ is the valid choice.

*The $RRT^*$ only adds a new edge to the tree if it satisfies the following two properties-*

- An edge is created from a vertex in $X_{near}$ to $x_{new}$ with minimum cost.
- New edges are created from $x_{new}$ to a vertex in $X_{near}$ if the path through $x_{new}$ has a lower cost than the path through the existing parent of that vertex.

The detailed description of $RRT^*$ is provided in Algorithm 2.

**Algorithm 2** RRT* Algorithm

**procedure** RRT*$(q_I, q_G, n, \gamma, \eta)$
  $V \leftarrow \{q_I\};\ E \leftarrow \emptyset$
  **for** $i = 1$ to $n$ **do**
    $q_{rand} \leftarrow \text{SampleFree}_i$
    $q_{nearest} \leftarrow \text{Nearest}(G = (V, E), q_{rand})$
    $q_{new} \leftarrow \text{Steer}(q_{nearest}, q_{rand})$
    **if** ObstacleFree$(q_{nearest}, q_{new})$ **then**
      $X_{near} \leftarrow \text{Near}(G = (V, E), q_{new}, \min\{\gamma\text{RRT*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$
      $V \leftarrow V \cup \{q_{new}\}$
      $q_{min} \leftarrow q_{nearest};\ c_{min} \leftarrow \text{Cost}(q_{nearest}) + c(\text{Line}(q_{nearest}, q_{new}))$
      **for all** $q_{near} \in X_{near}$ **do**
        **if** CollisionFree$(q_{near}, q_{new}) \wedge \text{Cost}(q_{near}) + c(\text{Line}(q_{near}, q_{new})) < c_{min}$ **then**
          $q_{min} \leftarrow q_{near};\ c_{min} \leftarrow \text{Cost}(q_{near}) + c(\text{Line}(q_{near}, q_{new}))$
        **end if**
      **end for**
      $E \leftarrow E \cup \{(q_{min}, q_{new})\}$
      **for all** $q_{near} \in X_{near}$ **do**
        **if** CollisionFree$(q_{new}, q_{near}) \wedge \text{Cost}(q_{new}) + c(\text{Line}(q_{new}, q_{near})) < \text{Cost}(q_{near})$ **then**
          $q_{parent} \leftarrow \text{Parent}(q_{near})$
          $E \leftarrow (E \setminus \{(q_{parent}, q_{near})\}) \cup \{(q_{new}, q_{near})\}$
        **end if**
      **end for**
    **end if**
  **end for**
  **return** $G = (V, E)$
**end procedure**

# 3 Result and analysis

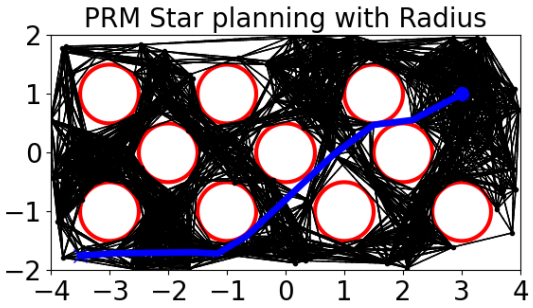
In terms of motion planning algorithms, the results and analysis section serves as a pivotal showcase, displaying the prowess of these algorithms within diverse motion planning scenarios. Delving into the experimental findings and comparative evaluations, alongside an asymptotic complexity analysis, we shed light on the implications within the motion planning sphere. Merging qualitative and quantitative observations, we unravel the intricate strengths and limitations of the aforementioned algorithms, offering a comprehensive understanding of their performance.

## 3.1 Experimental setup

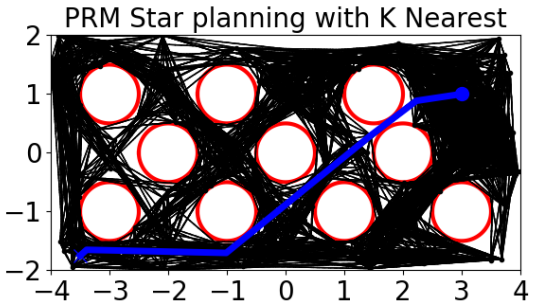
The implementation of $RRT^*$ and $PRM^*$ are based on the solution of homework 4. To implement these two algorithms, we consider the C-space $C = [-4, 4] * [-1, 1]$ and circle like obstacles are centered at position $(-1, -1)$, $(1, -1)$, $(2, 0)$, $(-2, 0)$, $(-3, 1)$, $(1.5, 1)$, $(0, 0)$, $(-1, 1)$, $(3, -1)$, $(-3, -1)$ with radius 0.5. The initial configuration $x_{init}$ = (-3.5,-1.75) and the goal configuration $X_{goal}$ =(3, 1) as in the Figure 1. The cost of each feasible path is measured as the summation of all paths that form the feasible path as described in the path concatenation in the problem formulation.

