RANDOMIZED MOTION PLANNING – A TUTORIAL

Stefano Carpin
International University Bremen, Germany, email: s.carpin@iu-bremen.de

Abstract

Sampling demonstrated to be the algorithmic key to efficiently solve many high dimensional motion planning problems. Information on the configuration space is acquired by generating samples and edges between them, which are stored in a suitable data structure. Following this paradigm, many different algorithmic techniques have been proposed, and some of them are now widely accepted as part of the standard literature in the field. The paper reviews some of the most influential proposals and ideas, providing indications on their practical and theoretical implications.

1 Introduction

Many fields beyond robotics, are benefiting from progresses in algorithmic motion planning. Examples include structural studies in biology, computer graphics, and computer assisted surgery among the others [44]. The successful application of these algorithms relies on the availability of an abstract formulation suitable to model many different real world applications, and on the ability to solve difficult instances in a reasonable time. Motion planning is performed as a search in a suitable space, called the *configuration space*. Early studies outlined that the basic version of this problem is PSPACE-complete [13],[60], and the best exact deterministic algorithm known is exponential in the dimension of the configuration space [12]. On the other hand, real world problems generate instances with high dimensional configuration spaces. Around the mid nineties a new approach was introduced, and this boosted the research in the field, as well as the practical use of these algorithms. This technique is based on the generation of samples to aquire information about the problem instance being solved. It has to be outlined that while the first algorithms heavily relied on random samples, there has been a recent trend to introduce also deterministic sampling schemas [46],[54], although this will not be covered in this paper. Samples are stored in a data structure which represents an approximation of the configuration space, as opposed to its exact combinatorial representation. The data structure is usually composed by nodes, i.e. samples in the configuration space, and links, i.e. valid paths connecting samples. Nodes and links can be stored in graphs or trees. The sampling based approach has some appealing properties. It is immediately applicable whenever a configuration state space is used to model a real world problem. For example, sampling based motion planners have been used for multi-robot systems [14],[65],[61], closed chain systems [20],[27],[68], and deformable objects [9],[42]. These algorithms are also well suited for practical parallel implementation [15],[16],[29], thus allowing further performance gains. Finally, the implementation of these algorithms is usually quite simple. The price to pay is completeness. Traditional combinatorial motion planning algorithms are complete, i.e. they will find a solution if one exists, and will report failure otherwise. Algorithms based on randomly generated samples obtain instead probabilistic completeness. This means that if a solution exists, the probability to find it converges to 1 when the computation time approaches infinity. So, if a path is not found it could be the case that a solution does not exist at all, or that the sampling process has not been able to get the information needed to solve the search. There have been also attempts to address the dual problem, i.e. to prove that a solution cannot exist [8]. However, while in principle this approach is appealing, it has not been shown to work under general conditions, and has not enjoyed great use up to now. In between the above mentioned forms of completeness, algorithms based on deterministic samples sequences achieve resolution completeness. This means that when the algorithm fails to find a solution, this implies that either the solution does not exist, or it requires a sampling resolution below the one used. This approach has been attracting some attention recently [17]. The randomized framework can be extended in many directions. This motivates the great number of different algorithms proposed up to now. Some algorithms and techniques proved to be very efficient and are now part of the standard literature in computational robotics. In addition to operative aspects, also the theoretical foundations of this approach have been addressed, so that some understanding about the power and the limitations of the framework has been obtained, although we are far from having a complete picture. This paper presents some of the most common algorithms, and give an overall perspective on the most widely used ideas in the field. For sake of completeness, we have to mention that the first broadly used motion planning algorithm incorporating random components has been

the so called Randomized potential field Path Planner (RPP) [7]. RPP uses a deterministic gradient descent strategy over a suitably defined potential field. When the planner gets stuck at in one of the possible local minima, a sequence of random motions of increasing length is activated, until the potential well is escaped. However, the RPP planner will not be addressed in the paper because the underlying mechanism is different, i.e. it does not generate samples to explore the configuration space, but only to recover from problematic situations.

The paper is organized as follows. Section 2 introduces the formal statement of the robot motion planning. Algorithms based on probabilistic roadmaps are discussed in the successive section. The basic formulation, as well as improvements, are illustrated and discussed. The section also provides the fundamental theoretical results concerning probabilistic convergence. Following the same approach, section 4 reviews algorithms whose underlying data structure is a tree rather than a graph. Finally, section 5 addresses practical issues related to the choice of a specific algorithm, while conclusions are offered in section 6.

2 Problem formulation

Extensive treatment of the basic computational aspects or robot motion planning can be found in [33],[43],[48],[62]. We here formalize the problem. The robot motion planning (RMP) problem is dealt with by using the configuration state space approach introduced by Lozàno-Perez [55]. Every robot is associated with a set of degrees of freedom which specify its placement in the workspace. The combination of the degrees of freedom assumes values in a space called the *configuration space*, usually indicated as $\mathcal{C}$. The configuration space is partitioned into two subsets, the space of free configurations, $\mathcal{C}_{free}$, and the space of obstacle configurations, $\mathcal{C}_{obs}$. The space of free configurations is the subset of valid configurations, i.e. configurations in which the robot does not collide with any obstacle and does not violate its mechanical constraints. The space of obstacle configurations is its complement, i.e. $\mathcal{C}_{obs} = \mathcal{C} \setminus \mathcal{C}_{free}$. Given two points $x_{start} \in \mathcal{C}_{free}$ and $x_{goal} \in \mathcal{C}_{free}$, the RMP problem requires to compute a continuous function $f : [0,1] \rightarrow \mathcal{C}_{free}$ such that $f(0) = x_{start}$ and $f(1) = x_{goal}$. In general this function could even not exist, for example if $\mathcal{C}_{free}$ is disconnected and x_{start} and x_{goal} belong to different components. An ideal RMP algorithm should be able to determine this situation and stop the computation as soon as it is possible to establish that a solution cannot be found. As we will see, randomized algorithms fail to accomplish this goal, so it is common to bound the computation time in order to avoid infinite loops while trying to solve unsolvable problem instances. In the sampling based motion planning framework, it is assumed the availability of a collision detection function. Given a configuration, the collision checker determines whether a configuration belongs to $\mathcal{C}_{free}$ or to $\mathcal{C}_{obs}$. Formally, it is a function

$$Check : \mathcal{C} \rightarrow \{0,1\} \tag{1}$$

where 0 indicates the sample belongs to $\mathcal{C}_{obs}$, and 1 indicates the sample belongs to $\mathcal{C}_{free}$. In many practical situations robots are not allowed to execute arbitrary motions. A classical example is a car-like robot. In this case the robot is not permitted to perform certain actions, like for example to move along the axis connecting the rear wheels. These limitations are usually defined as differential constraints, i.e. equations or inequalities involving not only the degrees of freedom, but also their time derivatives. In the related literature the configuration space $\mathcal{C}$ is frequently substituted with the state space $\mathcal{X}$, which includes both the degrees of freedom and their derivatives. An element of the state space is then in the form $(x, \dot{x})$, where $\dot{x}$ indicates the first order time derivative[1]. Differential constraints assume the form $f(x, \dot{x}) = 0$, $f(x, \dot{x}) < 0$ or $f(x, \dot{x}) \leq 0$. If the constraints are not integrable the problem is called *non holonomic motion planning*. If the constraints depend on x and limit the possible values of $\dot{x}$ and/or $\ddot{x}$, the problem is called *kinodynamic* motion planning. For instance, a bound on the maximal speed (dynamic constraint) might be not absolute but rather related to the position (kinematic constraint) of the robot in its environment. This could allow higher speeds in wide areas and force slow motions when the robot is near to an obstacle. The exact solution of the three dimensional kinodynamic problem is NP-hard, while approximated dynamic programming based solutions have been illustrated in [24],[22] and [23].

