Probabilistic Completeness Proofs for Sampling-Based Motion Planning

Andreas Orthey

August 2024

This document details how a proof of probabilistic completeness can be constructed for sampling-based motion planners [1]. Probabilistic completeness means that a motion planner will find a solution if one exists if time goes to infinity [2]. The proof covers tree-based planners like the rapidly-exploring random tree (RRT) [3] and relies on some mild assumptions, namely that the configuration space is well behaved [4], that it has an attached metric, and that there exists a feasible path with ϵ clearance to any constraints. Given those assumptions and a motion planner, we can use the series-of-balls argument [5, 6] to verify that the motion planner fulfills probabilistic completeness.

1 Motion Planning Problem

Let X a configuration space [2] and X_{free} the constraint-free region of the configuration space (i.e. avoiding obstacles, respecting joint limits, etc). A motion planning problem is defined as the tuple $(X, X_{\text{free}}, x_I, X_G)$ with $x_I \in X_{\text{free}}$ being an initial configuration, $X_G \subseteq X_{\text{free}}$ a goal region, and the task being to find a path $p : [0, 1] \to X_{\text{free}}$ (a continuous mapping) which connects the initial configuration ($p(0) = x_I$) with the goal region ($p(1) \in X_G$) while staying in the constraint-free region of the configuration space.

1.1 Probabilistic Completeness

Let A be a motion planner, i.e. an algorithm which solves motion planning problems $(X, X_{\text{free}}, x_I, X_G)$ by computing paths connection initial configuration to goal region.

We say that A is complete, if it finds a solution if one exists or to correctly report that no solution exists. This is a desirable property of any planning algorithm, because it gives us a guarantee that eventually we will find a valid solution if there is one.

Probabilistic completeness is a weaker notion as completeness and is defined as: A will find a feasible path if one exists when time goes to infinity. Note the difference to completeness in that it will not report that no solution exists, i.e. if there is no solution, a probabilistically complete planner would not terminate.

2 Proof of Probabilistic Completeness

The proof idea is relatively straightforward and done by construction through induction. While this works for many different motion planners, we concentrate here on tree-based planners which are relatively easy to define [2].

Here is the high-level proof idea. First, we assume that there exists a feasible path in our configuration space for a given motion planning problem. Second, we then cover this path by a set of overlapping n-balls of a certain size δ in the dimension n of the given configuration space. Given this representation, we then reason by induction, whereby we first prove the base case, i.e. that we can reach the first ball, and then we prove the induction step, namely that the algorithm will reach a ball $k+1$ if it has already reached ball k. Let us look at this proof in detail.

2.1 Step 0: Algorithm Structure

While the proof can be adapted to almost any planner to show probabilistic completeness, we focus in this document on tree-based sampling-based planners [3], who's structure is detailed in Alg. 1. This tree-based planner takes a motion planning problem as input (Line 1) and returns a path from the initial configuration to the goal region (Line 2). The planner first initializes a tree (Line 3), with the initial configuration as root node, and then enters a while loop (Line 4). In each iteration of this loop, the planner samples a random configuration (Line 5), computes the nearest configuration in the tree (Line 6), and connects both (Line 7) if possible. Finally, the planner checks if a solution exists in the tree (Line 8), and returns a feasible solution path on success (Line 9). This is the basic structure which underlines several tree-based planners like RRT [3] or the expansive space-trees (EST) planner [7]. Those planners, however, differ significantly in how the sampling and nearest functions are implemented.

Algorithm 1 Tree-Based Sampling-Based Motion Planning

```
1:  Input: Start configuration x_init, Goal region X_G, Configuration space X
2:  Output: Path τ from x_init to X_G
3:  Initialize tree T ← {x_init}
4:  while True do
5:      x_rand ← Sample(X)
6:      x_near ← Nearest(T, x_rand)
7:      x_new ← Connect(x_near, x_rand)
8:      if SolutionExists(T) then
9:          return Path(T)
10:     end if
11: end while
```

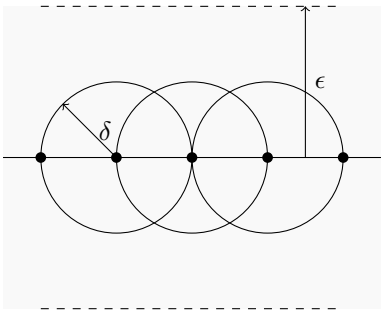

Figure 1: Covering a path of ϵ clearance with δ-spaced balls of radius δ.

2.2 Step 1: Assumptions

Let us first state all the required assumptions. Note that they all are relatively mild, i.e. this is actually fulfilled by most motion planning problems in real scenarios.

1. There exists a feasible path connecting start to goal. If such a path does not exist, the behavior of Alg. 1 is undefined.
2. The feasible solution path has $\epsilon > 0$ clearance, meaning the distance between the path image and the constraints on the configuration space is at least ϵ. If this is not the case, sampling-based planner will not be able to find solutions.
3. There exists a sampling function f on X which returns a dense set of samples on the configuration space[1].
4. If two points $x, y \in X$ have a distance of d between them, then we can always construct a path segment (an edge) between x and y of length smaller or equal to d.