Both algorithms dynamically compute the radius and k-nearest neighbors value as the function of the number of samples, but $RRT^*$ uses another predefined value $\eta$, given by the user. In our consideration, we used the elbow technique to compute $\eta$, and after several runs, we found $RRT^*$ works well for $\eta = 1.5$. The experiment runs on the MacBook Pro with Processor 2 GHz Quad-Core Intel Core i5 and 16 GB RAM. The details of running each algorithm are available in *readme.md* of the project source folder.

A pictorial view of the implement algorithms with their variation is given in Figure 1 and Figure 2 for 200 samples.

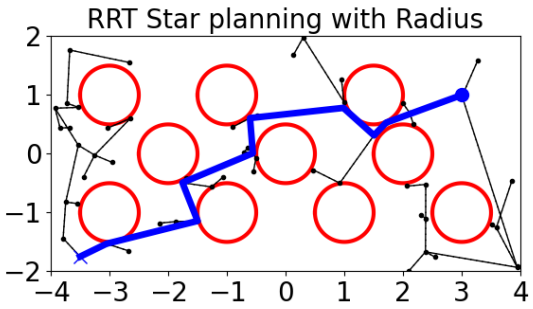



(a) radius value

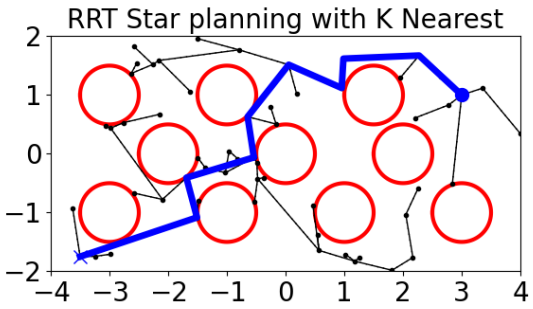



(b) k-nearest value

Figure 1: The $PRM^{\star}$ algorithm considering radius and k-nearest value as a function of sample




(a) radius value




(b) k-nearest value

Figure 2: The $RRT^{\star}$ algorithm considering radius and k-nearest value as a function of sample

| iteration | RRT | $RRT^*$(k) | $RRT^*$(r) |
|---|---|---|---|
| 100 | 41.45 | 49.55 | 47.70 |
| 200 | 70.90 | 105.50 | 116.50 |
| 300 | 45.60 | 185.90 | 185.70 |
| 400 | 75.10 | 244.95 | 258.90 |
| 500 | 69.90 | 329.25 | 330.90 |

Table 1: Number of iteration vs. number of avg vertices explored

### 3.2 Average vertices explored and Iteration wise analysis for RRT and $RRT^*$

In this section, a comprehensive comparison delves into the vertices explored by the RRT algorithm and its two variations—RRT* with radius ($RRT^*(r)$) and RRT* with k ($RRT^*(k)$). One important thing to notice is that vertices exploration does not work for PRM and its variations as they build the roadmap after exploring all the vertices, and the number of vertex explorations is higher in PRM compared to RRT. The primary objective of this analysis lies in offering a quantitative assessment of the efficacy of vertex exploration by RRT and the variations of $RRT^*$ in robotic motion planning.

Table 1 unveils the average number of vertices traversed by each algorithm at different iteration intervals, spanning from 100 to 500. It is crucial to note that the displayed vertex counts are averages derived from 20 separate iterations for each algorithm. Scrutinizing the data reveals that RRT consistently trumps $RRT^*(r)$ and $RRT^*(k)$, as it navigates through fewer vertices throughout all iteration counts. For instance, at 100 iterations, RRT explores 41.45 vertices on average, while $RRT^*(k)$ and $RRT^*(r)$ explore 49.55 and 47.70 vertices, respectively, because RRT quits exploration once it can connect goal but $RRT^*(k)$ and $RRT^*(r)$ continue to find the optimal goal. This trend is also evident at higher iteration counts, such as 400 iterations, where RRT explores 75.10 vertices compared to $RRT^*(k)$'s 244.95 and $RRT^*(r)$ 258.90. This outcome accentuates the superior efficiency of the RRT over $RRT^*$(r) and $RRT^*$(k) in the number of vertex explorations.