3 Probabilistic Roadmaps

Probabilistic roadmaps (PRM), introduced in [37] (see also [34],[58] for preliminary versions). For sake of completeness it has to be acknowledged that some related ideas can be found in the earlier work [26]. It should also be mentioned that in the same period a similar approach was introduced in [56], the so called *Ariadne's clew* algorithm. The difference is that the Ariadne's clew algorithm does not explore the configuration space, but rather the trajectory space. The PRM algorithm works in two steps, the first called *learning stage* and the second called *query stage*. Given an instance of the RMP problem, in the learning stage the algorithm samples the configuration space and builds an undirected graph $G = (V, E)$ which captures the information gathered. The graph is called *probabilistic roadmap* (PRM). In the query stage, the PRM

[1] even higher order derivatives could be included, for example to take into account accelerations and so on.

is used to solve specific RMP problem instances which are then reduced to a graph search. The availability of the following elements is assumed:

- a subroutine *Check* which computes the function described in equation 1
- a subroutine *Distance* which computes a distance between two configurations, i.e. it computes a function
$$D : \mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}^+ \cup \{0\} \quad (2)$$
that associates two configurations with a non negative real number.
- a fast and possibly incomplete deterministic planner

3.1 The learning stage

The learning algorithm is illustrated in algorithm 1. Samples randomly generated over $\mathcal{C}$ are accepted if they belong to $\mathcal{C}_{free}$, and discarded otherwise. When a sample is accepted, it becomes a vertex of the PRM graph. After a vertex is added, the algorithm checks if it is possible to add edges between the inserted vertex and vertices already in the graph. For this aim a subset of neighboring vertices are selected and the deterministic planner is run to determine if the new vertex can be connected to them. Two vertices are neighbors if their distance D is less than a fixed threshold M. This choice is made for sake of efficiency, as it is unlikely that a couple of far apart vertices could be connected by the simple planner. At the same time we wish to minimize the number of calls to the simple planner. Another often used efficiency driven choice is to limit the size of the neighbors set to a maximum size, say K. If the planner succeeds in finding a free path between them, and if the nodes do not belong to the same connected component of the graph, an edge connecting the two vertices is added to the graph. Even if different techniques have been evaluated [2],[25], in the vast majority of implementations the deterministic planner simply connects the two points with a straight segment and verifies if it lies in $\mathcal{C}_{free}$ or not. This is done by selecting a set of intermediate points along the segment and by calling the collision checker on each of them. The segment is declared to be free if all the intermediate points lie in $\mathcal{C}_{free}$. This approach is inherently error prone, as just a discretization is used to determine the validity of the entire segment. Recently however a new algorithm which computes exact collision checking has been introduced [63], and its use appears appealing in the context of sampling based RMP algorithms, as it can be used to perform exact validation of the edges. If the resulting graph contains more than one connected component (see figure 1), the learning stage is often followed by a roadmap

Algorithm 1 Basic PRM algorithm: learning stage

1: $V \leftarrow \emptyset$
2: $E \leftarrow \emptyset$
3: **loop**
4: Generate a random configuration $c \in \mathcal{C}_{free}$
5: $V \leftarrow V \cup \{c\}$
6: $V_n \leftarrow \{v \in V \quad | \quad Distance(c, v) < M\}$
7: **for all** vertices $v \in V_n$ in order of increasing $Distance(c, v)$ **do**
8: **if** c and v can be connected by the simple planner and they do not lie in the same connected component **then**
9: $E \leftarrow E \cup \{(c, v)\}$

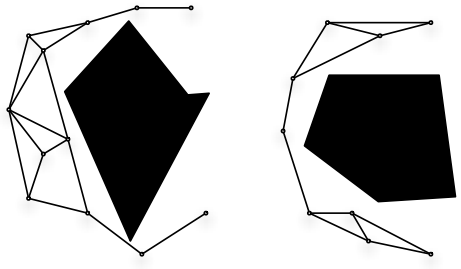

Figure 1. An example of Probabilistic Roadmap. Black regions indicate $\mathcal{C}_{obs}$. In this case the PRM consists of two connected components and a roadmap improving stage could succeed in merging them

improving substep. The improving is done by selecting some vertices assumed to lie in difficult regions and then trying to add more samples in their neighborhood and more edges. Different heuristics cab be used to decide if a vertex is in difficult region or not (see for example [37]). If the configuration space is indeed disconnected, the improving step could even be not necessary, meaning that it could not improve the overall roadmap quality at all. On the other hand, due to the approximated nature, it is unfeasible to determine whether the improving step is indeed needed or not.

3.2 The query stage

In the query stage two configurations x_{start} and x_{goal} are given, and the algorithm is required to produce a path between them, provided it can be extracted from the graph (see algorithm 2). The algorithm tries to connect the two points to vertices in the graph G. This trial is done by using the same technique used to to insert edges in the roadmap, i.e. vertices are probed by increasing distance order. If it is not possible to connect both x_{start} and x_{goal} to the roadmap

G, the algorithm reports failure. Otherwise, let us assume that x_{start} has been connected to a vertex V_s and that x_{goal} has been connected to a vertex V_g. Then, a graph search is performed to verify whether v_s and v_g belong to the same connected component of G. If the search succeeds, then the path is returned. Later, the sequence of segments could be smoothed by using standard path smoothing algorithms in order to improve path quality. The goal of this substep is to eliminate useless motions. This is usually done by picking some random couple of confiugarions and trying to replace the path connecting them with a straight line.

Algorithm 2 Basic PRM algorithm: query stage

$V_s \leftarrow \{v \in V \quad | \quad Distance(x_{start}, v) < M\}$
if the planner can find a path between x_{goal} and a vertex in V_s **then**
  Let $v_s \in V_s$ be that vertex
else
  return Failure
$V_g \leftarrow \{v \in V \quad | \quad Distance(x_{goal}, v) < M\}$
if the planner can find a path between x_{goal} and a vertex in V_s **then**
  Let $v_g \in V_g$ be that vertex
else
  return Failure
if a path P between v_s and v_g is found **then**
  return the overall solution path $(x_{start}, v_s), P, (v_g, x_{goal})$
else
  return Failure

3.3 Remarks on the PRM algorithm

The PRM algorithm has been successfully used to solve problems with many degrees of freedom. There is also experimental evidence that the algorithm fits well in a parallel computational environment [3]. However some drawbacks were also outlined and motivated the of further research. First, it is evident that the learning stage will take much more time than then query stage. Therefore the learning stage is worth only if many queries will benefit from the created roadmap. This is not the case when the user is interested in the *single shot* instance, i.e. just one instance of the problem has to be solved. Similarly, if the robot is to move in a dynamic environment, the PRM will be no more valid as obstacles move, thus vanishing the time spent to build it. Second, in the PRM framework it is common to use uniform sampling over $\mathcal{C}$ while generating samples, but this sampling strategy has its disadvantages, namely the low probability to place samples in narrow regions. The difficulty of discovering narrow passages in $\mathcal{C}$ is one of the motivations which led to development of many refinements proposed.

3.3.1 Extensions to the PRM algorithm

After the introduction of the aforementioned algorithm, there have been various extension proposed, like lazy PRMs [10] or visibility based probabilistic roadmaps [57]. In this subsection in particular we describe this last extension to the original PRM framework. One of the bottlenecks of PRMs is that as the number of samples grows, it becomes more and more expensive to updated the graph. It would then be beneficial to keep graph's size low. In the proposed algorithm, nodes in the graph belong to two different categories: guards and connection nodes. To each guard is associated a visibility region, i.e. the region of points in $\mathcal{C}_{free}$ that can be reached with a straight line segment. Connection nodes are nodes placed in the intersections of the regions associated with two different guards. Edges are created only between nodes of different types, i.e. between guards and connection nodes. This way, nodes are added to the roadmap only when they really improve it, i.e. when they allow to create paths previously not possible. Nodes generated into the visibility region of a single guard are not added to the graph, as this would not create paths previously not possible. Experimental results outline that with the same amount of generated random samples, this technique allows to cover more or less the same portion of the configuration space. At the same time, because of the restricted number of nodes in the graph, the processing time turns out to be much smaller.