What do those assumptions mean in practice? Assumptions 1 and 2 mean a restriction of the proof to a certain subclass of problems. Namely problems which have a solution (Assumption 1), and problems in which the robot is not touching the environment (Assumption 2). Assumptions 3 and 4 are additional properties of the algorithm itself, namely that the sampling function is uniform (Assumption 3) and that the connection between two configurations is consistent with the metric used (Assumption 4).

2.3 Step 2: Cover feasible path with balls

Let us conduct the first construction step in our proof, namely to cover a feasible path with a sequence of balls. For that, we use the assumption that there exists

[1]Recall that a sequence is said to be dense on a configuration space X if every point of X is arbitrarily close to a member of the sequence [4].

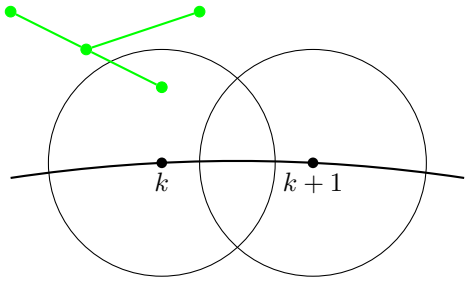

Figure 2: The tree of Alg. 1 (green) has reached ball k. To prove the induction step, we need to show that once we reach ball k, we will eventually reach ball $k+1$ in some future iteration of the algorithm.

some feasible path (Assumption 1) and that this path has at least $\epsilon > 0$ clearance to the environment (Assumption 2). To compute balls, we have to chose their radius and their spacing along the path. One option is to choose a δ radius given by $\delta = \frac{\epsilon}{3}$ and to create δ-spaced balls along the path which have δ radius. Fig. 1 shows schematically how this would look like. Note that since ϵ is a positive number, there always exists some δ for which we can construct the balls.

2.4 Step 3: Induction Base Case and Induction Step

Once we have constructed a series of balls along the path, we can conduct the main proof step, namely to show that this path is eventually found by using Alg. 1. To do this, we use the method of mathematical induction, which requires two steps: We need to show that the first ball is reached (induction base case), and we need to show that ball $k+1$ is reached once we reach k (induction step). With this, we would have proven that the path is found.

Let us prove those two steps. First, let us verify the base case, which is that our algorithm will reach the first ball in the sequence. Since the first ball contains the start configuration, we only need to ensure that the start configuration is added. This is fulfilled for example by RRT, since it creates a search tree with the start configuration as its root.

Second, we need to verify the induction step. Let us assume that there are K balls in the sequence and that we reached ball k along the way. We want to show that given that we reached ball k, we will eventually reach ball $k+1$. See Fig. 2 for clarification. To accomplish this, we first use the fact that the overlap of the balls has non-zero volume. Since our sampling sequence is dense (Assumption 3) we will eventually draw a sample inside of the overlap region. This new sample will have a distance upper-bounded by 2δ (the diameter) to the sample in ball k. That means that this sample is either the nearest or there exists a nearer one. However, if another sample exists, which is nearer, than this has to be distance bounded by $2\delta < \epsilon$ (Assumption 4), and thereby lie inside the ϵ-neighborhood of the path. This shows that the new sample will be connected to the tree, which concludes our proof.

3 Conclusion and Limitations

We presented a proof of probabilistic completeness for sampling-based motion planners. While this proof is general and could be applied to any sampling-based motion planners, we have focused here on tree-based planners.

There are two limitations of this proof concerning optimality and dynamics. While this proof works for probabilistic completeness, you cannot exchange feasible paths in the proof with optimal paths and thereby extend the proof to prove optimality or asymptotic optimality. This requires a slightly different approach, whereby we need to ensure that all past balls have also been reached (and not only the k-th ball) [8].

Another limitation are kinodynamic systems like a kinematic car. It turns out that this proof can only be extended for such systems if a steering function exists [2], and if we know that this steering function is bounded in length [5]. This means that if two configurations are a distance d apart, the steering function should produce a path with length bounded by some function of d. If this is the case, we can always make small balls which keep the steering bounded, and thereby extend the proof to kinodynamic systems.

References

[1] Andreas Orthey, Constantinos Chamzas, and Lydia E. Kavraki. Sampling-based motion planning: A comparative review. *Annual Review of Control, Robotics, and Autonomous Systems*, 7(1):285–310, 2024.

[2] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[3] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001. IEEE, 2000.

[4] James Raymond Munkres. *Topology*. Prentice Hall, Upper Saddle River, NJ, 2 edition, 2000.

[5] Petr Švestka and Markus Hendrik Overmars. Probabilistic path planning. In *Robot motion planning and control*, pages 255–304. Springer, 2005.

[6] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011.

[7] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *International conference on robotics and automation*, volume 3, pages 2719–2726. IEEE, 1997.

[8] Kiril Solovey, Lucas Janson, Edward Schmerling, Emilio Frazzoli, and Marco Pavone. Revisiting the asymptotic optimality of rrt. In *IEEE international conference on robotics and automation*, pages 2189–2195. IEEE, 2020.