Gaussian Sampling for Probabilistic Roadmap Planners

Valérie Boor, Mark H. Overmars, A. Frank van der Stappen
Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands
Email: {*valerie,markov,frankst*}*@cs.uu.nl*

Abstract

Probabilistic roadmap planners (PRMs) have become a popular technique for motion planning that has shown great potential. A critical aspect of a PRM is the probabilistic strategy used to sample the free configuration space. In this paper we present a new, simple sampling strategy, which we call the Gaussian sampler, that gives a much better coverage of the difficult parts of the free configuration space compared with the standard uniform sampler. This results in much smaller roadmaps that can be computed faster. The approach uses only elementary operations which makes it suitable for many different planning problems. Experiments indicate that the technique is indeed very efficient in practice.

Corresponding author:

Mark H. Overmars
Institute of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
the Netherlands
tel: +31-30-2533736
fax: +31-30-2513791
Email: *markov@cs.uu.nl*

Regular Paper

Keywords

motion planning, probabilistic roadmap planners, Gaussian sampling

Gaussian Sampling for Probabilistic Roadmap Planners*

Valérie Boor, Mark H. Overmars, A. Frank van der Stappen
Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands
Email: {*valerie,markov,frankst*}*@cs.uu.nl*

Abstract

Probabilistic roadmap planners (PRMs) have become a popular technique for motion planning that has shown great potential. A critical aspect of a PRM is the probabilistic strategy used to sample the free configuration space. In this paper we present a new, simple sampling strategy, which we call the Gaussian sampler, that gives a much better coverage of the difficult parts of the free configuration space compared with the standard uniform sampler. This results in much smaller roadmaps that can be computed faster. The approach uses only elementary operations which makes it suitable for many different planning problems. Experiments indicate that the technique is indeed very efficient in practice.

1 Introduction

Automated motion planning is rapidly gaining importance in various fields. Beside the obvious use in robotics, new applications arise in fields such as animation, computer games, virtual medicine, and maintenance planning and training in industrial CAD systems. See Figure 1 for a typical example of such an industrial scene with over 4000 obstacles, containing a total of over 350000 triangles. Such applications put different demands on the motion planners. In particular, the scenes they have to work in are very complex and obstacles tend to be clustered in certain areas. As a result motion planning is easy at many places in the scenes but very difficult in such cluttered areas. This asks for planners that adapt their strategy to the local properties of the scene and for which the complexity of finding a path depends more on the difficulty of the path than on the complexity of the total scene.

Over the past two decades the motion planning problem has been studied extensively. Many different approaches have been proposed, including potential field techniques, roadmap methods, cell decomposition, neural networks, and genetic algorithms. (See the book of Latombe [15] for an extensive overview of the situation up to 1991 and e.g. the proceedings of the yearly IEEE International Conference on Robotics and Automation for many more recent results.) Unfortunately, many of these approaches are not very well suited for the applications described above. The complexity of the methods often depends on the complexity of the total scene (like the cell-decomposition methods) or the planners cannot deal adequately with cluttered parts of the environments (like potential field approaches). The *probabilistic*

*This research has been supported by the ESPRIT LTR project MOLOG. A preliminary version of this paper appeared in [6].

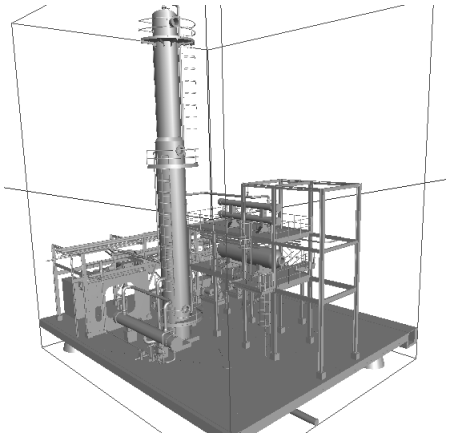

Figure 1: A typical industrial scene with over 4000 obstacles.

roadmap planner (PRM), also called the probabilistic path planner (PPP), is a relatively new approach to motion planning, developed independently at different sites [3, 9, 11, 17, 18]. A big advantage of PRM is that its complexity is indeed dependent on the difficulty of the path, and much less on the global complexity of the scene. Unfortunately, it does have problems with scenes with varying obstacle density. In this paper we will present a general approach to remedy this problem.

The motion planning problem is normally formulated in terms of the *configuration space* $\mathcal{C}$, the space of all possible configurations of the robot. Each degree of freedom of the robot corresponds to a dimension of the configuration space. Each obstacle in the workspace, in which the robot 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 of the robot 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 with $\mathcal{C}_{\text{free}}$.

