CHAPTER
TWO

PERFORMANCE-BASED COMPARISON

The Probabilistic Roadmap Method (PRM) is one of the leading motion planning techniques. Over the past fifteen years, the technique has been studied by many different researchers. This has led to many variants of the method, each with its own merits. It is difficult to compare the different techniques because they were tested on different types of environments, using different underlying libraries, implemented by different people on different machines.

We provide a comparative study of a number of these techniques, all implemented in a single system and run on the same test environments and on the same computer. In particular we compare *collision checking* techniques, *neighbor selection* techniques and *sampling* techniques. The results are surprising in the sense that techniques often perform differently than claimed by the designers.

The study also shows how difficult it is to evaluate the quality of the techniques. First, it is not necessarily true that combining different 'good' techniques leads to a good overall result. Second, many techniques have high variances in running time which is undesirable as a large variation complicates statistical analysis and can even make it unreliable. It is also undesirable from a users point of view, e.g. in a virtual environment where real-time behavior is required, only a particular amount of CPU time is scheduled for the motion planner. The variance can be reduced by properly choosing the techniques and their parameters.

The results of this study should help future users of the Probabilistic Roadmap Method in deciding which technique is suitable for their situation.

2.1 Introduction

Over the years, many different approaches to solving the motion planning problem have been suggested. See the books of Choset *et al.* [36], Latombe [99] and LaValle [100] for an extensive overview. A popular motion planning technique is the Probabilistic Roadmap Method (PRM), developed independently at different sites [4, 6, 80, 81, 83, 84, 124, 151]. The method turns out to be very efficient, easy to implement, and applicable to many different types of motion planning problems (see e.g. [16, 20, 26, 40, 66, 68, 77, 85, 97, 107, 117, 130, 144, 151–153]).

Globally speaking, the PRM samples the configuration space for collision-free configurations. These are added as nodes to a roadmap graph. Pairs of promising nodes are chosen in the graph and a simple local motion planner is used to try to connect such placements with a path. If successful, an edge is added between the nodes in the graph. This process continues until the graph represents the connectedness of the space.

The basic PRM leaves many details to be filled in, like how to sample the space, what local planner to use and how to select promising pairs. Over the past decade, researchers have investigated these aspects and developed many improvements over the basic scheme (see e.g. [3, 15, 20, 21, 23, 25, 69, 70, 85, 94, 121, 130, 131, 152, 156]). Unfortunately, the different improvements suggested are difficult to compare. Each author uses his or her own implementation of the PRM and uses different test scenes, both in terms of environment and the type of moving device (robot). Also the effectiveness of a technique sometimes depends on choices made for other parts of the method. Therefore, it is still rather unclear what is the best technique under which circumstances. See [42] for a first study of this issue.

In this chapter, we will compare different techniques developed. We implemented many different techniques in a single motion planning system and added software to compare the approaches. In particular, we concentrated on approaches checking local paths for collisions, the choice of promising pairs of nodes and the sampling technique. This comparison gives insight into the relative merits of the techniques and the applicability in certain types of motion planning problems. In addition, we hope that in the longer term our results will lead to improved (combinations of) techniques and adaptive approaches that choose techniques based on observed scene properties.

The chapter is organized as follows. In Section 2.2, we review the basic PRM. In Section 2.3, we describe our experimental setup and the environments we use. In Section 2.4, we compare different ways of performing collision checks

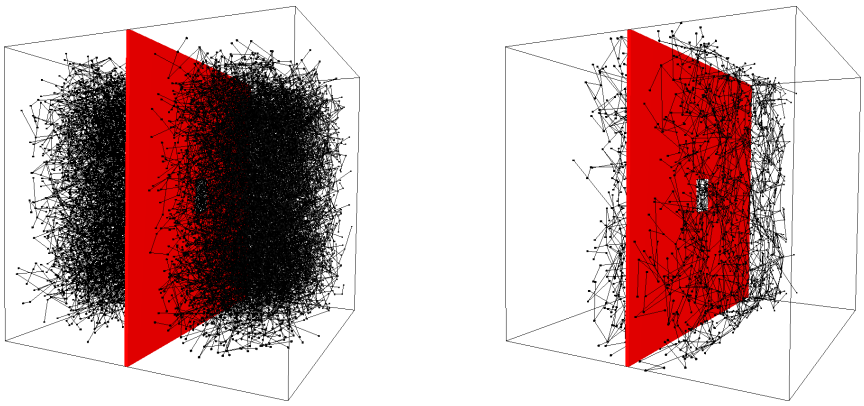
of the paths produced by the local planner. We will conclude that a binary approach performs best. In Section 2.5, we study different strategies for choosing promising pairs of nodes to connect. We will conclude that a technique based on connecting a new configuration to the nearest-k configurations works relatively well. In Section 2.6, we consider five different uniform sampling strategies. We conclude that, except for very special cases, a deterministic approach based on Halton points can be best used, although the differences between the methods are small. In Section 2.7, we consider six non-uniform sampling techniques that have been designed to deal with the so-called *narrow passage* problem and will conclude that these techniques should only be used in parts of the workspace containing narrow passages, i.e. they handle the test environment with one very narrow passage faster than uniform sampling, but are often much slower on all other environments. Finally, we draw conclusions in Section 2.8.

2.2 The Probabilistic Roadmap Method

The motion planning problem is usually formulated in terms of the *configuration space* $\mathcal{C}$, the space of all possible placements of the moving object. Each degree of freedom (DOF) of the object corresponds to a dimension of the configuration space. Each obstacle in the workspace, in which the object moves, transforms into an obstacle in the configuration space. Together they form the forbidden part $\mathcal{C}_{\text{forb}}$ of the configuration space. A path for the moving object corresponds to a curve in the configuration space, connecting the start and the goal configuration. A path is collision-free if the corresponding curve does not intersect $\mathcal{C}_{\text{forb}}$, that is, it lies completely in the free part of the configuration space, denoted by $\mathcal{C}_{\text{free}}$.

The PRM samples the configuration space for free configurations and tries to connect these configurations to a roadmap of feasible motions. There are several versions of the PRM, but they all use the same underlying concepts.

The global idea of the PRM is to pick a collection of (useful) configurations in the free space $\mathcal{C}_{\text{free}}$. These free configurations form the nodes of a graph $G = (V, E)$. A number of (useful) pairs of nodes are chosen and a simple local motion planner is used to try to connect these configurations by a path. When the local planner succeeds, an edge is added to the graph. The local planner must be fast (since checking local paths for collisions is the most time-consuming part of the PRM), but is allowed to fail on difficult instances. A typical choice is to interpolate between the two configurations, and then check whether the path is collision-free. So the path is a straight line in $\mathcal{C}$-space.



(a) Halton sampling (b) Gaussian sampling

Figure 2.1 The roadmap graph we get for the difficult Hole test environment used in this chapter. The left image shows the graph using Halton sampling (20,002 nodes and 20,019 edges) and the right image shoes the graph using Gaussian sampling (2,428 nodes and 2,425 edges).

Once the graph reflects the connectivity of $\mathcal{C}_{\text{free}}$ it can be used to answer motion planning queries. See Figure 2.1 for an example of the roadmap graphs computed. To find a motion between a start configuration and a goal configuration, both are added to the graph using the local planner. (Some authors use more complicated techniques to connect the start and goal to the graph, e.g. bouncing motions [124].) Then a path in the graph is found which corresponds to a motion for the object. The pseudo-code for the algorithm for constructing the graph is shown in Algorithm 2.1.

Algorithm 2.1 CONSTRUCTROADMAP

Require: $V \leftarrow \emptyset$; $E \leftarrow \emptyset$;

1: **loop**
2: $c \leftarrow$ a (useful) configuration in $\mathcal{C}_{\text{free}}$
3: $V \leftarrow V \cup \{c\}$
4: $N_c \leftarrow$ a set of (useful) nodes chosen from V
5: **for all** $c' \in N_c$, in order of increasing distance from c **do**
6: **if** c' and c are not connected in G **then**
7: **if** the local planner finds a path between c' and c **then**
8: add the edge $c'c$ to E

Note that in this version of the PRM, we only add an edge between nodes if they are not in the same connected component of the roadmap graph. This saves time because such a new edge will not help solving motion planning queries. Hence, we will not add these additional edges in this comparative study. However, to get short paths, such extra edges can be useful (see Section 7.2).

In this study, we concentrate on various choices for picking useful samples (line 2 of the algorithm), for picking useful pairs of nodes for adding edges (that is, on the choice of N_c in line 4) and for collision checking those edges (line 7). These are the most crucial steps and they strongly influence the running time and the structure of the roadmap graph.