3.4 The problem of narrow passages

As already stated, most of the proposed PRM based approaches rely on uniform sampling over $\mathcal{C}$. Uniform sampling poorly deals with narrow passages. In fact, if we indicate with $\mu(S)$ the measure of the set S, by using random sampling over $\mathcal{C}$ the probability of placing a sample inside S is $\mu(S)/\mu(\mathcal{C})$. This clearly indicates the inability of the algorithm to quickly find out small volume regions.

3.4.1 Obstacle based and medial axis probabilistic roadmaps

The problem of narrow passages is tackled in [1]. The authors address the problem in the context of *Obstacle Based Probabilistic Roadmaps* (OBPRM from now on). OBPRM [4] are a variant of the PRM algorithm where node generation is performed with the goal of placing samples near to or in contact with the obstacles. The underlying idea is that in this way it is possible to correctly operate even in the presence of cluttered environments, as narrow passages are the result of facing obstacles. The authors illustrate then

a set of different approaches both for node generation and for roadmap connection. Nodes generation is performed by using three different methods. The first one generates configurations in contact with obstacles (see algorithm 3). The second generates samples in

Algorithm 3 Algorithm for creating samples on the boundary of the obstacle O_j

1: GENERATE_CONFIGURATION
2: determine a point p inside the obstacle O_j
3: let M be a set of directions emanating from p
4: **for all** $m \in M$ **do**
5: use binary search to determine a point lying on the boundary of O_j along the direction m

free space, but near to the boundary of $\mathcal{C}_{free}$. These points can be obtained by a slight modification of algorithm 3 so that free space points rather than contact points are generated. The third one aims to create *shells* of configurations around obstacles, so that paths in those difficult regions can be quickly found. Shells are obtained by retaining some of the valid samples generated while looking for contact and free configurations in the previous steps. During the connection stage three different local planners are used. The first one is the usual straight line planner used in the simplest PRM implementation. The second one is the so called rotate–at–s, where s is a number between 0 and 1 [2]. While seeking a path between the configurations c_1 and c_2, this planner tries to translate the robot from c_1 to an intermediate configuration along the line connecting c_1 and c_2. Then it rotates the robot and it tries to translate it to the final c_2 configuration. The value s is the fraction along the straight line where the rotation is performed. The last planner is an A^*-like planner. During roadmap connection, three different stages are carried out. The first one, called *Simple Connection* utilizes the simplest planner (straight line), which is called many times to try many cheap connections among samples belonging to the same obstacle. The second stage, called *Connecting Components*, tries to create connections between disjointed roadmap components. The third stage, called *Growing Components*, has the same goal, but while trying to create connections, it can also generate new samples if needed. This is done by enhancing the map adding nodes near to small components and by keeping valid samples lying in segments not entirely accepted. In the second and third stages, while trying to connect two nodes the local planners are used in this order: straight line, rotate–at–1/2, rotate–at–0, rotate–at–1, A^*. Thus, by following an increasing planning cost strategy, expensive local planners are used just when cheaper and simpler planners fail to succeed. The authors report extensive simulation results illustrating that significant speedups can be obtained by using the combination of techniques they propose and also provide experimental driven recommendations about the techniques to utilize. The use of many different sampling and connecting techniques however has its own drawbacks, as many parameters should be fixed and it can then be non trivial to determine a suitable combined tuning. To overcome these difficulties, a slightly different approach that uses sampling on the medial axis of $\mathcal{C}_{free}$ was proposed [53],[66],[67]. In this framework, called MAPRM, Medial Axis PRM, samples are not generated on the surface of the obstacles, but rather over $\mathcal{C}$ and are then *retracted* into the medial axis of $\mathcal{C}_{free}$. The medial axis of the configuration space are the points in $\mathcal{C}_{free}$ with maximal distance from $\mathcal{C}_{obs}$. Formally, for $x \in \mathcal{C}_{free}$, we define $B_{\mathcal{C}_{free}}$ to be the largest closed ball centered in x and completely lying in $\mathcal{C}_{free}$. The medial axis of the space of free configurations are defined as the set of points whose associated $B_{\mathcal{C}_{free}}$ are maximal, i.e.:

$$MA(\mathcal{C}_{free}) = \{x \in \mathcal{C}_{free} | \nexists y \in \mathcal{C}_{free}$$
$$\text{with} \quad B_{\mathcal{C}_{free}}(x) \subsetneq B_{\mathcal{C}_{free}}(y)\}.$$

Known properties of medial axis guarantee that the network or medial axis associated with the configuration space captures the connectivity of the space itself. Therefore, by relying on this reduced representation no significant information is lost. The strength of the algorithm relies on the ability to efficiently push or pull a sample to a medial axis. The authors report algorithms for dealing both with two and three dimensional workspaces, while here (see algorithm 4) we sketch the simple algorithm for retracting a sample into the medial axis of a two dimensional environment. Once a sample c over $\mathcal{C}$ is generated, the nearest point lying on the boundary of $\mathcal{C}_{free}$ is determined (here it is indicated as n). Then, if c lies in $\mathcal{C}_{free}$, it is retracted to the medial axis, otherwise the point to move is n. The line to move along in order to reach the medial axis is determined by c and n, while the direction depends whether c is in $\mathcal{C}_{free}$ or not. In order to efficiently

Algorithm 4 Medial axis retraction algorithm

1: Let $c \in \mathcal{C}$
2: Among the points in $\partial\mathcal{C}_{free}$ determine the nearest to c, and call it n
3: **if** $c \in \mathcal{C}_{free}$ **then**
4: $\vec{d} \leftarrow n\vec{c}$
5: $s \leftarrow c$
6: **else**
7: $\vec{d} \leftarrow c\vec{n}$
8: $s \leftarrow n$
9: Move s along the direction $\vec{d}$. Stop moving s when n is not the unique nearest point of $\partial\mathcal{C}_{free}$ to s

perform the computation, the retraction step (line 9) is performed by using a bisection technique. The overall PRM algorithm generates samples uniformly over $\mathcal{C}$

and the retracts them over the medial axis. A graph is then built by connecting those samples, and this builds up the probabilistic roadmap. The rational of this approach is the following: to discover narrow passages, it is no more necessary to generate samples into the narrow passages themselves, but rather to generate samples that once retracted end up into the medial axis associated with the narrow passages, thus increasing the probability of discovering them. This is achieved by sampling over the entire $\mathcal{C}$ rather than over just $\mathcal{C}_{free}$.