Globally speaking, the probabilistic roadmap planner samples the configuration space for free configurations and tries to connect these configurations into a roadmap of feasible motions using a simple local planner. See Section 2 for a more detailed description of the technique. Over the past few years the method has been successfully applied in various motion planning areas, such as articulated robots [12], mobile robots with non-holonomic constraints [19, 21], multiple robots [20], and flexible objects [7, 14]. The method is very efficient but, due to the probabilistic nature, it is difficult to analyse (see e.g. [10]).

Already in the earliest papers on probabilistic roadmap planners it was noticed that the random adding of free configurations is a bottleneck when small cluttered regions of $\mathcal{C}_{\text{free}}$ play an essential role. Various techniques were proposed to overcome this, for example by adding extra configurations at promising places (see e.g. [17]), or by extending the roadmap at difficult places [12]. Hsu *et al.* [8] describe a technique based on a dilation of the free configuration space, that is, they allow configurations in which the robot slightly penetrates the obstacles. In later stages free configurations are created in the neighborhood of these penetrating configurations. Amato *et al.* [1, 2] describe a number of techniques that try to add configurations near favourable points on obstacles (e.g. along edges or near vertices).

Wilmarth *et al.* [22] propose a technique that samples near the medial axis of $\mathcal{C}_{\text{free}}$. These approaches have been shown to work well in particular environments. Unfortunately, they require rather complicated geometric operations and they work on individual obstacles, making them sensitive to the particular subdivision of a scene into individual obstacles. Recently some new global techniques have been described. Nissoux *et al.* [16] describe a technique called *visibility based PRM* that tries to keep the roadmap small by only adding "useful" nodes. Bohlin and Kavraki [5] describe a lazy-evaluation technique that only checks those parts of the roadmap that are relevant for a query. Both of these techniques are complimentary to our approach and can be used in combination with it.

Because the problems are caused by uniform sampling, the obvious idea is to try to sample the configuration space in a non-uniform way. The sample density should depend on the clutteredness of the surroundings. Cluttered areas should get many more samples than open areas. (Actually for two reasons: the chance that the samples lie in $\mathcal{C}_{\text{forb}}$ is larger, and motion planning in these areas is more difficult.) One could try to compute such areas in the workspace but this requires complicated geometric operations. In this paper we describe a new, general sampling approach that obtains such a distribution without any precomputation. The approach is especially useful for motion planning problems in which a relatively small robot moves in a large environment with non-uniform obstacle distribution, such as industrial CAD models. We call the approach the *Gaussian sampler*. It is based on the notion of Gaussian blurring, used in image processing. The technique adds configurations in difficult regions without doing any computations in the configuration space. It only uses simple intersection tests in the workspace. This makes the approach suitable for many different motion planning problems. Experiments show that Gaussian sampling considerably reduces the number of samples required, in this way improving the efficiency of PRM.

The paper is organised as follows. In Section 2 we briefly recall the PRM method and make some general remarks about its efficiency. In Section 3 we formally describe the favorable sample distribution we would like to obtain, basing ourselves on Gaussian blurring techniques from image processing. Rather than computing such distributions, we show in Section 4 that such a distribution can be obtaining use a very simple sampling strategy. Experiments, reported in Section 5, show that the method indeed leads to significant improvements. Finally, we draw some conclusions in Section 6.

2 Probabilistic roadmap planners

Let us start with a brief introduction on probabilistic roadmap planners. There are a number of versions of PRM, but they all use the same underlying concepts. Here we base ourselves on the description in [18].

The global idea of PRM is to pick a collection of (random) configurations in the free space $\mathcal{C}_{\text{free}}$. These free configurations form the nodes of a graph $G = (V, E)$. A number of 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 very fast, but is allowed to fail on difficult instances. A typical choice is to use a simple interpolation between the two configurations, and then check whether the path is collision-free. See Figure 2 for an example of a graph created with PRM in a simple 2-dimensional scene. (Because the configuration space is 3-dimensional, the graph should actually be drawn in this 3-dimensional space. In the figure and in all other figures in this

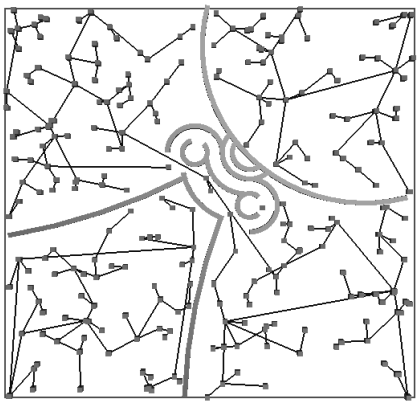

Figure 2: A typical graph produced by PRM.

paper we project the graph back into the workspace.)