In this study, we focus on multiple shot techniques and will not consider single shot methods such as RRT-based planners [94, 130].

2.3 Experimental setup

In this study we restrict ourselves to free-flying objects in a three-dimensional workspace. Such objects have six degrees of freedom (three translational and three rotational). In all experiments (except for the *rotate-at-s* local planner in the next section), we use the most simple local method that consists of a straight-line motion in configuration space. For other types of devices or local planners the results might be different. This will be investigated in Chapter 3.

The PRM builds a roadmap, which, in the query phase, is used to solve motion planning problems. We aim at computing a roadmap that covers the free space adequately but this is difficult to test.[1] Instead, in each test environment we define a relevant query and continue building the roadmap until the query configurations are in the same connected component.

For the experiments we use our SAMPLE system. In all experiments we report the running time in seconds. Because the experiments are conducted under the same circumstances, the running time is a good indication of the efficiency of the technique. For those techniques where random choices are involved, we report statistics gathered from 100 independent runs. See Section 1.2 for more information on our general experimental setup.

For the experiments we use the six environments depicted in Figure 2.2, all representing different types of problems. The Cage and Wrench environments

[1] Actually, this is too expensive to test for problems involving more than three degrees of freedom. In Chapter 3, we use a termination criterion that is based on coverage (and connectivity) of the free configuration space.

have been taken from the Motion Strategy Library [100]. To make extensive experimentation possible, we do not include huge environments such as those common in CAD environments. The environments have the following properties (see Table 2.1 for their dimensions).

Cage This environment consists of many primitives. The flamingo (7,049 polygons) must navigate from inside the cage (1,032 triangles) to outside the cage. The complexity of this environment will put a heavy load on the collision checker but the paths are relatively easy.

Clutter This environment consists of 500 uniformly distributed tetrahedra. A torus must move among them from one corner to the other. The configuration space will consist of many corridors. There are many solutions to the query.

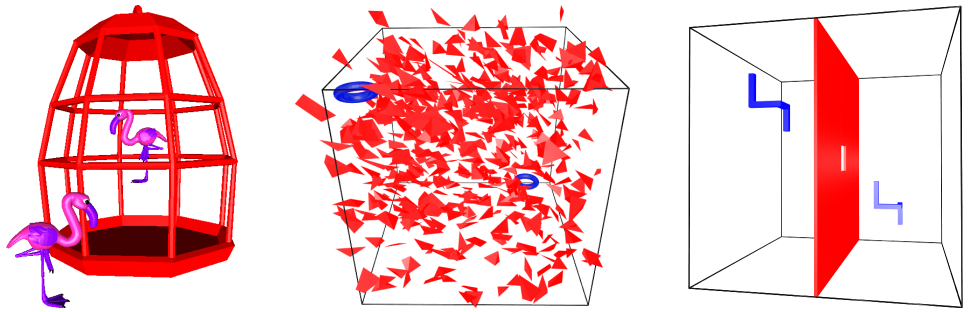
Hole The moving object consists of four legs and must rotate in a complicated way to get through the hole. The hole is placed off-center to avoid that certain grid-based sampling methods have an advantage. The configuration space will have two large open areas with two narrow winding passages between them.

House The house is a relatively complicated environment consisting of approximately 2,200 polygons. The moving object (table) is small compared to the house. Because walls are thin, the collision checker must make rather small steps along the paths, resulting in higher collision checking times. Because of the many different parts in the environment, the planner can be lucky or unlucky in finding relevant parts of the roadmap quickly. Therefore, we expect large differences in the running times of different runs.

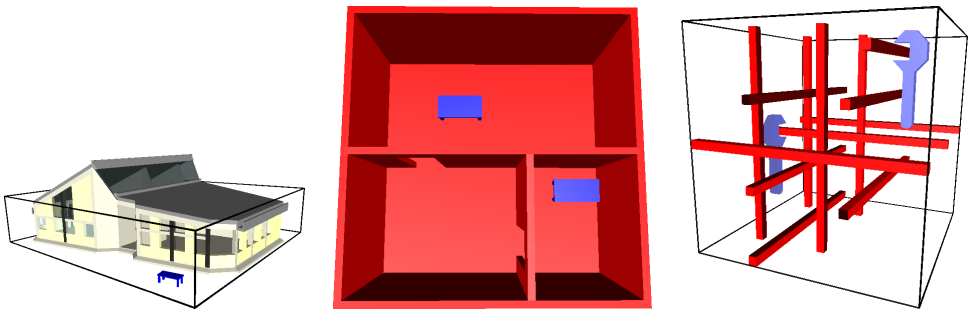
Rooms In this environment there are three rooms with ample free space and with narrow doors between them. So the density of obstacles is rather non-uniform. The table must navigate through the two narrow doors to the other room.

Wrench This environment features a large moving object (156 triangles) in a small workspace (48 triangles). There are many different solutions. The object is rather constrained at the start and goal.

We use the distance metric described in Section 1.2.3. Table 2.2 enumerates the weights for the translational and rotational DOFs. The step sizes for the local planner are listed in Table 2.3.



(a) Cage (b) Clutter (c) Hole



(d) House (e) Rooms (f) Wrench

Figure 2.2 The six environments used for testing.

	Dimensions of the bounding boxes	
	environment	**robot**
Cage	$45 \times 40 \times 45$	$4 \times 13 \times 10$
Clutter	$48 \times 48 \times 48$	$8 \times 1.5 \times 8$
Hole$^{(*)}$	$40 \times 40 \times 40$	$5 \times 5 \times 10$
House	$35 \times 8 \times 38$	$2.5 \times 0.5 \times 1.5$
Rooms	$40 \times 20 \times 40$	$8 \times 2.5 \times 4$
Wrench	$160 \times 160 \times 160$	$68 \times 24 \times 8$

Table 2.1 The axis-aligned bounding boxes of the environments and robots. $^{(*)}$The dimensions of the hole are $5 \times 5 \times 0.5$. The legs of the robot are 1 thick.

	Weights for the DOFs of a robot	
	translational	**rotational**$_3$
Cage	1, 1, 1	30
Clutter	1, 1, 1	7
Hole	1, 1, 1	11
House	1, 1, 1	4
Rooms	1, 1, 1	9
Wrench	6, 6, 6	30

Table 2.2 The weights for each DOF of the robots.

	step size
Cage	1.0
Clutter	1.0
Hole	1.0
House	0.5
Rooms	1.0
Wrench	3.0

Table 2.3 The step sizes used by the local planners.

2.4 Collision checking

The most time-consuming steps in the Probabilistic Roadmap Method are the collision checks that are required to decide whether a sample lies in $\mathcal{C}_{\text{free}}$ and whether the motion produced by the local planner is collision-free. In particular the second type of checks is time-consuming. In this section we investigate some techniques for collision testing of the paths.

When testing a path for collisions we can use one of the following three techniques:

incremental In the incremental method we take small steps along the path from start to goal. Let d be the distance between the start and goal, then the number of steps equals to d divided by the appropriate step size from Table 2.3. We check each step (i.e. the placement of the robot) for collisions with the environment.

binary In the binary method we start by checking the middle position along the path. If it is collision-free we recurse on both halves of the path, checking the middle positions there. In this way, we continue until either a collision is found or the checked placements lie close enough together (again determined by the given step size) [130].

rotate-at-s While the previous methods check collisions along a straight line in the $\mathcal{C}$-space, the rotate-at-s approach first translates from start to an intermediate configuration s halfway, then rotates, and finally translates to the goal [2]. We set s to 0.5, i.e. halfway the line.

The incremental method was used in early papers on the PRM. Later papers suggest that the binary method works better [131]. The reason is that the middle position is the one that has the highest chance of not being collision-free. This means that, when the path is not collision-free, a collision will most likely be found earlier, that is, after fewer collision checks.