3.4.2 Planning in dilated spaces

Along the same lines of OBPRM, some authors [30] pushed the idea of sampling on obstacle surfaces even further, by allowing the generation of samples lying outside $\mathcal{C}_{free}$. The planning is divided into two stages. First a roadmap is created in a *dilated* configuration space. This means that if a sample lies inside $\mathcal{C}_{obs}$, but its distance from $\mathcal{C}_{free}$ is smaller than a certain threshold δ, it is retained rather than discarded. This step is performed using the classical PRM algorithm where the space $\mathcal{C}_{free}$ is substituted by its dilatation $\mathcal{C}_{free}^{dil}$

$$\mathcal{C}_{free}^{dil} = \mathcal{C}_{free} \cup \{c \in \mathcal{C}_{obs} | Distance(c, \mathcal{C}_{free}) < \delta\}.$$

In the second stage samples inside $\mathcal{C}_{obs}$ are pulled into $\mathcal{C}_{free}$. This is done by resampling in their neighborhood. Next, edges have to be created, and also in this case resampling could be needed in order to push edges into $\mathcal{C}_{free}$. Algorithm 5 illustrates this approach. In the algorithm, $U_v(v)$ is the resampling region associated with vertex v, while $U_e(v_1, v_2)$ is the resampling region associated with the edge (v_1, v_2). By using the dilated configuration space, narrow passages are easier to detect, as they are widened. The choice of the value of δ is very important. Taking it too small would not give too many advantages over the basic PRM algorithm, but taking a too big value of δ has its disadvantages too, as entire obstacles could then disappear. The authors illustrate the encouraging results of their simulation and offer some hints about practical implementation. The resampling region $U_v(v)$ is a sphere centered in v while $U_e(v_1, v_2)$ is a square. Moreover, the authors found that by using a series of decreasingly dilated spaces better results can be obtained.

3.5 Probabilistic convergence of the PRM algorithm

The PRM algorithm is probabilistic complete, meaning that if enough time is alloted to the learn stage, it will eventually create a roadmap that will determine the solution of every solvable RMP problem instance. This is intuitive, as the uniform sampling process over

Algorithm 5 Algorithm for creating a valid roadmap starting from a roadmap created in then dilated configuration space

```
1: Generate a Roadmap R' = (V', E') in the dilated space C_free^dil.
2: V ← ∅    E ← ∅
3: for all v' ∈ V' do
4:   if v' ∈ C_free then
5:     V ← V ∪ {v'}
6:   else
7:     pick up to k samples in U_v(v') and add to V the first on lying in C_free (if any)
8:   let p(v') be the vertex added to V (if any)
9: for all (v_1, v_2) ∈ E' do
10:  if (p(v_1), p(v_2)) ∈ C_free then
11:    E ← E ∪ {(p(v_1), p(v_2))}
12:  else
13:    Resample in U_e(p(v_1), p(v_2)). Let R be this sample set
14:    if by using samples in R a path connecting p(v_1) and p(v_2) is found then
15:      add the samples and the edges to V and E respectively
```

$\mathcal{C}_{free}$ will eventually cover it all, but from a practical point of view it would be precious to know how many nodes should be in the roadmap in order to get a desired probability of success. This is of course related to shape of $\mathcal{C}_{free}$, and to the placement of x_{start} and x_{goal} therein. We here give the results concerning the two main contributions given in the literature. In [35] basic speculations concerning the basic version of the PRM algorithm are illustrated. There are three parameters playing an important role in the overall planning performance. Let us suppose that there exists a path p connecting x_{start} and x_{goal}. The first relevant parameter is L, the length of p. The second parameter is ε which is the Euclidean distance of p from $\mathcal{C}_{obs}$, and the third parameter is N, the number of vertices in the graph (i.e. $N = |V|$). The first result proved by the authors is the following.

Theorem 3.1 *Let $p : [0, L] \rightarrow \mathcal{C}_{free}$ be a path connecting x_{start} and x_{goal} and let ε be its distance from $\mathcal{C}_{obs}$. Let d be the dimension of the configuration space. Then the probability that the PRM will fail to connect x_{start} and x_{goal} is at most*

$$\frac{2L}{\varepsilon}(1 - \alpha\omega_d)^N \tag{3}$$

where $\alpha = \varepsilon^d/(2^d|\mathcal{C}_{free}|)$ and ω_d is the volume of the unit ball in the $d-$dimensional space.

The result provides two indications. First, the bound claims that the probability of failure decreases while increasing the number of samples in the roadmap. The second point is that the failure probability increases

with the increase of path length. Again, this is somehow expected. Long paths require more information, i.e. more samples, to be caught. Also the dependence on ε, is to be expected. It takes into account the problems given by narrow passages in $\mathcal{C}_{free}$, since more samples will be needed to discover them. Theorem 3.1 considers just the minimum distance between the path p and $\mathcal{C}_{free}$. The authors then provide a second bound which considers a mean distance between the path and the obstacle space.

Theorem 3.2 *Let p be a path of length L connecting x_{start} and x_{goal}, i.e. $p : [0, L] \to \mathcal{C}_{free}$. Let $\varepsilon(t)$ be the distance between the path and $\mathcal{C}_{obs}$ at instant t. Then, the failure probability is bounded by*

$$6 \int_0^L \frac{(1 - \frac{\alpha_d}{2^d} \omega_d \varepsilon^d(t))^N}{\varepsilon(t)} dt \tag{4}$$

where $\alpha_d = 2^{-d} |\mathcal{C}_{free}|$ and ω_d is the volume of the unit ball in the $d-$dimensional space.

Both bounds depend either on ε or $\varepsilon(t)$, and also on α. This bound is then far from being trivial to compute, since the exact computation of these parameters is not easy. They depend on the path p and on the shape of $\mathcal{C}_{free}$. A different analysis is presented in [6], [36], and [29]. It is performed assuming of the availability of a fast but incomplete planner, indicated as B_S, and of a complex but complete planner, indicated as B_C. In the following, two configurations are said to be *visible* if they can be connected by using the simple planner B_S. Two basic concepts there introduced are the notion of *ε-goodness* and of *adequate* samples set.

Definition 3.3 *Let $c \in \mathcal{C}_{free}$ and let $S(c) \subseteq \mathcal{C}_{free}$ be the set of points visible from c. Let $\varepsilon > 0$ be a real number. A configuration $c \in \mathcal{C}_{free}$ is ε-good if $\mu(S(c)) \geq \varepsilon \mu(\mathcal{C}_{free})$, where μ indicates the volume of the given set. $\mathcal{C}_{free}$ is ε-good if all its elements are ε-good.*

Definition 3.4 *Let $\mathcal{C}_{free}$ be an ε-good space of free configurations. A set of samples V is* adequate *if the volume of $\mathcal{C}_{free}$ not reachable from V by using B_S is at most $(\varepsilon/2)\mu(\mathcal{C}_{free})$*

The ε-goodness property states that from every point of $\mathcal{C}_{free}$ it is possible to see a significant portion of $\mathcal{C}_{free}$ itself, while a set of vertices is declared to be adequate if from that set the part of $\mathcal{C}_{free}$ not visible from it is bounded. Thus one can expect that an ε-good configuration space can be well covered by using not too many samples. The bounds on the failure probability are formulated in terms of the following preprocessing procedure, which is a slight variation of the basic PRM algorithm.

1. Generate k samples in $\mathcal{C}_{free}$
2. Run the simple planner B_S over every couple of samples
3. Pick a sample from every connected component of the graph built so far
4. Use the complex planner B_C to try to connect couples of samples belonging to different components

The reader is referred to the cited papers for details on the fourth step, called *permeation*.

Theorem 3.5 *Let $\beta \in [0, 1)$ and let $k = (c/\varepsilon)(\ln 1/\varepsilon + \ln 4/\beta)$ where c is a fixed positive constant large enough that for all $x \in [0, 1)$ the inequality $(1 - x)^{(c/x)(\ln 1/x + 4/\beta)} \leq x\beta/4$ holds. In such hypothesis the preprocessing stage will generate an adequate samples set with probability at least $1 - \beta$.*

An additional bound can be obtained for the query step, provided that the the it is performed according to the algorithm illustrated in algorithm 6. In that case the following theorem holds.