Once the graph reflects the connectivity of $\mathcal{C}_{\text{free}}$ it can be used to answer motion planning queries. To find a motion between a start configuration and a goal configuration, both are added to the graph using the local planner. Then a path in the graph is found which corresponds to a motion for the robot. In a post-processing step this path is then smoothed to improve its quality. The pseudo code for the algorithm for constructing the graph is shown in Algorithm ConstructRoadmap.

Algorithm 1 ConstructRoadmap

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

1: **loop**
2: $c \leftarrow$ a (random) configuration in $\mathcal{C}_{\text{free}}$
3: $V \leftarrow V \cup \{c\}$
4: $N_c \leftarrow$ a set of 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

There are many details to fill in in this global scheme: which local planner to use, how to select promising pairs of nodes to connect, what distance measure to use, how to improve the resulting paths, etc. See the various references for more information.

If we already know the start and goal configuration, we can first add them to the graph and continue the loop until a path between start and goal exists. Note that the test in line 6 guarantees that we never connect two nodes that are already connected in the graph. Although such connections are indeed not necessary to solve the problem, they can still be useful for creating shorter paths.

The two time-consuming steps in this algorithm are line 2 where a free sample is generated, and line 7 where we test whether the local method can find a path between the new sample and a configuration in the graph. The geometric operations required for these steps dominate the work. Let T_s denote the time required to create a free sample (line 2) and let T_a denote

the time required to add it to the graph (lines 3–8). Then the total time for the algorithm is roughly

$$T = n(T_s + T_a) \tag{1}$$

where n denotes the total number of samples required to solve the problem. In the standard implementation of PRM T_s is *much* smaller than T_a (typically two or three orders of magnitude, depending on the local planner and the choice of N_c). This suggests that we could improve the running time of the algorithm considerably by sampling more carefully, that is, by increasing T_s to reduce n. Different sampling schemes studied recently follow this approach (e.g. [16, 22]).

Many sampling techniques globally work as follows: they compute certain configurations, test whether they are useful (that is, whether they lie in the free space and satisfy some additional properties) and, if so, add them to the graph. In this case we can rewrite the above formula as

$$T = n_t T_t + n_a T_a \tag{2}$$

Where n_t is the number of tried samples, n_a is the number of successful samples that we add to the graph, T_t is the time required to test a sample, and T_a is, as above, the time required to add a sample to the graph. Again T_t is normally much smaller than T_a. Testing a sample typically involves checking whether the robot in a particular configuration intersects an obstacle. Adding the sample to the graph, however, involves the computation of various paths, using the local planner and testing these paths for collisions with the obstacles. This suggests that it is advantageous to try many more samples (that is increase n_t), but be much more selective in the ones to keep (that is, decrease n_a). The Gaussian sampler described below does exactly this. It throws away almost all samples in open areas while keeping those in cluttered areas.

3 A favorable sample distribution

What would be a good collection of samples for the probabilistic roadmap planner? As stated before, we do not need many samples in large open regions in the configuration space. We primarily need samples that lie in difficult regions, close to obstacles, so that the local planner has more chance of success in finding a path there. Therefore, we want the probability that a sample is added to the graph to depend on the amount of forbidden configurations nearby.

Because we assume that the robot is small with respect to the workspace, the translational degrees of freedom of the robot are crucial in determining whether the robot lies in a cluttered or an open area. Hence, we only need to adapt the sample distribution for these. The remaining (rotational and other) degrees of freedom can be choosen uniformly. Assume we picked values for these remaining degrees of freedom. Let $\mathcal{C}^t$ be the resulting cross-section of the configuration space for the translational degrees of freedom. We need to define a sample distribution on $\mathcal{C}^t$.

Borrowing from image processing terminology, we can formally describe the desired sample distribution as follows: We define a Gaussian probability distribution on $\mathcal{C}^t$ (with dimension d) as:

$$\phi(p, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}^d} e^{-\frac{p^2}{2\sigma^2}} \tag{3}$$

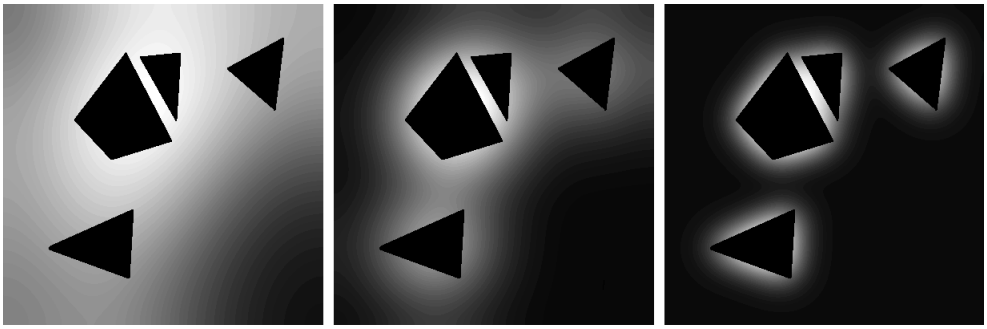

Figure 3: The sample distribution $g(p, \sigma)$ for a, from left to right, decreasing scale σ (white = 1, black = 0).