It has been suggested that one should try to compute sweep volumes and use these for collision tests. As a result, a path check would require just one collision test. Unfortunately, it is very difficult to compute sweep volumes for three-dimensional moving objects with six degrees of freedom. A much simpler technique is to first check with the sweep volume induced by the origin of the object, that is, with a line segment between start and goal position, see [41].

preceding line check In this method we first perform a collision test with the line segment between the start and goal position in the workspace. Only

	Collision checking without line check		
	incremental	**binary**	**rotate-at-*s***
Cage	2.4	**1.5**	4.2
Clutter	2.3	**1.9**	6.5
Hole	226.7	**213.6**	929.5
House	5.5	**5.1**	14.3
Rooms	**0.3**	**0.3**	0.9
Wrench	1.4	**1.1**	4.6

	Collision checking with line check		
	incremental	**binary**	**rotate-at-*s***
Cage	2.4	**1.5**	3.9
Clutter	**2.4**	**2.4**	5.8
Hole	**186.2**	213.7	979.2
House	6.6	**6.3**	13.9
Rooms	**0.2**	**0.2**	1.0
Wrench	1.5	**1.1**	4.6

Table 2.4 Average running times for three different collision checking methods. Each method was tested without and with line check.

if it is collision-free, we perform one of the three techniques. Note that this assumes that the origin of the object lies inside it.

We would expect that this test will quickly discard many paths that have a collision, leading to an improvement in running time.

We conducted experiments with the three collision-check techniques. Each experiment was run 100 times. To determine whether first performing a collision test with an appropriate line segment decreases the running time, we tested each technique with and without line check. Figure 2.3 and Table 2.4 summarize the results (using Halton* points for sampling and a simple nearest-k node adding strategy[2]; see below). Note that the best times are indicated in bold.

They show that in all environments the binary approach was faster than the incremental approach although the improvement varied over the type of environment. Also the standard deviation of this approach was smaller, making it

[2]The values used for the two parameters of this node adding strategy are stated in Table 2.5.

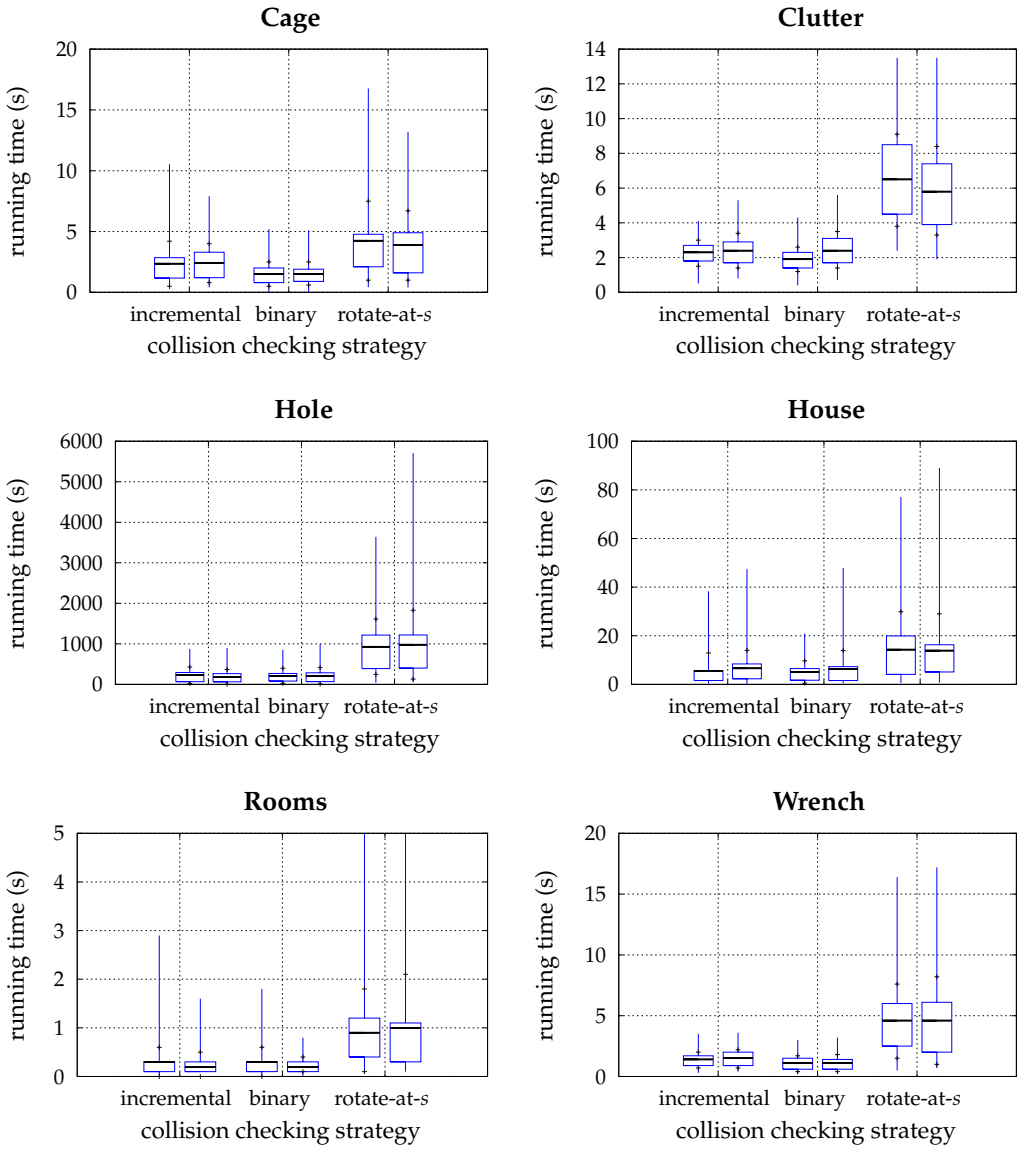


Figure 2.3 Relation between the *collision checking strategy* and the running time. Box plots are shown for each environment. Each strategy was tested without (left box plots) and with a line check (right box plots). Each plot consists of a box (the middle 50% of the data), one large horizontal line (average) and two small horizontal lines (average $\pm$ standard deviation) and a vertical line (minimum and maximum value).

more robust against outliers. The line check only had a marginal effect, contrary to the claim in [41]. This might be due to the way Solid does the collision checking with line segments. In [41], it was suggested to only apply the line check when the distance between the endpoints is large. We tried this but did not see any significant improvement in performance.

Also the rotate-at-s technique did not give the improvement suggested in [2]. Actually, its average running times were three times as large as the running times of the binary approach. Also its standard deviation was much larger. However, it should be noted that this can depend on the underlying collision checking package used. For the rest of this chapter we will use the binary approach without line checks.

2.5 Neighbor selection strategy

The neighbor selection strategy specifies for a particular sample how a set of neighbor samples is chosen to which it is connected using a local planner. The goal of the strategy is to make the graph connected as fast as possible. A strategy usually selects neighbors based on the *maximum connection distance*, *maximum number of connections* tried, and the *connected components* in the graph.[3] We study the effects of different choices for these criteria.

It is recognized that the maximum connection distance should not be too small nor too large: Although making long connections seems to be important for the PRM, it is in general not useful to make very long connections since the chance of success for such connections is small while the collision checks required for testing the local path are expensive. On the other hand, a very small connection distance will always require an exponential number of samples. We will show how to choose the maximum connection distance in Section 2.5.1.

In Section 2.5.2, we will study how to choose the maximum number of connections. If this number is too small, then it might be hard to get the free space connected because the chance is small that those few nodes are selected to which a connection is possible. If connections are tried with too many nodes, the total number of nodes will be less, but this might negatively influence the running time as testing those connections is expensive.

[3] If these parameters are not set to infinity (which is the default case), then we have to find the nearest-k neighbors. This is done by calculating the distance to each of the n nodes in the graph. A node is stored in a sorted list of maximum size k if the distance to this node is less than some maximum distance. This approach takes $O(n \log k)$ time. More efficient nearest-neighbor searching algorithms based on kd-trees are discussed in [158].

In Section 2.5.3, we will compare five node adding strategies. These strategies do not produce graphs with cycles. It is not useful (from a complexity point of view) to make connections to nodes that are already in the same connected component as such a new connection will not help solving motion planning queries.

2.5.1 Maximum connection distance

Before we can conduct experiments to determine the maximum connection distances, we have to make some choices for the PRM. We use the nearest-k node adding strategy. We set parameter k (which denotes the maximum number of connections) to 25 as this seems to work reasonably. Furthermore, we use the Halton* sampling strategy (see below).

In each of the following experiments, we vary the maximum connection distance from a small value (close to zero) to a large value. For each distance value, we perform 100 runs and gather the following running time statistics: the middle 50% of the data, the average, the standard deviation, the minimum and the maximum. See Figure 2.4 for the results.