Algorithm 6 Query algorithm to be used in order to obtain the bound given in theorem 3.6

```
for i = 1 to 2 do
    if a sample s ∈ V is visible from c_i then
        s_i ← s
    else
        for log(2/γ) times do
            let u_i be a random point visible from c_i
            if a sample s ∈ V is visible from u_i then
                s_i ← u_i
        if all log(2/γ) trials failed then
            Output FAILURE
if s_1 and s_2 are in the same component of G then
    Output SUCCESS
else
    Output FAILURE
```

Theorem 3.6 *If the samples set produced by the preprocessing stage is adequate, then the probability that the query algorithm outputs FAILURE is at most γ.*

A different type of analysis is presented in [41], where a framework based on probability spaces is introduced. From the two theoretical analysis briefly summarized, it is evident that while to a certain extent it is feasible to prove the probabilistic convergence of the PRM based algorithms, it is far from trivial to provide the numbers for a given environment. It is then still not possible to derive expectations on the required number of samples needed to achieve a given success probability while solving a generic RMP problem instance.

4 Tree based randomized motion planners

Samples and edges can also be organized in a tree rather than in a graph. By using this approach efficient planners have been designed. They are well suited for addressing *single-shot* motion planning problems, and while growing a tree it is possible to utilize the motion equations of the robot, thus obtaining paths complying with kinodynamic constraints.

4.1 Rapidly Exploring Random Trees

Rapidly Exploring Random Trees (RRT) are a class of RMP algorithms that can be used both for systems involving kinodynamic constraints or not [49],[50],[19],[40],[51],[52],[45]. RRT proved to be suitable for being used in very different real world applications [39],[11]. In addition to the *Check* and *Distance* routines used in the PRM framework, the RRT algorithm assumes the availability of the following elements:

- a set U of inputs to be applied to the system
- an incremental simulator, i.e. a procedure that given a state $x(t) \in \mathcal{X}$ and an input $u \in U$, produces the state $x(t+\Delta t)$, provided that the input u has been applied over the given time interval.

Then, by including system's equations into the incremental simulator, the planner is able to directly produce paths satisfying the kinodynamic constraints. If no such constraints are given, i.e. the planner is required to produce a path for a holonomic robot, no incremental simulation takes place and simple interpolation is performed, as every motion is allowed. The basic version of the algorithm is given in 7. The algorithm starts by building a tree rooted at the staring point x_{start}. Samples are then randomly generated over $\mathcal{C}_{free}$. If the incremental simulator is included, the *NEW_STATE* subroutine (called in line 3 of the *EXTEND* substep) chooses an input u, either randomly or by determining the one which will give a new state as closed as possible to the new random state, and determines a new state to be added to the tree. The routine returns *Reached* if the new sample can be reached, *Advanced* if it cannot be reached but a new state has been added, or *Trapped* if no new state has been produced. If the simulator is not needed, *NEW_STATE* simply tries to place the new state at a fixed distance from x_{near} along the segment connecting it with x (see figure 2). The rationale behind the RRT algorithm is the following (see figure 3). Let x be a node in the tree T and let $V(x)$ be its associated Voronoi region, i.e. the set of states nearer to x than to every other state in T. Then, by uniformly sampling over the state space, it is more likely to place samples into a big Voronoi region rather than into a small one. If a sample is placed into $V(x)$, then x

Algorithm 7 Algorithms for the construction of an RRT

```
1: BUILD_RRT(x_init)
2: T.init(x_init)
3: for k = 1 to K do
4:    x_rand ← RANDOM_STATE()
5:    EXTEND(T,x_rand)
6: return T
1: EXTEND(T, x)
2: x_near ← NEAREST_NEIGHBOR(x, T)
3: if NEW_STATE(x, x_near, x_new, u_new) then
4:    T.add_vertex(x_new)
5:    T.add_edge(x_near, x_new, u_new)
6:    if x_new = x then
7:       return Reached
8:    else
9:       return Advanced
10:   return Trapped
```

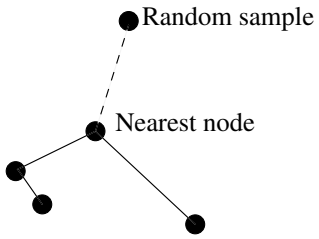

Figure 2. Extension of an RRT. Starting from a randomly sample x_{rand} generated over the state space X, the nearest RRT node is found (x_{near}), and a new node is created as its child. The new node is placed along the segment connecting x_{near} and x_{rand} if the system is holonomic, otherwise it is generated by applying the incremental simulator to x_{near}

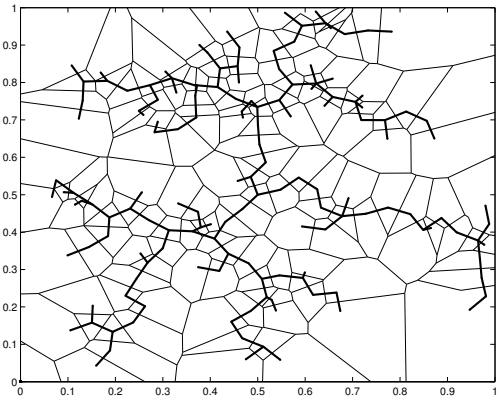

Figure 3. A RRT built over an obstacle free $[0,1]^2$ state space and the associated Voronoi regions

will be chosen to be extended. Thus the tree is biased to grow towards unexplored regions. The schema depicted in figure 7 simply builds a RRT which explores the state space starting from the given x_{start} point. If a couple of points is given in a *single shot* framework, a significant speedup is obtained by growing two trees, one from x_{start} and the other from x_{goal}. The *RRT-Connect* algorithm [40], designed for problems not involving kinodynamic constraints, exploits this technique, as well as a greedy tree extension to cut down planning time even more. Figure 8 illustrates this improved version. An ever more aggressive behavior can be obtained by using always the *CONNECT* routine instead of alternating it with *EXTEND*. This version of the planner is often indicated as *RRT-ConCon*. In this way, as soon as a promising direction is discovered, the tree is expanded in that direction as much as it is possible thus decreasing exploration time.

Algorithm 8 The RRT-Connect algorithm

```
1: CONNECT(T, q)
2: repeat
3:    S ← EXTEND(T,x_rand)
4: until NOT S =Advanced
5: return S
1: RRT_CONNECT(x_start, x_goal)
2: T_a.init(x_start)
3: T_b.init(x_goal)
4: for k = 1 to K do
5:    x ←RANDOM_CONFIG()
6:    if not (EXTEND(T_a,x) = Trapped ) then
7:       if CONNECT(T_b,x_new) = Reached then
8:          return PATH(T_a,T_b)
9:    SWAP(T_a,T_b)
10: return Failure
```

4.1.1 Probabilistic completeness of the RRT algorithm

The RRT algorithm has been proved to be probabilistic complete under rather mild hypothesis. As for the PRM algorithm, detailed proofs can be found in the aforementioned references, while here we provide just the results. Two different theorems are valid, one for holonomic systems and one for nonholonomic systems.

Theorem 4.1 *Let x_{init} and x_{goal} lie in the same connected component of a nonconvex, bounded, open, n-dimensional connected component of an n-dimensional state space. The probability that an RRT constructed from x_{init} will find a path to x_{goal} approaches one as the number of RRT vertices approaches infinity.*

A similar theorem holds for nonholonomic systems. In what follows it is assumed that in the *NEW_STATE* routine the input u is uniformly randomly chosen over the set of available inputs U.

Theorem 4.2 *In the same hypothesis of theorem 4.1, let further assume that $|U|$ is finite, Δt is constant, and no two RRT vertices lie within a specified $\varepsilon > 0$, according to the used metric* Distance. *Let also assume that there exists a sequence $u_1, \ldots, u_n$ of inputs that when applied to x_{init} will lead the system to the state x_{goal}. Then the probability that an RRT initialized at x_{start} will contain a vertex in the X_{goal} region approaches 1 as the number of vertices approaches infinity.*

While valuable in itself, probabilistic convergence only ensures convergence to the solution when the number of vertices, and then the computation time, approaches infinity. As with PRM, it would be highly useful to have a rate of convergence of the planner, in order to be able to predict the expected time. Again, while some results have been obtained, they are expressed in terms of environment specific quantities not easy to determine. The following theorems apply to single RRT and assume that an instance of the RMP problem is given in terms of a starting point x_{goal} and of a goal region X_{goal}, i.e. the robot is required to reach a region rather than a point. Both theorems rely on the following definition.