σ is the scale (or width) of the Gaussian. Let $\mathcal{C}^t_{\text{free}}$ be the free part of $\mathcal{C}^t$ and let $\mathcal{C}^t_{\text{forb}}$ be the forbidden part. Let

$$f(p, \sigma) = \int_{p' \in \mathcal{C}^t_{\text{forb}}} \phi(p - p', \sigma) \mathrm{d}p' \tag{4}$$

$f(p, \sigma)$ blurs the obstacles (in $\mathcal{C}^t$) with the Gaussian. To avoid forbidden configurations we define

$$g(p, \sigma) = \begin{cases} f(p, \sigma) & \text{if } p \in \mathcal{C}^t_{\text{free}} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$g(p, \sigma)$ is the probability distribution we want. Note that $g(p, \sigma)$ depends on the value of the other degrees of freedom, because $\mathcal{C}^t_{\text{free}}$ is different for different values.

The scale σ is an important parameter here. It indicates how close we like the configurations to stay to the obstacles. In Figure 3 an example is given of this sample distribution for a very simple motion planning problem for a point robot in the plane, for three different scales. Here $\mathcal{C}^t$ is equal to the workspace because there are no additional degrees of freedom. The lighter the color, the higher the probability a configuration will be sampled.

One could compute an approximation to $g(p, \sigma)$ using standard algorithms from image processing. Unfortunately $\mathcal{C}^t$ is normally 3-dimensional and, because the workspace is large compared to the size of the robot, a high percision would be required. Also the distribution depends on the value of the other degrees of freedom. In the next section we will show that there is a much easier way to obtain samples according to this distribution.

4 Obtaining the sample distribution

We will now describe a simple algorithm to obtain a set of configurations, distributed according to the distribution $g(p, \sigma)$ described above. For efficiency reasons, and for generality of the approach, we want to avoid computations in the configuration space. Following the philosophy of PRM, we restrict ourselves to using only one operation: determine whether the robot in a particular configuration intersects an obstacle, or whether the configuration is free. Such an operation can be implemented very efficiently.

A configuration c for the robot is a pair (c_t, c_r) where c_t consists of the translational (positional) degrees of freedom and c_r consists of the remaining degrees of freedom. As indicated in the previous section we want to choose c_t according to the Gaussian distribution while c_r can be chosen uniformly. The algorithm to achieve this is described in Algorithm GAUSSIANSAMPLING.

Algorithm 2 GAUSSIANSAMPLING

```
1: loop
2:    c1 ← a random forbidden configuration (ct, cr)
3:    d ← a distance chosen according to a normal distribution
4:    c2 ← a configuration (c't, cr) with c't at distance d from ct
5:    if c2 ∈ Cfree then
6:       add c2 to the graph
```

The idea of the algorithm is that we only add a free configuration to the graph (in the way described in Section 2) if we found a forbidden configuration close to it (according to the normal distribution). We will now show that this algorithm gives the distribution required.

Theorem 4.1 *The Gaussian sampling method generates nodes distributed according to distribution* $g(p, \sigma)$.

Proof: When taking a sample we pick the parameters c_r random, as required. So we can restrict ourselves to the translational parameters c_t. Using a normal distribution for the distance d with standard deviation σ^2, c'_t is chosen according to the distribution $\phi(c'_t - c_t, \sigma)$ defined in Equation 3. Because c_t is choosen randomly in $\mathcal{C}^t_{\text{forb}}$it follows that we pick samples c'_t according to the distribution $f(p, \sigma)$ defined in Equation 4. By testing whether $c_2 = (c'_t, c_r)$ lies in $\mathcal{C}_{\text{free}}$ the final distribution for the translational parameters of the nodes added to the graph is indeed $g(p, \sigma)$ as defined in Equation 5. □

The Gaussian sampler described above is slightly wasteful in the sense that if c_1 is not forbidden we discard it. Algorithm GAUSSIANSAMPLING2 avoids this. It is easy to show that it is equivalent to the first algorithm in the sense that it generates the same distribution of nodes.