When we make the connection distance very small, the PRM starts looking like grid-based techniques in which nodes are only connected to their direct neighbors. Figure 2.4 shows that this considerably increases the running time. Indeed, the power of PRM is that it can make longer connections. A relatively small value is only useful in environments were the connections are expected to be short. When we make the connection distance large (or even set this parameter to ∞), the average running times are in general higher than the running time corresponding to the optimal maximum connection distance. A larger value is best for environments in which long connections can be made and where the weights, used by the metric, are large. (For example, as the rotational DOFs play an important role in the Cage and Wrench environments, the corresponding weights for these DOFs are large.) Hence, the optimal value is correlated with the average visibility of the nodes and the weights used by the metric. Moreover, there is some optimal trade-off. See Table 2.5 for the optimal values.[4] These values will be used in the remainder of the chapter.

The box plots of Figure 2.4 reveal a problem when dealing with statistical analysis of data produced by the PRM. They show that there can be a large difference between the minimum and maximum running times. Furthermore, the large sizes of the boxes show that there is a large variance in the running

[4]For other types of problems, such as problems involving car-like or articulated robots, we also advise not to use a small maximum connection distance.

	Neighbor selection parameters	
	max. connection distance	**max. number of connections**
Cage	45	75
Clutter	15	75
Hole	10	75
House	15	75
Rooms	20	75
Wrench	350	75

Table 2.5 The optimal values used in the neighbor selection strategy.

times. This phenomenon is undesirable because of two reasons. First, a large variation complicates statistical analysis and can even make it unreliable. Second, it is undesirable from a users point of view, e.g. it can be hard to give a user an indication of how long the method will take to terminate. Hence, we have to be very careful analysing the results. As the running times can vary extensively, we performed 100 runs for each experiment. In this way we can increase its statistical significance. We noticed that some authors choose a low number of runs per experiment while only mentioning the average. Because of the large variance between runs that we observe, we think that this reduces the validity of the claims that are made.

2.5.2 Maximum number of connections

In the following experiments, we will vary the maximum number of connections k to determine the optimal value for each environment. For each k, we perform 100 runs and create a corresponding box plot. As maximum connection distance, we use the results of our experiments of the previous section as shown in Table 2.5. We again use the Halton* sampling strategy (see below). Figure 2.5.2 shows the relation between the maximum number of connections and the running time and Figure 2.6 shows the effect on the number of nodes.

The results show that a small maximum number of connections (e.g. $k < 5$) leads to high running times and high peaks. Corresponding graphs contain a huge number of nodes; they are typically one order of magnitude larger than graphs corresponding to the optimal value of k. As only a few connections per new node are tried, chances are small that the node will get connected to other nodes.

It is not useful to us a k larger than the (expected) number of nodes in the graph. In the Cage experiment for example, we observe that the average num-

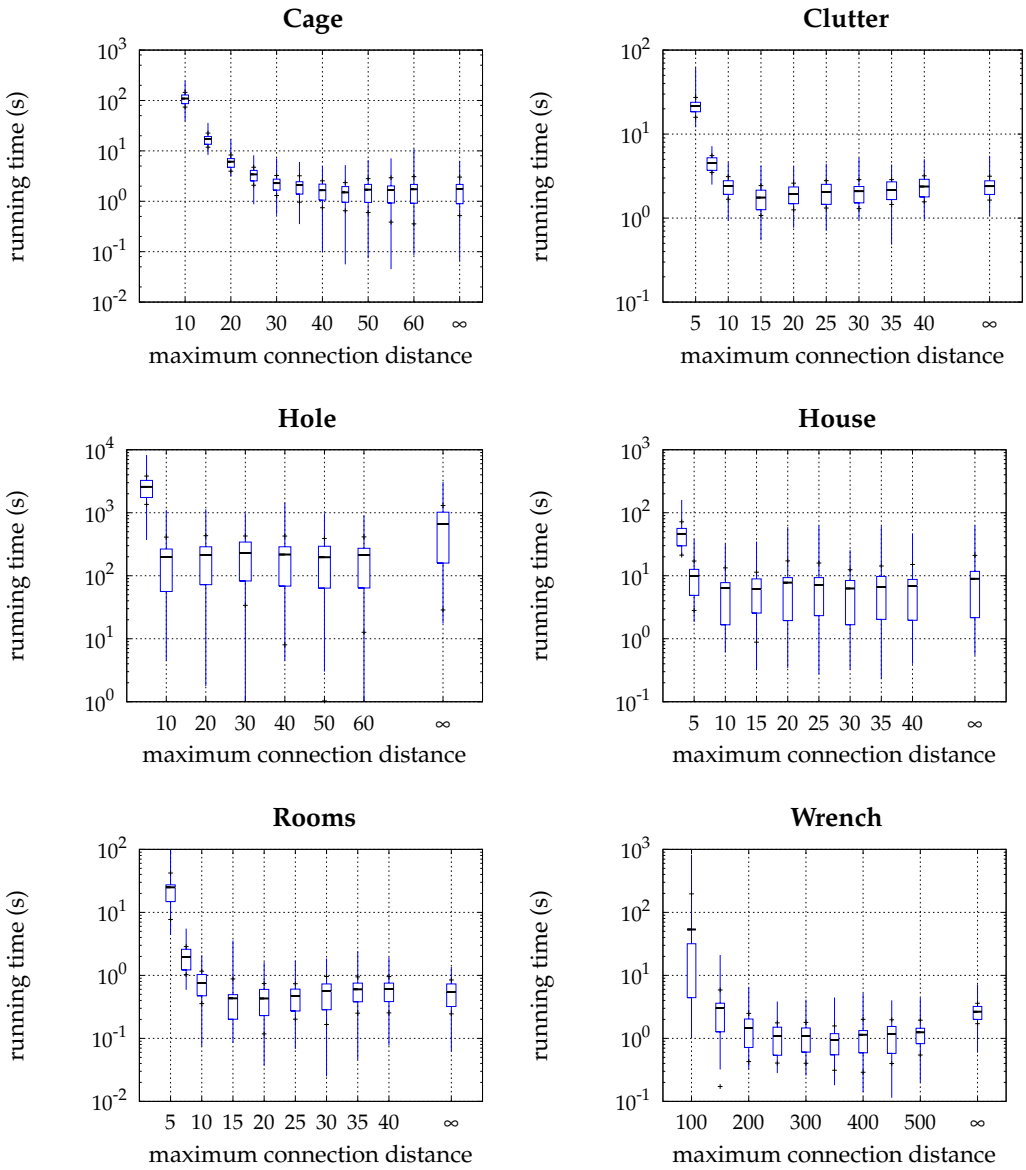


Figure 2.4 Relation between the *maximum connection distance* and the running time. A series of box plots is shown for each environment. Note that we use a logarithmic scale for the 'running time'-axis.

ber of nodes in the graph is about 40 when k is set to at least 20. Using a value larger than 40 will not lead to extra collision checks or distance calculations. Hence, this will not influence the running times. Against our expectations, a high number of maximum connections had only a marginal effect on the running times (compared to the optimal running times). This can be explained as follows.

In order to find a path, at least the following criterion must hold: each configuration on the path must be in at least one visibility area of a sample/node in the graph. In other words, a set of nodes should cover the path. However, when these nodes belong to different connected components, the path cannot be found (yet). In the next chapter we will show that connecting the different connected components is more complicated than satisfying the coverage criterion. This especially holds for narrow passage problems which arise e.g. in the Hole environment, but also in the House and Rooms environments. When more effort is put in connecting the components, i.e. the maximum number of connections is increased (up to a certain level), the problem is solved in less time.

When a new node ν is connected to more nodes (up to a certain level), the chance is larger that different connected components will be merged, but this may take more time for collision checking. Note that the amount of extra coverage of ν is not affected by the maximum number of connections. (After adding some n nodes to the graph, the path will be covered.) On the other hand, when ν is connected to less nodes, the chance is smaller that different connected components will be merged, and hence, more than n nodes are needed. Although these extra nodes do not contribute to the coverage anymore, their only goal is to connect the components. As a smaller k complicates this task, too much time may be spent on checking the connections for collisions while the node may be less useful. We conclude that the extra time needed to check more connections saves computation time compared to the time needed for adding a larger number of useless nodes.

We advise to use a high value (e.g. 75) for the maximum number of connections.