Definition 4.3 *A sequence of subsets $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ of the state space X is an attraction sequence if $A_0 = \{x_{init}\}$, $A_k = X_{goal}$, and for each A_i there exists a basin $B_i \subseteq X$ such that:*

1. *for all $x \in A_{i-1}$, $y \in A_i$, and $z \in X \setminus B$, the metric Distance yields $Distance(x, y) < Distance(x, z)$*
2. *for all $x \in B_i$, there exists a number m such that the sequence of inputs $\{u_1, u_2, \ldots, u_m\}$ selected by the EXTEND algorithm will bring the state into $A_i \subseteq B_i$*

Given a set S, we indicate with $\mu(S)$ its measure.

Theorem 4.4 *Let assume that a connection sequence $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ of length k exists, and let $p = min_i\{\mu(A_i)/\mu(X_{free})\}$. Then, the expected number of iterations required to connect x_{start} and X_{goal} is no more than k/p.*

Theorem 4.5 *If an attraction sequence of length k exists and $\delta \in (0, 1]$, then the probability that the RRT finds a path after n iterations is at least $1 - \exp(-np\delta^2/2)$, where $\delta = 1 - k/(np)$, and p is defined as in theorem 4.4.*

Both theorems 4.4 and 4.5 suffer from the dependence on k, the length of the attraction sequence. While given a solvable instance of the RMP problem one can assume the existence of such sequence, the number of elements in it depends heavily on the shape of the environment and is far from being easy to compute.

4.1.2 Remarks on the RRT algorithm

The *EXTEND* substep of the RRT algorithm starts by determining the nearest node to the last generated sample. This has at least two implications from a practical point of view. First, the search has to be performed over the whole set of nodes generated so far. If one does not adopt a suitable data structure, but rather scans the whole sequence, this will yield a quadratic dependence. This problem has been addressed in [5], where the problem of efficient neighbor search is tackled and a solution based on $Kd-$trees is proposed, yielding an overall $n \log n$ complexity. It has however to be pointed out that the techniques utilized are quite involved, and could take some effort to be implemented. While implementing this technique, one has additionally to take care that the topology of $\mathcal{C}$ is respected. The second important issue related to neighbor search is the metric used. The choice of a good metric is a fundamental problem, as the use of an inappropriate one could lead the tree to grow toward the wrong direction. Ideally, the value returned by the *Distance* function should reflect the *cost to go*, while moving from one state to another. It is evident that this cost depends on the underlying system model and that the metric should then be strictly related to it. This problem is well discussed in [18]. Recently, some possible extensions for the RRT algorithm were introduced. In [64], the idea of bidirectional search is pushed forward, towards multiple trees based search. These additional trees, called *local trees* are generated when a newly generated sample lies in free space, but cannot be connected to any of the already created trees. An upper bound on the number of trees is considered as well in order to eliminate the possibility to create too many trees. Experiments show that this expedient can significantly reduce the time needed to solve motion planning problems in maze like environments with narrow passages. A similar idea has also been proposed in [59], where nodes in the PRM roadmaps are substituted with trees. The authors also address how this schema can be efficiently implemented over distributed architectures in order to exploit the inherent parallelism.

4.2 Planning in expansive spaces

An approach similar to RRT have been proposed in [32], [38] [28] and [31]. As the RRT planner, this planner efficiently builds a tree data structure. We first illustrate the basic version which does not deal with kinodynamic constraints. Given an instance of the motion planning problem, the algorithm starts building two trees, one rooted at the start point x_{start}, and the other rooted at the goal point x_{goal}. As for the PRM algorithm, we assume that a tree T consists of a couple of sets $T = (V, E)$, where V is the set of tree nodes and E is the set of edges between nodes. The two trees are iteratively expanded by using the same algorithm, and the process terminates if it is possible to find a free path connecting the two trees. Algorithm 9 illustrates this iterative approach. The iteration terminates when either a solution is found or the maximum number if iterations is reached. Algorithm 10 illustrates how a

Algorithm 9 Expansive planner

```
1: Let T_1 be a tree rooted at x_start and with no other nodes
2: Let T_2 be a tree rooted at x_goal and with no other nodes
3: for MAX_ITERATIONS times do
4:    EXPANSION(T_1)
5:    EXPANSION(T_2)
6:    if CONNECT(T_1, T_2) then
7:       return the PATH connecting x_start and x_goal
8: return FAILURE
```

tree $T = (V, E)$ can be expanded. In what follows, let $B_d(s)$ be the ball of radius d centered in s. At each step the algorithm associates a *weight* with every node in V. The weight of the node $s \in V$ is the number of sampled nodes of V lying in $B_d(s)$, i.e.

$$w(x) = |V \cap B_d(s)|.$$

The goal of the weight function is to avoid oversampling in regions already explored and to rather bias the expansion towards unexplored areas of the configuration space. In this respect both RRT and the expansive planner aim to the same goal, the only difference being in the technique used to identified poorly explored zones. Finally, algorithm 11 illustrates how

Algorithm 10 Expansion algorithm

```
1: EXPANSION(T)
2: Pick a sample s from V with probability proportional to 1/w(s)
3: Let K be a set of N samples lying in B_d(s)
4: for all k ∈ K do
5:    compute w(k) and retain k with probability proportional to 1/w(k)
6:    if k is retained and k ∈ C_free and the segment (s, k) ∈ C_free then
7:       V ← V ∪ {s}
8:       E ← E ∪ {(s, k)}
```

connection between trees is verified. To limit the number of useless trials, the algorithm ignores nodes couples too far apart. The algorithm assumes that a path can be found if a couple of nodes is close enough. In that case the segment is stored so that it can be later used to produce the path connecting x_{start} with x_{goal}. The above algorithm can be adapted in order to deal with kinodynamic constraints. In this case, a single

Algorithm 11 Connection

```
1: CONNECT(T_1, T_2)
2: for all x ∈ V_1 do
3:   for all y ∈ V_2 do
4:     if Distance(x, y) < Threshold then
5:       if (x, y) ∈ C_free then
6:         store (x, y)
7:         return TRUE
8: return FALSE
```

tree is built, but the samples space is not X but rather

$$\mathcal{CT} = \mathcal{C}_{free} \times [0, +\infty]$$

which is called *space×time*. The subset of free valid configurations of $\mathcal{CT}$ is indicated as $\mathcal{CT}_{free}$. Along the same lines of the RRT algorithm, it is assumed the availability of an incremental simulator and of a set of inputs U_l. In this context it is assumed that inputs in U_l are piecewise constant functions with at most l pieces. Algorithm 12 illustrates how it is possible to generate a trajectory complying with the kinodynamic constraints. A problem instance is again formulated in terms of a start point and of a goal region, which the authors indicate call *ENDGAME* and will be here indicated as E. Extensive results over

Algorithm 12 Randomized kinodynamic motion planner

```
1: let T be a tree whose root is (x_start, 0)
2: for at most MAX_ITERATIONS times do
3:   Pick a sample s from V with probability 1/w(s)
4:   Pick an input u from U_l uniformly at random
5:   s' =INTEGRATE(s, u)
6:   if s' ∈ CT then
7:     V ← V{s'}
8:     E ← E{(s, s')}
9:   if s' ∈ E then
10:    Terminate with success
```

simulations and real robots are illustrated in [31]. The trials involved both nonholonomic robots and systems performing in dynamic environments, i.e. with moving obstacles. Detailed results provide evidence that the devised algorithms lead to real time compliant systems.