Algorithm 3 GAUSSIANSAMPLING2

```
1: loop
2:    c1 ← a random configuration (ct, cr)
3:    d ← a distance chosen according to a normal distribution
4:    c2 ← a configuration (c't, cr) with c't at distance d from ct
5:    if c1 ∈ Cfree and c2 ∉ Cfree then
6:       add c1 to the graph
7:    else if c2 ∈ Cfree and c1 ∉ Cfree then
8:       add c2 to the graph
9:    else
10:      discard both
```

Clearly, we still pay a price in efficiency. Let us compare the Gaussian sampler with the standard uniform sampler where we simply take a random configuration and add it to the

graph when it lies in the free space. The Gaussian sampler is slower for the following reasons: it tests two configurations to add one to the graph, and it discards them if both are free. This last point might seem rather counterintuitive at first, but it is the crux of the algorithm: avoid adding configurations in large empty regions. In Section 5 we will give experimental results, showing that the Gaussian sampler is indeed more efficient in scenes with varying obstacle density.

5 Experimental results

To test the effect of using Gaussian sampling rather than uniform sampling, we implemented the approach within the SAMPLE motion planning environment.[1] All tests deal with free-flying robots in a 3-dimensional workspace. Given a particular start and goal configuration we test how much time it takes (on average) before they are connected in the roadmap. All experiments were carried out on a computer with a Pentium III processor running on 500 MHz and a sufficient amount of memory.

As testcases we consider three problems that all have varying obstacle density, resulting in large open areas and small passages. These are the types of scenes where we expect the Gaussian sampler to be effective. If on the contrary the obstacles are reasonably uniformly distributed with wide corridors, uniform sampling will be a bit more efficient, because of the overhead involved in the Gaussian sampler.

5.1 Planner choices

There are a number of choices to be made when using a probabilistic roadmap planner, in particular the choice of the local planner and the set of neighbor nodes used for trying connections.

As local planner we use the simple straight line interpolation in the configuration space. Collisions are tested by taking small steps along the path and testing for intersection with the obstacles at each step. The stepsize of the local planner is kept constant throughout all testruns. (Although not the most fancy local planner, it is well suited for comparing the different sampling approaches.)

For neighbor selection we use the neighbor approach as described in [18]. Of all nodes that lie at a distance from the new configuration of at most half the size of the largest dimension of the workspace, we consider only the closest per connected component of the graph. As distance measure we use the Euclidian distance in configuration space, with the rotational dimension weighted appropriately according to the robot geometry.

5.2 Choosing the Parameter

As noted before there is one parameter that seems to plays an important role: the standard deviation σ^2 of the normal distribution, which corresponds to the scale σ of the Gaussian. If we choose a small standard deviation most configurations will lie very close to obstacles and it will be difficult to find them. If, on the other hand, we choose a very large standard deviation the configurations are almost uniformly distributed over the free space.

[1] System for Advanced Motion PLanning Experiments (SAMPLE), developed at Utrecht University by Valérie Boor, Arno Kamphuis, and Petr Švestka.

In our applications, where the robot is an object that translates and rotates, it turns out to be best to choose the standard deviation such that most configurations lie at a distance of at most the diameter of the robot from the obstacles. This keeps most configurations close to the obstacles, but often allows for some rotation of the robot around its reference point. It is easy to automatically pick a standard deviation σ_{default} that achieves this. The advantage of this is that no user-interaction is required.

We choose σ_{default} to be

$$\sigma_{\text{default}} \;=\; 2 * max(\|\overrightarrow{v_1}\|, \ldots, \|\overrightarrow{v_n}\|) \tag{6}$$

where $\overrightarrow{v_i}$ denotes the vector from the reference point of the robot to its i-th vertex. Since we want the value of σ_{default} to reflect the diameter of the robot, preferably the reference point should be located inside the convex hull of the robot. Logical choices are the centroid or the center of mass.

We did some experiments with different values of σ. It turns out that the above choice is a good one and that the efficiency of the approach is not highly dependent on the value of σ, that is, small changes in σ have little effect on the running time. But large changes definitely have a degrading effect on the performance.

5.3 A simple example

Our first test scene is depicted in Figure 4, together with a possible path. Although the scene is 3-dimensional the problem is basically 2-dimensional. The robot has to move from the large open area at the top, through a narrow winding corridor, to the large open area at the bottom. The robot is a small sphere that has two degrees of freedom (it always stays in contact with the ground). The scene is actually a lot harder than it might seem. The reason is that the corridor is very narrow and the robot just fits through. Hence, many samples are required to solve the problem.

The collections of samples created with uniform and Gaussian sampling are shown in Figure 4. As expected, the uniform sampler has to create a large number of nodes in the large open areas before enough nodes are created in the corridor to get a connection. As a result, many calls to the local planner are made that do not add any interesting new information. The following table summarizes the results (being the rounded average over a number of runs):

	Uniform	Gaussian
Total time:	920 s	39 s
Number of nodes:	18,000	460
Total number of samples:	22,000	86,000
Number of local planner calls:	74,000	1,800
Number of collision checks:	5,500,000	100,000

It clearly demonstrates the effect of Gaussian sampling: although the number of samples tried is larger, the number of calls to the local planner, and hence the number of collision checks, is a lot smaller, resulting in a huge improvement in running time.