2.5.3 Node adding strategies

We will now compare some node adding strategies. We consider the following techniques:

nearest-k We try to connect the new configuration to the nearest k nodes in the graph that lie close enough. The rationale is that nearby nodes result in

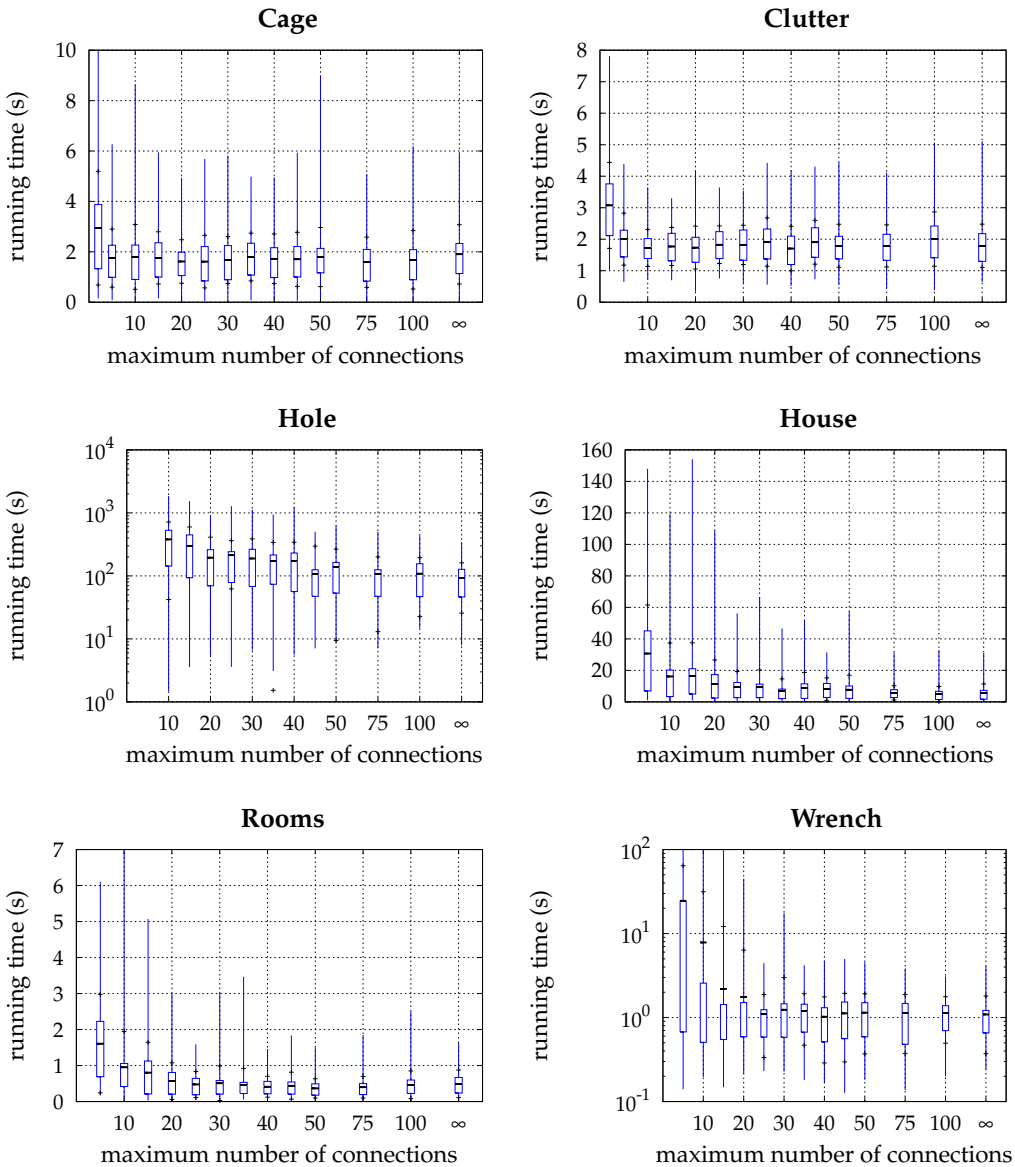


Figure 2.5 Relation between the *maximum number of connections* and the running time. A series of box plots is shown for each environment.

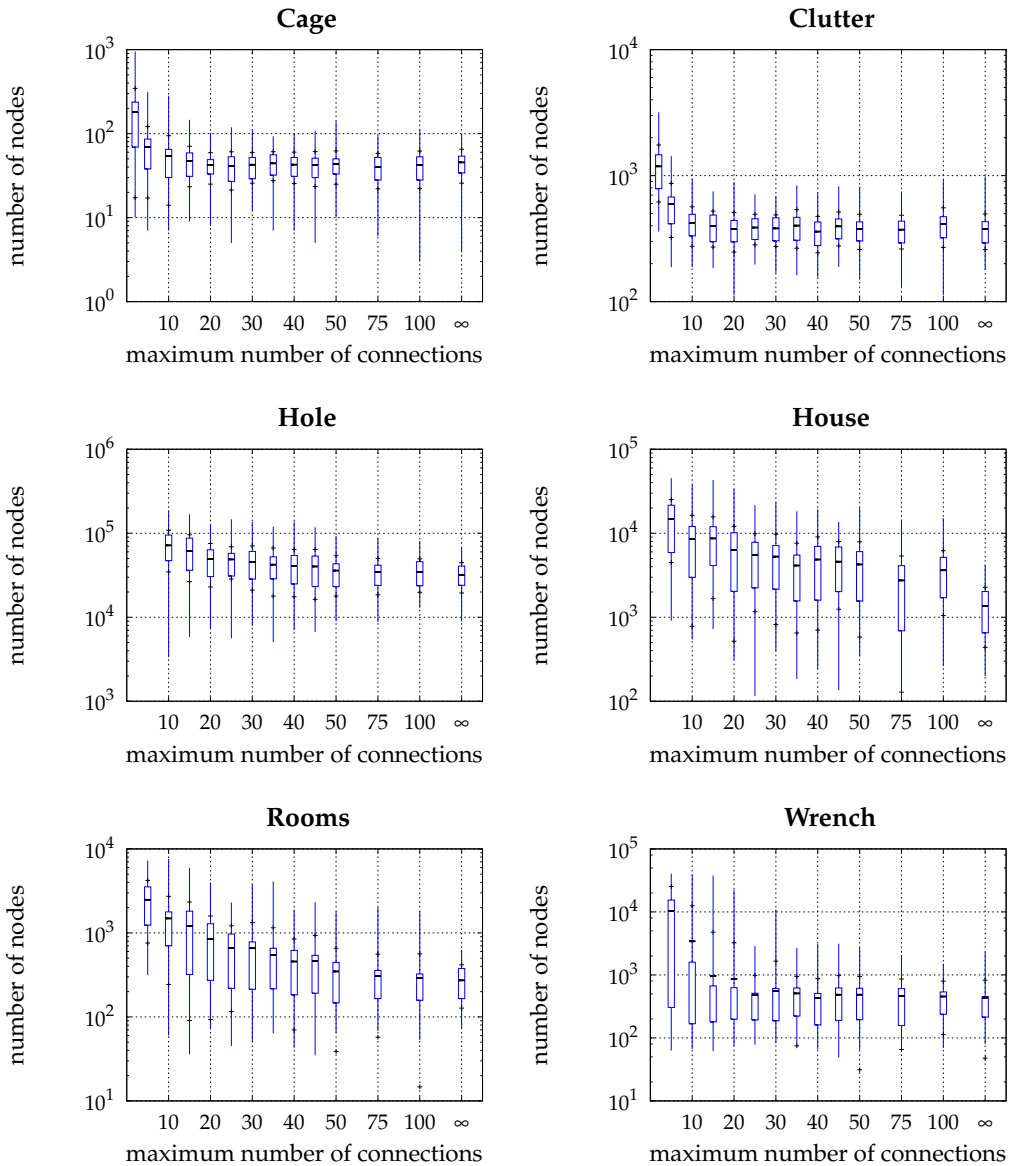


Figure 2.6 Relation between the *maximum number of connections* and the number of nodes. A series of box plots is shown for each environment. Note that we use a logarithmic scale for the 'number of nodes'-axis.

short connections that can be efficiently checked for collisions.

component We try to connect the new configuration to the nearest node in each connected component that lies close enough. The rationale is that we prefer to connect to multiple connected components.

component-*k* We try to connect the new configuration to at most k nodes in each connected component. Still, we keep the total number of connections tried small (the same number as for nearest-k). The rationale is that when the number of components is small we prefer to spend some extra time on trying to make connections. Otherwise the time required for adding the node will become the dominant factor. Preliminary experiments showed that setting k to 3 works fine in our environments.

visibility This method is based on the visibility sampling technique described in [121]. This technique only connects configurations to useful nodes. Usefulness is determined as follows: when a new node cannot be connected to other nodes it forms a new connected component and is labeled useful. If it connects two or more components it is also labeled useful. If it can be connected to just one component it is not labeled useful. It has been observed in [121] that the number of useful nodes remains small, making it possible to try connections to all of them. Hence, we do not restrain the connection distance and number of connections.

visibility-*k* This method is again based on the visibility sampling technique, but now only the nearest k nodes that lie close enough will be considered for the usefulness test.