### 3.3 Impact of radius and K value

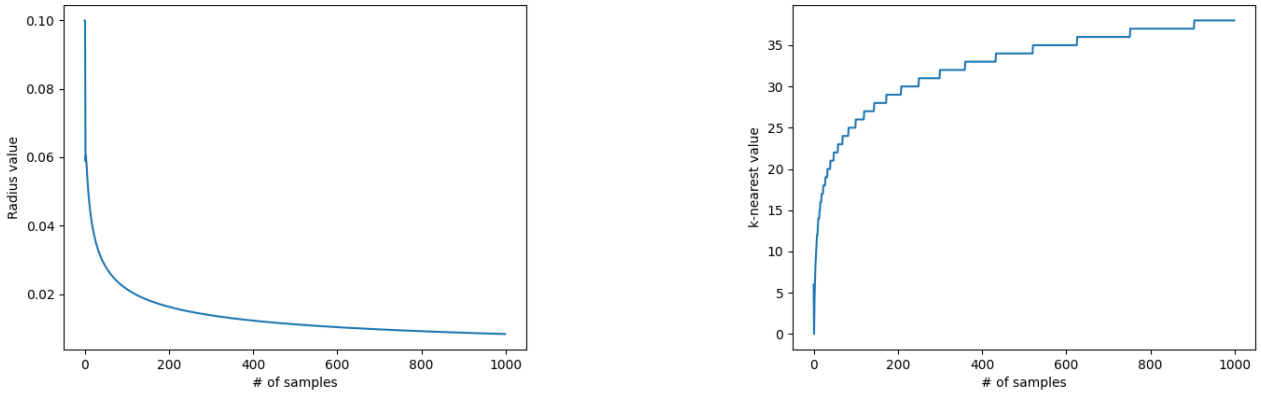
In this section, a meticulous comparison probes the influence of radius and k neighbor values in $RRT^*$ and $PRM^*$. The primary objective of this analysis resides in providing a quantitative assessment of the effects of radius and k neighbor value on each of the $RRT^*$ and $PRM^*$, taking into account the progressive decrease in radius and increase in k neighbor value over the number of samples.

As illustrated in Figure 3, it becomes clear that the radius gradually decreases as the sample number rises, while the k neighbor value demonstrates a contrary pattern, steadily increasing. The observed behavior can be attributed to the increased local connectivity as the number of samples rises, leading to a denser distribution of vertices in the search space. As the search space becomes densely populated, the effective radius for identifying neighboring vertices decreases, resulting in higher chances of discovering nearby vertices. Simultaneously, with the growth in vertex density, the algorithm requires a larger value for the k neighbor to accommodate the increased number of potential neighbors. This adaptation ensures the algorithm's sustained effectiveness in exploring the search space and finding feasible paths, despite the escalating number of samples.

### 3.4 Empirical Time and cost analysis

This detailed comparative analysis examines RRT and PRM, with the variations of $PRM^*$, and $RRT^*$, focusing on their average time and path cost (length). The results from Table 2 provide valuable insights along with the efficiency and effectiveness of each algorithm.

The comparison between PRM and different versions of $PRM^*$ depicts the superior performance of $PRM^*(r)$ in terms of cost with a path length 7.293700 units. However, PRM is faster than other versions of the $PRM^*$ algorithm, with an average time of 1.109426 seconds. On the contrary, RRT-based algorithms show extremely superior performance in average time, where the RRT stands out from other $RRT^*$ algorithms with an average time of 0.024845 seconds. Even though RRT stands out in time, it exhibits poor performance in terms of cost, where the cost of $RRT^*(k)$ and $RRT^*(r)$ are 9.277821 and 9.351986 units, respectively, while the cost of RRT is 9.905258 units. Edge adjustments contribute to the increased time consumption observed in both $RRT^*(k)$ and $RRT^*(r)$ algorithms.