4.2.1 Probabilistic convergence

The former algorithms have been formulated in a framework based on the *expansiveness* concept introduced in [32]. We here report the generalized results illustrated in [31] which concerns the kinodynamic motion planner. Given (s,t) and $(s',t') \in \mathcal{CT}_{free}$ we say that (s',t') is reachable from (s,t) if there exists a control function that leads to an admissible trajectory from (s,t) to (s',t'). If such a trajectory can be obtained by applying just the inputs of the U_l set, then we say that (s',t') is l-reachable from (s,t). According to these definitions, it is possible to define the set of points *reachable* and *l-reachable* from a point $p = (s,t)$. We indicate the first set as $R(p)$ and the second as $R_l(p)$. Then, given a subset of $S \subset \mathcal{CT}_{free}$, it is possible to define its reachable sets:

$$R(S) = \bigcup_{p \in S} R(p)$$

$$R_l(S) = \bigcup_{p \in S} R_l(p)$$

Definition 4.6 *Let $\beta \in [0,1)$ be a constant and Let $S \subset \mathcal{CT}$. The* lookout *of the set S is*

$$Lookout_\beta(S) = \{p \in S | \mu(R_l(p)) \setminus S) \geq \beta \mu(R(s) \setminus S)\}$$

Definition 4.7 *Let α, β be constants in $[0,1]$. For any $p \in \mathcal{CT}_{free}$, $R(p)$ is* (α,β)-expansive *if for every connected subset $S \subseteq R(p)$,*

$$\mu(Lookout_\beta(S)) \geq \alpha \mu(S).$$

$\mathcal{CT}_{free}$ is (α,β)-expansive *if for every $p \in \mathcal{CT}_{free}$, $R(p)$ is (α,β)-expansive.*

The following theorem proves that the algorithm will reach the *ENDGAME* region, i.e. will succeed in finding a solution, with high probability.

Theorem 4.8 *Let $\mathcal{X}$ be the reachability of the start point (s,t) and let $g = \mu(E \cap \mathcal{X})$ be strictly positive. Let $\mathcal{X}$ be (α,β)-expansive. Let $\gamma \in (0,1]$ be a constant. Let T be a tree rooted at (s,t) with r nodes. The probability that T has a node in E is at least $1-\gamma$ if*

$$r \geq \frac{k}{\alpha} \ln \frac{2k}{\gamma} + \frac{2}{g} \ln \frac{2}{\gamma},$$

where $k = (1/\beta)\ln(2/g)$.

A set of similar results and definitions holds for the basic planner that does not deal with kinodynamic constraints. It is again evident that probabilistic convergence can be proved, but convergence rate is difficult to measure in terms of the problem instance to be solved.

5 Practical considerations

Given a motion planning problem, the choice of the planning algorithm to use is driven by different factors. The first aspect to consider is whether the problem involves kinodynamic constraints or not. If this is the case, the choice is for one of the the two algorithms illustrated in section 4. Up to now no analytic comparison is available, and also no fair experimental comparisons have been performed. On the other

hand both algorithms proved to be suitable for being used in real world applications, they address the same class of problems, and they require the same components. Certain authors report that expansive planners are more difficult to tune because of the higher number of parameters. It is however somehow difficult to give general indications on the one which could better fit the needs, or could be easier to implement. If the problem to be solved involves just kinematic constraints, then all the proposed algorithms can be used. In the single shot scenario, tree based algorithms are in general much faster. It has however to be pointed out that speed comes to the price of path quality, since these planners stop as soon as a path is found. PRM based planners, instead, can produce a set of paths, and then the most favorable one is returned. In a situation where many successive queries have to be solved, also the use of the basic PRM algorithm appears appropriate. If the operating environment exhibits a configuration space with narrow passages, then one of the outlined refined PRM algorithms is the choice. It is nevertheless evident that in general no algorithm is better, but rather the environment influences the performance. The opportunity of having more planners to be used in different regions is addressed in [21], but up to now no well defined heuristic is available to drive the choice.
Another important issue is the sampling and resampling strategy. The vast majority of the proposed planners propose uniform sampling over either $\mathcal{C}$ or a suitable subset. This leads to easy implementation, but has the outlined drawbacks. For what concerns resampling, associating a weight to each vertex and then choosing vertices to resample with a probability proportional to the inverse of the weight is easy to implement. The easiest weight to compute is the number of edges outgoing from a vertex. Also the weight suggested in algorithm 10 is easy to compute, but could take more time.

Collision detection is a challenging problem in itself, but fortunately there exist very efficient algorithms whose implementations are freely available to the scientific community. These algorithms often assume that the description of the objects is given in terms of meshes of triangles. This is a very favorable hypothesis, as many CAD systems export this type of representation for solid objects. However different algorithms exhibit different performances in various operating scenarios and a preliminary evaluation is needed in order to select the one better fitting the needs of the problem to be solved. The choice of the collision detector is extremely important, as most of the time spent by planners is devoted to collision checking, both for validating samples and edges connecting samples. Another important practical aspect concerns the *Distance* function. Since this is much easier to implement, it is practical to try different definitions and than rely on the one giving better experimental results. This is extremely important since sampling based algorithms use the Distance function in order to decide how the supporting data structure will be expanded. Common choices are the L_1, L_2 or L_∞ norms in the configuration space $\mathcal{C}$. While computing distances between configurations, it is also usual to assign different weights to the degrees of freedom, or to normalize them to a given common interval. This is usually the case when both translational and rotational joints are present, as rotating and translating can have a different impact on the geometry of the system.

6 Conclusions

We presented the most influential algorithms developed in the last years in the field of randomized robot motion planning. This paper illustrated the most common sampling based algorithmic techniques, namely graph based and tree based. The field is however continuously growing, and more and more refinements are being proposed, so that an exhaustive enumeration of the many possible variations is doomed to early obsolescence. The user who needs to implement them should have had concrete indications about their strength and limitations, and will not find too difficulties in adapting them to its specific needs.

References

[1] N.M. Amato, O. Burchand Bayazit, L.K. Dale, H. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. In P.K. Agarwal et al. editors, *Robotics : The Algorithmic Perspective. The Third Workshop on the Algorithmic Foundations of Robotics*, pages 156–168. A.K. Peters, 1998.

[2] N.M. Amato, O.B. Bayazit, L.K. Dale, and C. Jone. Choosing good distances metrics and local planners for probabilistic roadmap methods. *IEEE Transactions on Robotics and Automation*, 16(4):442–447, 2000.

[3] N.M. Amato and L.K. Dale. Probabilistic roadmaps are embarrassingly parallel. In *ICRA*, pages 688–694, 1999.

[4] N.M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *ICRA*, pages 113–120, 1996.

[5] A. Atramentov and S.M. LaValle. Efficient nearest neighbor searching for motion planning. In *ICRA*, pages 632–637, 2002.

[6] J. Barraquand, L. Kavraki, J. Latombe, T. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. *International Journal of Robotics Research*, 15(6):759–774, 1997.

[7] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *Inter-*

national Journal of Robotics Research, 10(6):628–649, 1991.

[8] J. Basch, L.J. Guibas, D. Hsu, and A.T. Nguyen Disconnection proofs for motion planning In *ICRA*, pages 1765–1772, 2001

[9] O. Burchan Bayazit, Jyh-Ming Lien, and Nancy M. Amato. Probabilistic roadmap motion planning for deformable objects. In *ICRA*, pages 2126–2133, 2002.

[10] R. Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *IROS*, pages 49 – 54, 2001.

[11] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *IROS*, pages 2383–2388, 2002.

[12] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge (MA), 1988.

[13] J. Canny. Some algebraic and geometric computations in pspace. In *Proc. of FOCS*, pages 460–467, 1988.