Table 2.6 and Figure 2.7 summarize the results. Although the visibility approach pruned the graph a lot, it still performed worse on most environments. Only for the Hole environment it performed better. We feel that the reason is that the approach is too strict in rejecting nodes. However, the method performed better when restrictions were put on the two parameters. In general, the nearest-k technique performed relatively well, except in the Hole environment. We must remark that for this environment it is better to use an obstacle-based sampling technique such as *nearest contact* which will be discussed in Section 2.7. This technique is two orders of magnitude faster than the visibility approach.

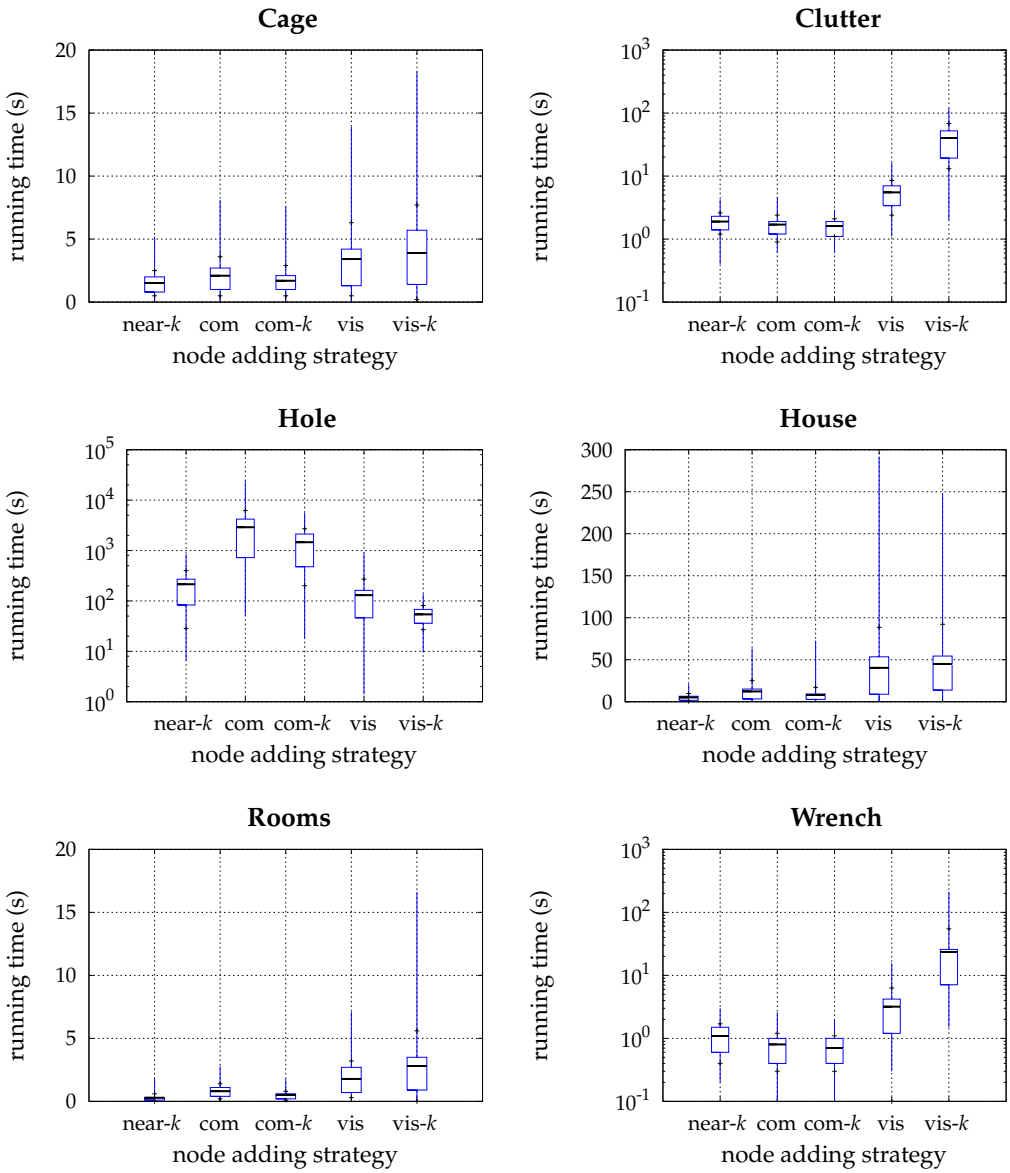


Figure 2.7 Relation between the *node adding strategy* and the running time. A series of box plots is shown for each environment. Note that we sometimes use a logarithmic scale on the 'running time'-axis.

	Node adding strategy				
	nearest-k	**comp**	**comp-k**	**visibility**	**visibility-k**
Cage	**1.5**	2.1	1.7	4.0	3.4
Clutter	1.9	1.7	**1.6**	40.0	5.5
Hole	213.6	2,886.5	1,451.9	129.4	**54.0**
House	**5.1**	12.2	7.8	44.5	40.5
Rooms	**0.3**	0.8	0.5	2.8	1.8
Wrench	1.1	0.8	**0.7**	23.5	3.2

Table 2.6 Average running times of five distinct node adding strategies.

2.6 Uniform sampling

The first papers on the PRM used uniform random sampling of the configuration space to select the nodes that are added to the graph. In recent years, other uniform sampling approaches have been suggested to remedy certain disadvantages of the random behavior. In particular, we study the following techniques (see Figure 2.8 for a visual impression of the corresponding sampling distributions):

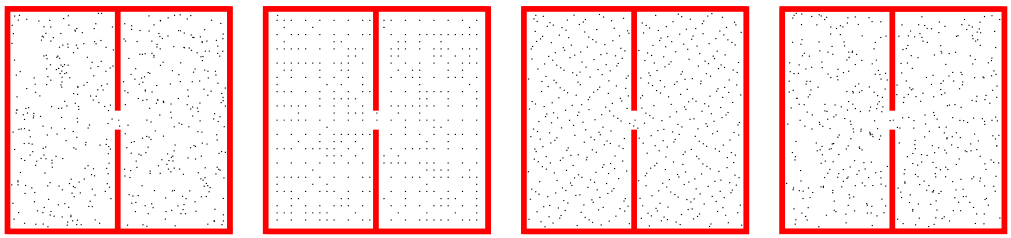
random In the random approach a sample is created by choosing random values for all degrees of freedom of the moving object.

grid In this approach we choose samples on a grid. Because the grid resolution is unknown in advance, we start with a coarse grid and refine this grid in the process, halving the cell size. Grid points on the same level of the hierarchy are added in random order.

Halton In [23] it has been suggested to use so-called Halton point sets as samples. Halton point sets have been used in discrepancy theory to obtain a coverage of a region that is better than using a grid (see e.g. [30]). It has been suggested in [23] that this deterministic method is well suited for the PRM.

Halton* In this variant of Halton we choose a random initial seed instead of setting the seed to 0 [154]. The claims in [23] should still hold because they are independent of the seed. By choosing a random seed we avoid the situation in which seed 0 is lucky or unlucky.

cell-based In this approach we take random configurations within cells of decreasing size in the workspace. The first sample is generated randomly



(a) random (b) grid (c) Halton (d) cell-based

Figure 2.8 Uniform sampling strategies in a 2D environment. Each image shows a typical distribution of 500 samples.

	Uniform sampling strategy				
	random	**grid**	**Halton**	**Halton***	**cell-based**
Cage	1.8	3.0	**1.3**	1.5	2.2
Clutter	1.7	3.7	**1.5**	1.9	1.6
Hole	237.9	**84.7**	101.5	213.6	232.4
House	20.6	14.5	**2.3**	5.1	21.2
Rooms	0.5	0.6	**0.2**	0.3	0.6
Wrench	1.5	1.4	**1.0**	1.1	1.4

Table 2.7 Average running times of five uniform sampling strategies.

in the whole space. Next we split the workspace in 2^3 equally sized cells. In a random order we generate a configuration in each cell. Next we split each cell into sub-cells and repeat this for each sub-cell. This should lead to a better distribution of the samples over the configuration space compared to random sampling. A similar approach was used in [129].