(a) radius value

(b) k-nearest value

Figure 3: Changes of radius and k-nearest value over the different values of samples

| Algorithm Name | TIME(Secs) | COST(length) |
|---|---|---|
| PRM | 1.109426 | 15.026883 |
| $PRM^*$(k) | 3.931428 | 7.305814 |
| $PRM^*$(r) | 3.875976 | 7.293700 |
| RRT | 0.024845 | 9.905258 |
| $RRT^*$(k) | 0.762789 | 9.277821 |
| $RRT^*$(r) | 0.726062 | 9.351986 |

Table 2: Algorithm wise average time and average cost

In the PRM and $PRM^*$, roadmaps generate numerous random points throughout the configuration space, connecting pairs of neighboring points with obstacle-free paths. Upon completing the roadmap, a path between the start and goal configurations is identified by searching this graph. The PRM roadmap construction is computationally demanding, connecting each sampled point to its neighbors, resulting in a dense graph. The fundamental PRM algorithm crafts a dense roadmap without optimizing path length, whereas $PRM^*$ with radius and k neighbor refines path cost through local search strategies. Consequently, the path cost of $PRM^*$(r) drops to 7.293700 from PRM's 15.026883, albeit with an increase in computational time (from 1.109426 to 3.875976 seconds).

Conversely, the RRT-based algorithms gradually build a tree by extending it from the initial configuration towards randomly sampled points in the configuration space. Expansion ceases upon reaching the goal configuration or a predefined maximum number of iterations. The RRT's path construction is computationally efficient, expanding the tree without requiring connections to multiple neighbors for each point. This results in shorter construction times (e.g., 0.762789 seconds for $RRT^*(k)$ compared to 3.931428 seconds for $PRM^*(k)$). The RRT algorithm emphasizes rapid exploration by incrementally expanding a tree structure with randomly generated samples leading to swift computation times but sub-optimal path costs of 9.905258. In contrast, $RRT^*$ with radius and k neighbor refines the tree structure by rewiring nearby nodes to optimize path cost (improving from 9.905258 to 9.277821), increasing computational demands but enhancing path quality.

In summary, the $PRM^*$ and $RRT^*$ families of algorithms exhibit unique advantages and drawbacks, with PRM prioritizing cost-efficient paths and RRT emphasizing rapid exploration. The choice between them depends on the specific application requirements and desired trade-offs.

### 3.5 Obstacle region vs. radius

In this analysis, we explore the effects of rising obstacle counts on the dynamic radii of $RRT^*(r)$ and $PRM^*(r)$. The obstacle count progressively increases from 0 to 10, allowing us to observe the dynamic changes. To maintain consistency, each algorithm is run 1000 times for a specific obstacle number, eliminating the element of randomness. With the configuration space size, the value of eta is capped at 1.5, ensuring that the minimum portion of the $RRT(r)$ dynamic radius calculation never exceeds 1.5. Consequently, the dynamic radius formulas for $RRT(r)$ and

| Obstacle Count | PRM* Radius | RRT* Radius | k-nearest |
|---|---|---|---|
| 0 | 0.3998 | 0.3965 | 45.1254 |
| 1 | 0.3662 | 0.3637 | 45.1254 |
| 2 | 0.2635 | 0.2628 | 45.1254 |
| 3 | 0.0053 | 0.0056 | 45.1254 |
| 4 | 0.0053 | 0.0056 | 45.1254 |
| 5 | 0.0053 | 0.0056 | 45.1254 |
| 6 | 0.0053 | 0.0056 | 45.1254 |
| 7 | 0.0053 | 0.0056 | 45.1254 |
| 8 | 0.0053 | 0.0056 | 45.1254 |
| 9 | 0.0053 | 0.0056 | 45.1254 |
| 10 | 0.0053 | 0.0056 | 45.1254 |

Table 3: Comparison of $PRM^*(r)$, $RRT^*(r)$, and K-nearest with Increasing Obstacle Count