Although this first example is difficult for the uniform sampler, it can be solved rather easily using for example geometric node adding [18] or the techniques from Amato et al.[1].

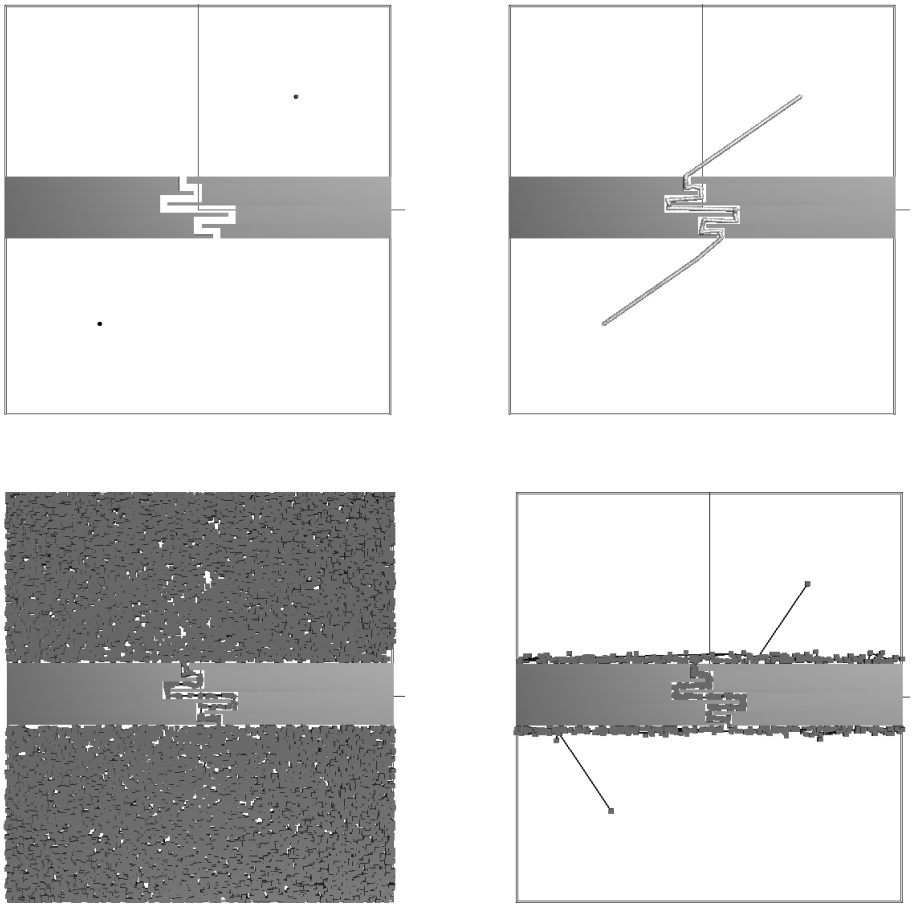

Figure 4: A simple scene in which the robot must move through the corridor from top to bottom. The top left picture shows the scene, the top right picture a possible path, the bottom left picture the nodes obtained with uniform sampling, and the bottom right picture the nodes obtained with Gaussian sampling.