[14] S. Carpin and E. Pagello. Exploiting multi-robot geometry for efficient randomized motion planning. In Maria Gini et al., editor, *Intelligent Autonomous Systems 7*, pages 54–62. IOS Press, 2002.

[15] S. Carpin and E. Pagello. On parallel rrts for multi-robot systems. In *Proceedings of the 8th conference of the Italian Association for Artificial Intelligence*, pages 834–841, 2002.

[16] D. Challou, D. Boley, M. Gini, V. Kumar, and C. Olson. Parallel search algorithms for robot motion planning. In K. Gupta and A.P. del Pobil, editors, *Practical Motion Planning*, pages 115–132. John Wiley & Sons, 1998.

[17] P. Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *ICRA*, pages 267–272, 2002.

[18] P. Cheng and S.M LaValle. Reducing metric sensitivity in randomized trajectory design. In *IROS*, 2001.

[19] P. Cheng, Z. Shen, and S. M. LaValle. RRT-based trajectory design for autonomous automobiles and spacecraft. *Archives of Control Sciences*, 11(3-4):167–194, 2001.

[20] J. Cortes, T. Simeon, and J.P. Laumond. A random loop generator for planning the motions of closed kinematics chains using prm methods. In *ICRA*, pages 2141–2146, 2002.

[21] L.K. Dale and N.M. Amato. Probabilistic roadmaps - putting it all together. In *ICRA*, pages 1940–1947, Seoul, May 2001.

[22] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators. *Algorithmica*, 14(6):480–530, 1995.

[23] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamic bounds. *Algorithmica*, 14(6):443–479, 1995.

[24] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, November 1993.

[25] R. Geraerters and M.H. Overmars. A comparative study of probabilistic roadmap planners. In J.D. Boissonat et al. editors, *Workshop on Algorithmic Foundations of Robotics*, Advanced Robotics Series. Springer, 2002.

[26] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *ICRA*, pages 1718–1723, 1990.

[27] L. Han and N. Amato. A kinematics-based probabilistic roadmap method for closed chain systems. In B. Donald et al., editors, *Algorithmic and Computational Robotics - New Directions*, pages 233–246. A.K. Peters, 2000.

[28] D. Hsu. *Randomized single-query motion planning in expansive spaces*. PhD thesis, Department of Computer Sciece, Stanford University, 2000.

[29] D. Hsu, L.E. Kavraki, J-C. Latombe, and R. Motwani. Capturing the connectivity of high-dimensional geometric spaces by parallelizable random sampling techniques. In P.M. Pardalos and S. Rajasekaran, editors, *Advances in Randomized Parallel Computing*, pages 159–182. Kluwer Academic Publishers, Boston, MA, 1999.

[30] D. Hsu, L.E. Kavraki, J.C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P.K. Agarwal et al. editors, *Robotics : The Algorithmic Perspective.The Third Workshop on the Algorithmic Foundations of Robotics*, pages 142–153. A.K. Peters, 1998.

[31] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motiona planning with moving obstacles. In B. Donald et al. editors, *Algorithmic and Computational Robotics - New Directions*. A.K. Peters Ltd, 2000.

[32] D. Hsu, J-C. Latombe, and R. Motwani. Path planning and expansive configuration spaces. *International Journal of Computational Geometry and Applications*, 9:495–512, 1999.

[33] Y.K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computational Surveys*, 24(3):219–290, 1992.

[34] L. Kavraki and J.-C Latombe. Randomized preprocessing of configuration space for fast path planning. In *ICRA*, pages 2138–2145, 1994.

[35] L.E. Kavraki, M.N. Kolountzakis, and J.C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, 1998.

[36] L.E. Kavraki, J.C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and Systems Science*, 57(1):50–60, 1998.

[37] L.E. Kavraki, P. Švestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[38] R. Kindel, D. Hsu, J.C. Latombe, and S. Rock. Kinodynamic motion planning admnist moving obstacles. In *ICRA*, pages 537–543, 2000.

[39] J. Kufner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *ICRA*, pages 692–698, 2001.

[40] J.J. Kufner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *ICRA*, pages 995–1001, San Francisco, April 2001.

[41] A. Ladd and L. Kavraki. Generalizing the analysis of prm. In *ICRA*, pages 2120–2125, 2002.

[42] F. Lamiraux and L.E. Kavraki. Planning paths for elastic objects under manipulation constraints. *International Journal of Robotics Research*, 20(3):188–208, 2001.

[43] J.C. Latombe. *Robot Motion Planning*. Kluver Academic Publishers, 1990.

[44] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research - Special Issue on Robotics at the Millennium*, 18(11):1119–1128, 1999.

[45] S. M. LaValle. From dynamic programming to RRTs: Algorithmic design of feasible trajectories. In A. Bicchi et al. editors, *Control Problems in Robotics*, pages 19–37. Springer-Verlag, Berlin, 2002.

[46] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. In J.-D. Boissonat et al. editors, *Algorithmic Foundations of Robotics*. Springer-Verlag, Berlin, 2003. To appear.

[47] S.M. LaValle. Msl - the motion strategy library software, version 2.0. http://msl.cs.uiuc.edu.

[48] S.M. LaValle. *Planning Algorithms*. Available Online.

[49] S.M. LaValle. Rapidly-exploring random trees. Technical Report TR 98-11, Computer Science Department Iowa State University, October, 1998.

[50] S.M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *ICRA*, pages 473–479, 1999.

[51] S.M. LaValle and J.J. Kufner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[52] S.M. LaValle and J.J. Kufner. Rapidly-exploring random trees: Progress and prospects. In B. Donald et al. editors, *Algorithmic and Computational Robotics: New Directions*, pages 45–59. A.K. Peters, 2001.

[53] J.-M. Lien, S. L. Thomas, and N.M. Amato. A general framework for sampling on the medial axis of the free space. In *ICRA*, 2003.

[54] S. R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. In *Proc. Eighth Int'l Symp. on Robotics Research*. Springer-Verlag, 2004.

[55] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

[56] E. Mazer, J.M. Ahuactzin, and P. Bessiére. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, 1998.

[57] C. Nissoux, T. Siméon, and J.P. Laumond. Visibility based probabilistic roadmaps. In *IROS*, pages 1316–1321, 1999.

[58] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K. Goldberg et al. editors, *Algorithmics Foundations of Robotics*. A K Peters Ltd., 1995.

[59] E. Plaku and L.E. Kavraki. Distributed sampling-based roadmap of trees for large-scale motion planning. In *ICRA*, pages 3879–3884, 2005

[60] J.H. Reif. Complexity of the mover's problem and generalization. In *Proc of FOCS*, pages 421–427, 1979.

[61] G. Sánchez and J.C Latombe. Using a prm planner to compare centralized and decoupled planning for multi-robot systems. In *ICRA*, 2002.

[62] J.T. Schwartz and M. Sharir. Algorithmic motion planning in robotics. In *Handbook of theoretical computer science*, volume 1, chapter 8, pages 392–430. Elsevier, 1990.

[63] F. Schwarzer, M. Saha, and J.C. Latombe. Exact collision checking of robot paths. In J.D. Boissonat et al. editors, *Workshop on Algorithmic Foundations of Robotics*, Advanced Robotics Series. Springer, 2002.

[64] M. Strandberg Augmenting RRT-planners with local trees In *ICRA*, pages 3258–3262, 2004

[65] P. Švestka and M.H. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23(3):125–152, 1998.

[66] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. In *Proceedings of the 15th Annual ACM Symposium on Computational Geometry*, pages 173–180, 1999.

[67] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. A probabilistic roadmap planner with sampling on the medial axis of the free space. In *ICRA*, pages 1024–1031, 1999.

[68] J.H. Yakey, S.M. LaValle, and L.E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–958, 2001.