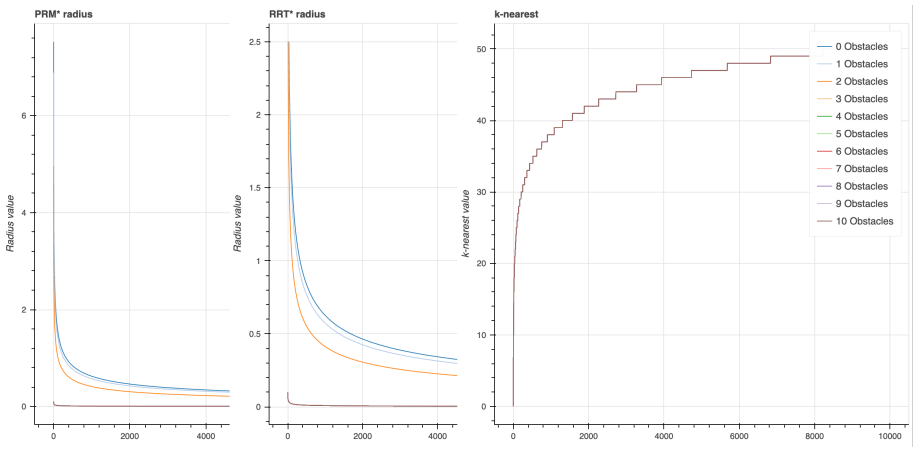



Figure 4: The divergence of radius value for different numbers of obstacles

$PRM^*(r)$ are identical. For simplicity, we will examine the dynamic radius changes for a single iteration as the number of vertices increases with ten obstacles. The figure demonstrates that the algorithm dynamically adjusts the radius accordingly as the number of vertices increases. This is because, as the space becomes denser, the algorithm must reduce its radius to locate the nearest points effectively. Table 1 corroborates this finding, as it shows that the number of vertices is significantly lower for the same number of iterations between $RRT(r)$ and RRT/$RRT(k)$. Utilizing a dynamic radius allows for exploring fewer vertices while calculating a probabilistically optimal path. Figure 3 displays how the radius changes for each obstacle, with radius values averaged over 1000 iterations. The average radius for obstacle numbers 7 and 10 exhibits considerable differences, while it remains consistent for other radii. Radii greater than 1.5 are deemed extraneous by the algorithm, although they are included in the figure for illustrative purposes. Interestingly, the average K-nearest neighbor count remains constant regardless of the obstacle count. However, when examining a specific iteration in Figure 4, it is apparent that the number of neighbors increases as the number of vertices/samples rises. Since we are averaging over 1000 iterations and there is minimal deviation, the average remains consistent, as demonstrated by our empirical analysis in Table 3.

## 3.6 Analysis

This section discusses the complexity analysis of $PRM^*$ and $RRT^*$ and lists some theorems from the original work to understand these algorithms succinctly.

### 3.6.1 Complexity Analysis

Lemma 42 in [KF11] shows the complexity of $RRT^*$ and $PRM^*$ is $O(nlog(n))$. The below section discusses the complexity analysis of the proposed algorithms.

$PRM^*$

The complexity analysis of $PRM^*$ is demonstrated below with the main steps contributing to the time complexity:

1. The initialization step (Line 2) takes $O(n)$ time to generate n samples.
2. The outer loop (Lines 3-10) iterates over every n+1 vertices in V.
3. In the inner loop (Lines 4-9), the Near function computes the set of nearby vertices U for each vertex v, with the number of nearby vertices being proportional to the logarithm of the number of vertices (log(n)). The computation of the Near function takes $O(log(n))$ time.

Here, CF means the complexity of collision-free space.

$$T(n) = O(n) \text{ (Line 1)} + O(n \log(n) \times CF)$$

,

Assuming the *CollisionFree* function (Line 5) has a constant time complexity (O(1)) for a 2D problem, the overall time complexity is:

$$T(n) = O(n) \text{ (Line 1)} + O(n \log(n) \times 1) \text{ (Lines 2–6)}$$
$$T(n) = O(n) + O(n \log(n))$$

The actual time complexity will depend on the number of neighbors (n) and the efficiency of the nearest neighbor $\log(n)$. So the overall time complexity can be approximated as $O(n \times \log(n))$.

*RRT**

Asymptotic analysis of $RRT^*$ will be shown as we examine how the algorithm behaves as the number of sample $n$ grows.The main steps contributing to the algorithm's time complexity are as follows:

1. The loop iterating $n$ times: $O(n)$.
2. Finding the nearest vertex, $q_{nearest}$: $O(\log n)$, assuming a balanced search data structure.
3. Finding the near vertices, $X_{near}$: $O(n^{\frac{1}{d}})$, where $d$ is the dimensionality of the configuration space.
4. Checking for collision-free paths: $O(n^{\frac{1}{d}})$, as it is performed for each vertex in $X_{near}$.