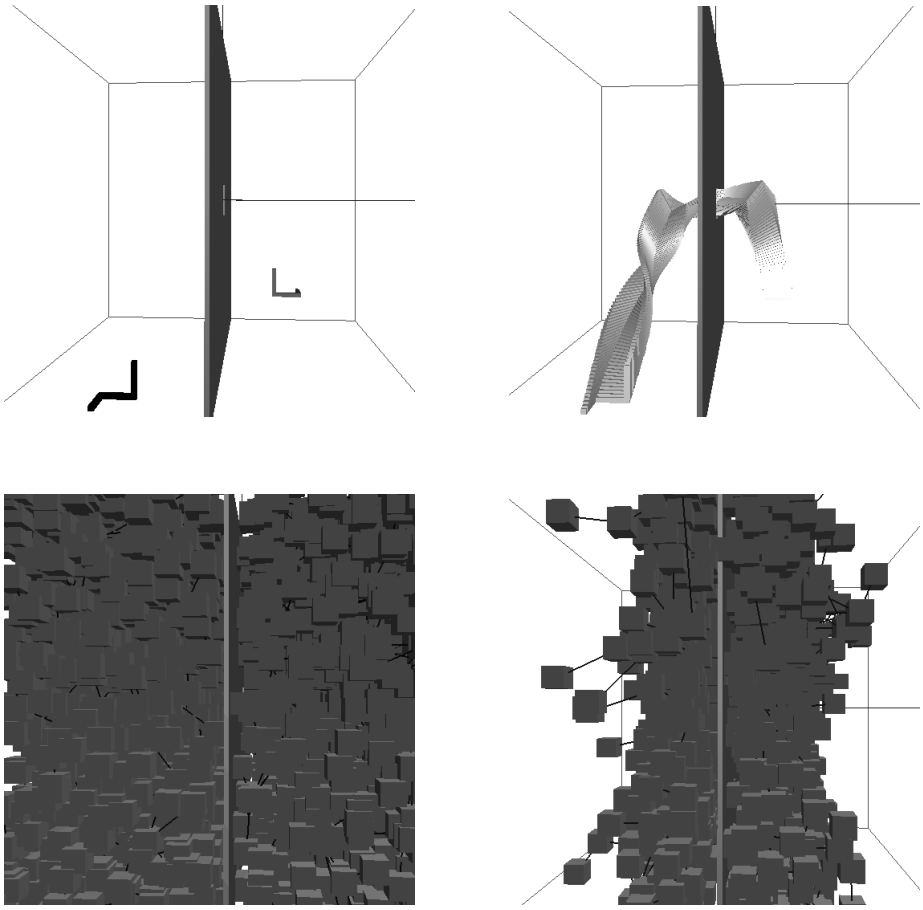

Figure 5: A more complicated motion planning problem in which the robot must move through the hole in the middle. The top left picture shows the scene, the top right picture a possible path, the bottom left picture the nodes obtained with uniform sampling, and the bottom right picture the nodes obtained with Gaussian sampling.

5.4 A small hole

Our second example, shown in Figure 5, is truly 3-dimensional. Here a robot, consisting of three perpendicular blocks has to move through a small hole in the middle of a big obstacle. The robot needs all its 6 (translational and rotational) degrees of freedom to get through the hole. Again we compare the uniform sampler with the Gaussian sampler. The resulting collections of nodes are shown in Figure 5.

The following table summarizes the results (being the rounded average over a number of runs).

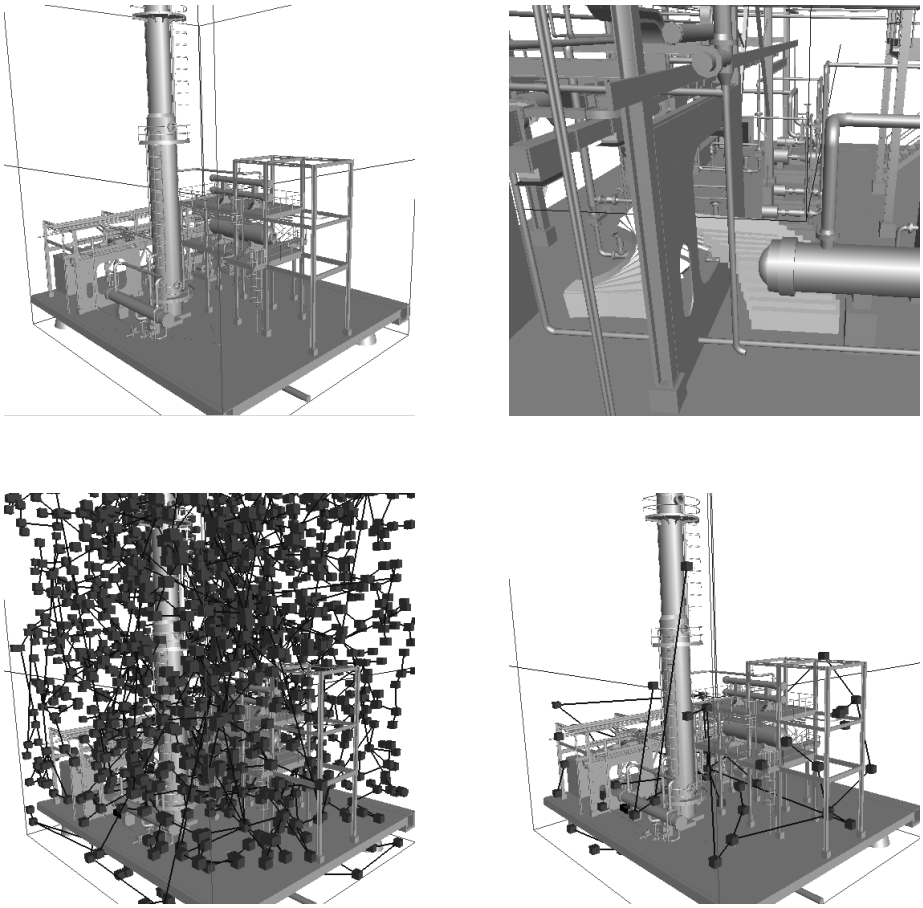

Figure 6: A more realistic motion planning problem in a complex industrial environment with over 4000 obstacles. The top left picture shows the scene, the top right picture a possible path, the bottom left picture the nodes obtained with uniform sampling, and the bottom right picture the nodes obtained with Gaussian sampling.