An important issue is how to choose random values. Random values were obtained by the Mersenne Twister [113]. Sampling for the rotational degrees of freedom was performed by choosing random unit quaternions [92], except for the Halton approach as this method is deterministic.[5] See [159] for a more extensive elaboration on sampling methods for these DOFs.

[5]We are aware that this choice might negatively influence the performance of this method. Contrary to the claim in [159], we experienced little difference in our experiments when we chose uniform random quaternions instead of Halton Euler angles which were converted to quaternions.

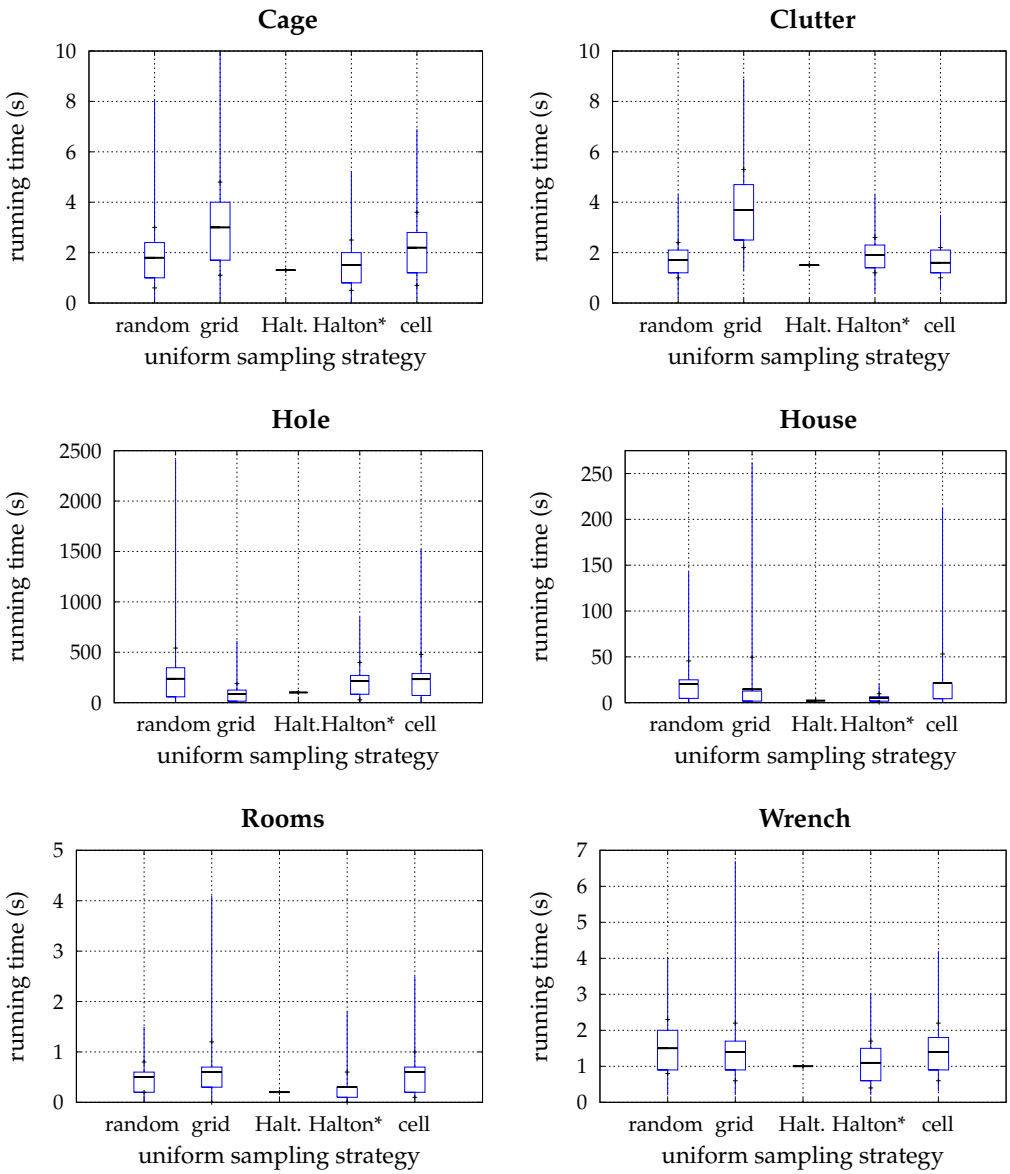


Figure 2.9 Relation between the *uniform sampling strategy* and the running time. A series of box plots is shown for each environment.

Table 2.7 and Figure 2.9 summarize the results. At first glance, it may appear that the deterministic Halton approach performed best. Especially the differences for the Hole and House environments were large. We tested whether this approach was lucky for these environments by translating the house and hole by a few units. Now the differences were negligible. Hence, the method was lucky for these environments. The running times being constant does not necessarily mean that this deterministic method is more consistent in performance. If the seed is randomized, which is the case for the Halton* method, its variance was comparable to e.g. the random approach.

The differences were in general small, i.e. the slowest method took about twice the time of the fastest method, except for the House environment. In general we must conclude that there is little to win when using different kinds of uniform sampling in terms of average running time. Nonetheless, there can be other arguments to use a particular technique. For example, the Halton method is deterministic. However, such a deterministic approach might be unlucky for a particular environment.

2.7 Non-uniform sampling

Rather than using uniform sampling, it has been suggested to add more samples in difficult regions of the environment. In this section we study a number of these techniques (see Figure 2.10 for their typical distributions):

Gaussian Gaussian sampling is intended to add more samples near obstacles. The idea is to take two random samples, where the distance σ between the samples is chosen according to a Gaussian distribution. Only if one of the samples lies in $\mathcal{C}_{\text{free}}$ and the other lies in $\mathcal{C}_{\text{forb}}$ we add the free sample. This leads to a favorable sample distribution [20]. We conducted preliminary experiments to find the optimal values for σ. We set σ to $\{2, 2, 1, 1, 2, 8\}$ for the six environments, respectively.

obstacle-based This technique, based on [3], has a similar goal. We pick a uniform random sample. If it lies in $\mathcal{C}_{\text{free}}$ we add it to the graph. Otherwise, we pick a random direction and move the sample in that direction with increasing steps until it becomes free and add the free sample. We set the initial step size to the corresponding step size from Table 2.3.

obstacle-based* This is a variation of the previous technique where we discard a sample if it initially lies in $\mathcal{C}_{\text{free}}$. This will avoid many samples in large open regions.

	Non-uniform sampling strategy						
	Gaussian	**obstacle**	**obstacle***	**bridge**	**MA**	**NC**	**Halton***
Cage	5.2	2.5	5.0	6.0	143.2	4.1	**1.5**
Clutter	2.8	2.9	4.2	6.3	411.8	4.5	**1.9**
Hole	4.5	27.2	3.0	38.4	111.5	**1.2**	213.6
House	7.1	6.0	5.1	9.9	433.4	**3.9**	5.1
Rooms	0.4	0.4	0.4	0.6	1.4	0.4	**0.3**
Wrench	2.3	1.9	3.0	7.7	17.9	4.0	**1.1**

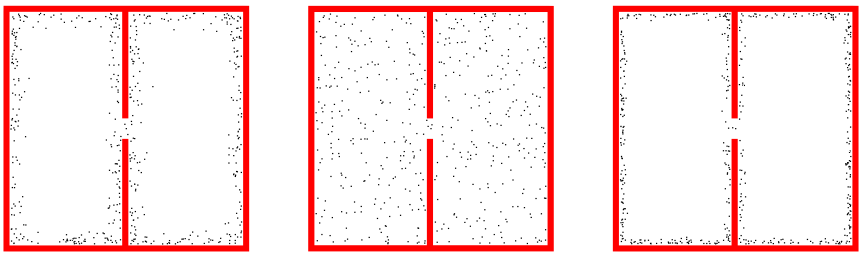
Table 2.8 Comparison of average running times of six non-uniform sampling strategies.

bridge test The bridge test is a hybrid technique that aims at better coverage of the free space [69]. The idea is to take two random samples, where the distance σ between the samples is chosen according to a Gaussian distribution. Only if *both* samples lie in $\mathcal{C}_{\text{forb}}$ and the point in the middle of them lies in $\mathcal{C}_{\text{free}}$ the free sample is added. To also get points in open space, every sixth sample is chosen random. We set σ to $\{5, 5, 2, 1, 4, 5\}$ for the six environments, respectively.