Keeping the above-mentioned factors in mind, the overall time complexity of the RRT* algorithm is $O(n^{\frac{d+1}{d}} \log n)$. As asymptotic constants are insignificant, we can consider the complexity as $O(n \log n)$. As the number of samples $n$ increases, the algorithm converges to an optimal solution with probability approaching 1, making it an asymptotically optimal planner.

### 3.6.2 Theorems regarding probabilistic completeness, and asymptotic optimality

Several theorems concerning the probabilistic completeness, asymptotic optimality, and complexity of the algorithms are listed here, and the proofs are available in [KF11].

**Theorem 1** *(Probabilistic Completeness of $RRT^*$). The proof is available in [KF11].*

**Theorem 2** *(Probabilistic Completeness of $PRM^*$). The proof is available in [KF11].*

**Theorem 3** *(Asymptotic optimality of $PRM^*$). If $\gamma_{PRM} > 2(1+\frac{1}{d})^{\frac{1}{d}} * \frac{\mu(X_{free})}{\zeta_d}$, then $PRM^*$ algorithm is asymptotically optimal.*

**Theorem 4** *(Asymptotic optimality of $RRT^*$). If $\gamma_{RRT} > 2(1+\frac{1}{d})^{\frac{1}{d}} * \frac{\mu(X_{free})}{\zeta_d}$, then $RRT^*$ algorithm is asymptotically optimal.*

# 4 Discussions

The limitations of PRM* and RRT* algorithms can be attributed to several factors, such as the increased dimensionality of the configuration space, the nature of optimal paths, and the reliance on random sampling strategies.

As the dimension of the configuration space expands, the complexity of the configuration space, increases. This change implies that the asymptotic complexity of the configuration space will no longer be $O(nlogn)$ and will grow with the dimensionality. In real-world scenarios, which typically involve 3D environments, the collision space complexity escalates , consequently affecting the time complexity of both algorithms.

Moreover, PRM* and RRT* are probabilistically optimal, indicating that they can generate an optimal solution as the number of samples approaches infinity. However, in practical situations, the environment's complexity and the dependency on sample generation can hinder this guarantee. The convergence rate of both algorithms is contingent on the sampling rate and the intricacy of the geometry.

Although employing dynamic radius or K-nearest strategies mitigate the issue of excessive vertex exploration, both algorithms may require more samples to navigate through narrow passages. This issue can lead to local minima, resulting in suboptimal paths in real-world applications.

Using random sampling techniques in both algorithms may lead to inefficient search strategies and the generation of suboptimal paths. By introducing more advanced sampling methods, such as informed or adaptive sampling, the exploration of vertices could be significantly improved, given ample time. Empirical observations reveal a certain bias in the sampling approach during some runs, avoiding specific regions within the configuration space.

In particular, PRM* may encounter memory issues due to its storage of the entire graph structure, which can result in high memory requirements in real-world scenarios. Furthermore, the sampling strategy is crucial for both algorithms, as it heavily depends on the environment. Inefficient sampling strategies in high-dimensional spaces might consume a significant amount of time. In contrast, the $RRT^*$ is faster and can produce feasible solutions quickly, making it suitable for real-time applications.

From the algorithm enhancement standpoint, a handful of algorithms build upon $RRT^*$, such as $RRT\#$. However, to the best of our knowledge, no algorithms have been proposed that adapt $RRT^*$ or $PRM^*$ for dynamic environments. Thus, the prospect of integrating these two algorithms in situations where obstacles dynamically change their positions offers an enticing research avenue. Additionally, addressing the sampling challenges by examining the performance of these algorithms with samples generated following the Van der Corput theorem could provide a captivating opportunity for further investigation, ensuring a more comprehensive understanding of their potential applications.

# References


[BLP85] Rodney A Brooks and Tomas Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):224–233, 1985.

[KB+91] Yoram Koren, Johann Borenstein, et al. Potential field methods and their inherent limitations for mobile robot navigation. In *Icra*, volume 2, pages 1398–1404, 1991.

[KF11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[KL00] James J Kuffner and Steven M LaValle. 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. IEEE, 2000.

[LK01] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan. *Algorithmic and computational robotics*, pages 303–307, 2001.

[Rei79] John H Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427. IEEE Computer Society, 1979.