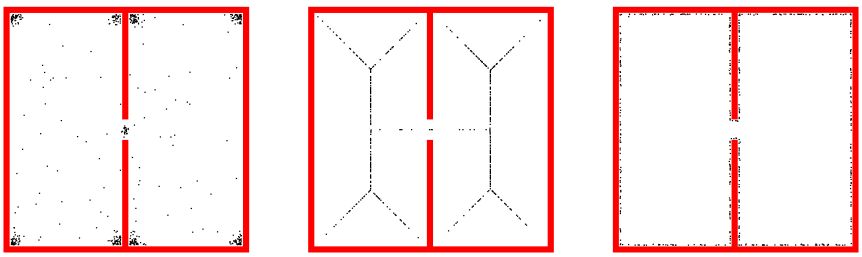
medial axis This technique generates samples near the medial axis (MA) of the free space [156]. (Algorithm 4.1 shows how to generate such samples). All samples have two equidistant nearest points resulting in a large clearance from obstacles. The method is relatively expensive to compute since expensive closest pair calculations are involved.

nearest contact This method is based on [107] and generates samples on the boundary of the $\mathcal{C}$-space and can be considered as the opposite of the medial axis technique. First, we choose a uniform random sample c. If c lies in $\mathcal{C}_{\text{free}}$ we discard it, else we calculate the penetration vector v between c and the environment. Then, we move c in the opposite direction of v and place c on the boundary of the $\mathcal{C}$-space. Care must be taken not to place c exactly on the boundary, because then it would be difficult to make connections between the samples. Hence, we move c away from the boundary with the step size listed in Table 2.3.

Due to their biased distributions, we expect these techniques to be useful only in environments where there are ample spaces (in the configuration space) and some narrow passages. Table 2.8 and Figure 2.11 show the results. (The results of the Halton* approach is stated for comparison.)



(a) Gaussian (b) obstacle-based (c) obstacle-based*



(d) bridge (e) medial axis (f) nearest contact

Figure 2.10 Non-uniform sampling strategies in a 2D environment. Each image shows a typical distribution of 500 samples.

As expected, the techniques only performed considerably better for the Hole environment. Also for the House environment, the methods works well but the improvement was not significant. However, in other situations, the methods were up to 10 times slower. The medial axis approach was even worse, due to the expensive calculations. This method does provide samples that are nicely located between the obstacles which results in motions with a higher clearance. (We will discuss clearance in Chapter 4 and 6.) We conclude that special non-uniform techniques should only be used in specific situations with narrow corridors. Preferably, they should only be used in the parts of the workspace where this is relevant, see e.g. [73].

2.8 Discussion

In this chapter we presented the results of a comparative study of various PRM techniques. The results confirm previous claims that the binary approach for collision checking works well. In contrast, the results also show that many claims on efficiency of certain sampling approaches could not be verified, i.e.

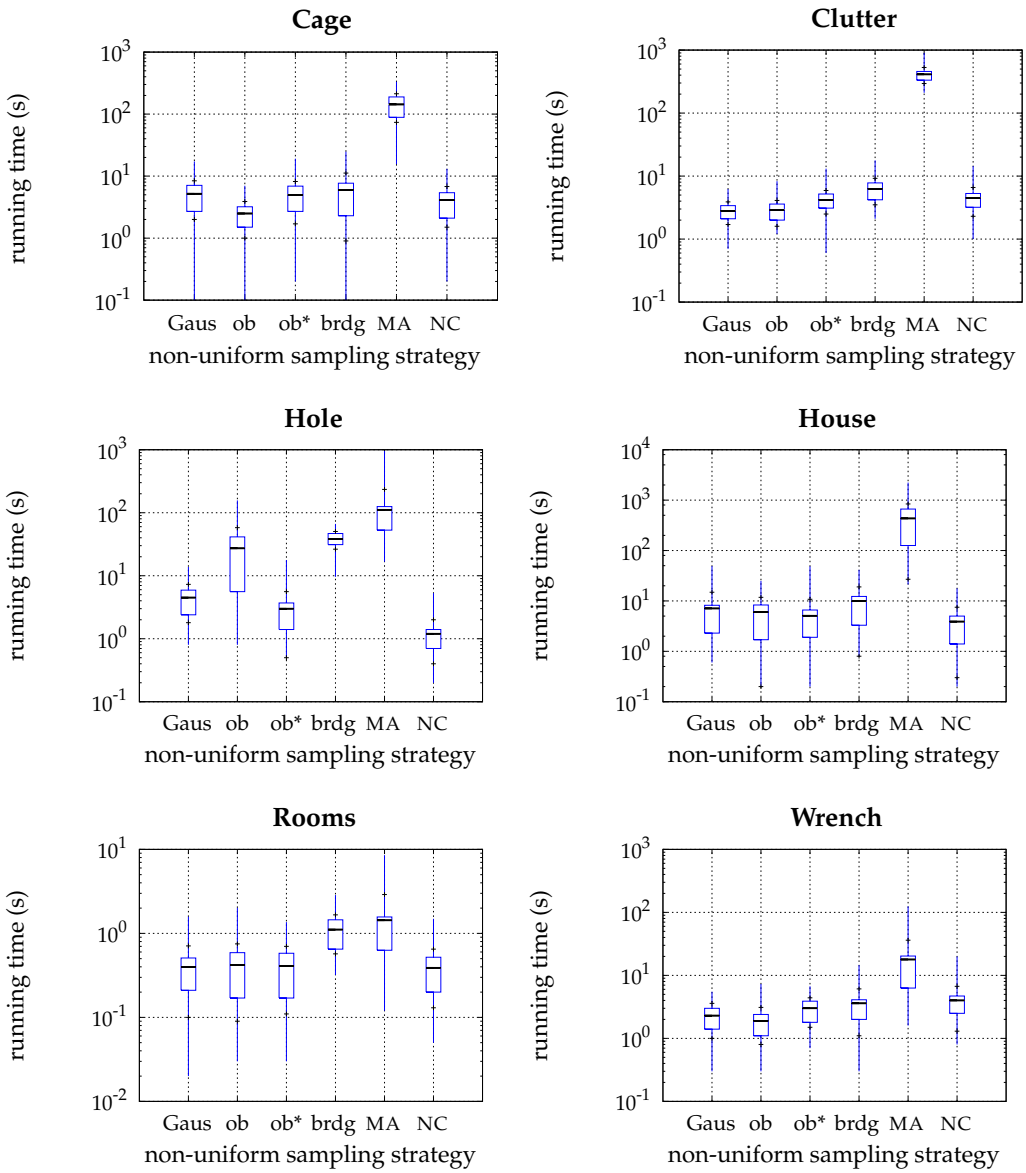


Figure 2.11 Relation between the *non-uniform sampling strategy* and the running time. A series of box plots is shown for each environment. Note that we use a logarithmic scale on the 'running time'-axis.

there was little difference between the various uniform sampling methods. However, the (deterministic) Halton approach performed best. These methods were only outperformed by non-uniform sampling methods for special (narrow passage) cases.

For node adding it turned out that visibility sampling did not perform as well as expected. A technique based on connecting a new configuration to the nearest-k configurations works relatively well. It turned out that the maximum number of connections attempted, i.e. the value of k, has to be set fairly high (like 75). The maximum connection distance is dependent on the environment and robot. This distance should not be set too low as this may increase the running time by orders of magnitude compared to the optimal running time.

One thing that is clear from this study is that a careful choice of techniques is important. Also, it is not necessarily true that a combination of good techniques and parameter choices results in optimal running times. For example, for the Hole environment one might expect that a combination of nearest contact sampling and visibility node adding works best. But experiments showed that this combination is actually about two times worse than the best combination.

The study also shows the difficulty of evaluating the performance of the techniques. In particular the variance in the running time and the influence of certain bad runs were surprisingly large. Further research, in particular into adaptive sampling techniques, will be required to improve this. In addition, further study would be interesting for other robot types, such as articulated and car-like robots, since we only compared techniques for free-flying objects.

We hope that our study shed some more light on the question of what technique to use in which situation. A major challenge is to create planners that automatically choose an appropriate combination of techniques based on scene properties or that learn the optimal settings while running.

In this chapter we compared and analyzed techniques based on solving a particular query. In contrast, the next chapter evaluates the techniques based on solving every possible query. To solve each query, the following two criteria have to be satisfied. First, the free configuration space ($\mathcal{C}_{\text{free}}$) must be covered by the nodes in the graph. Second, for each two nodes in the graph: if these nodes share the same connected component in $\mathcal{C}_{\text{free}}$, then there must exist a path between them in the graph. This distinction will provide an even better understanding of the techniques.