	Uniform	Gaussian
Total time:	120 s	30 s
Number of nodes:	7,000	2,500
Total number of samples:	7,500	50,000
Number of local planner calls:	12,500	5,500
Number of collision checks:	260,000	120,000

So again Gaussian sampling is quite a bit faster than uniform sampling. The effect though is smaller than in the previous scene because the robot is larger with respect to the size of the workspace.

5.5 An industrial setting

Both of the previous scenes contain only few obstacles. As a final example we consider a true industrial scene with over 4000 obstacles consisting of a total of over 350000 triangles. As can

be seen in Figure 6 the density of the obstacles is varying over the workspace. Even though the scene looks complicated, most motion planning problems in it are rather easy because there is a lot of space. Probabilistic roadmap planners are particularly suited for such scenes.

We try to move a block between two relatively difficult configurations. The following table summarizes the results (being the rounded average over a number of runs).

	Uniform	Gaussian
Total time:	48 s	34 s
Number of nodes:	970	210
Total number of samples:	1,100	4,000
Number of local planner calls:	3,800	2,700
Number of collision checks:	110,000	41,000

As can be seen the problem is solved reasonably fast for both the uniform and the Gaussian sampler. The difference in runtime is not as significant as in the previous scenes. The reason is that no area of the scene through which the motion of the block must run is really cluttered. Hence, the fact that the uniform sampler does place less samples at such locations does not effect its ability to solve the query too much.

6 Conclusions

In this paper we have presented a Gaussian sampler for probabilistic roadmap planners that results in a favorable distribution of samples over the configuration space. The technique is simple, general, and experiments demonstrate that it results in improvements over uniform sampling.

Even though the method as described already leads to improvements in efficiency, a number of issues remain that are interesting to study further.

A major question that remains is the correct choice for σ. Although experiments have shown that the choice is not too crucial, the efficiency of the approach is dependent on it. Our default choice σ_{default}, based on the diameter of the robot is only partially motivated. The best value would probably depend on the local properties of the scene. We can envision using a neural network approach (e.g. a Kohonen network) to learn the correct values, by looking at the success of earlier samples taken in particular areas in the scene.

In our approach we only use Gaussian sampling for the translational degrees of freedom for the robot. An interesting question is whether Gaussian sampling can under certain circumstances also be useful for other degrees of freedom and for other types of robots, like articulated robots and robots with non-holonomic constraints (e.g. car-like robots).

Finally, we would like to point out that Gaussian sampling should not be seen as the ultimate sampling method for PRMs. It merely indicates that using more clever sampling schemes important improvements can be achieved. To obtain the "best" PRM we probably need a combination of the various techniques developed over the past few years. Moreover, we need an automated system that adapts these techniques to the (local) properties of the motion planning problem to be solved. A major problem here is that most techniques have some parameters. Ideally, the user should not be asked for values of these parameters, and the system should automatically choose reasonable values and adapt the parameters using information obtained during the execution. Such integrated and adaptive planners form an important and chalenging topic of study for the coming time.

Acknowledgements

We would like to thank Arno Kamphuis who wrote a large portion of the SAMPLE motion planning environment that we used in the experiments reported in this paper.

References

[1] N. Amato, O. Bayazit, L. Dale, C. Jones, D. Vallejo, OBPRM: An obstacle-based PRM for 3D workspaces, in: P.K. Agarwal, L.E. Kavraki, M.T. Mason (eds.), *Robotics: The algorithmic perspective*, A.K. Peters, Natick, 1998, pp. 155–168

[2] N. Amato, O. Bayazit, L. Dale, C. Jones, D. Vallejo, Choosing good distance metrics and local planners for probabilisitc roadmap methods, *Proc. IEEE Int. Conf. Robotics and Automation,* 1998, pp. 630–637.

[3] N. Amato, Y. Wu, A randomized roadmap method for path and manipulation planning, *Proc. IEEE Int. Conf. Robotics and Automation,* 1996, pp. 113–120.

[4] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan, A random sampling scheme for path planning, *Int. Journal of Robotics Research* **16** (1997), pp. 759–774.

[5] R. Bohlin, L.E. Kavraki, Path planning using lazy PRM, *Proc. IEEE Int. Conf. Robotics and Automation,* 2000, pp. 521–528.

[6] V. Boor, M.H. Overmars, A.F. van der Stappen, The Gaussian sampling strategy for probabilistic roadmap planners, *Proc. IEEE Int. Conf. Robotics and Automation,* 1999, pp. 1018–1023.

[7] C. Holleman, L. Kavraki, J. Warren, Planning paths for a flexible surface patch, *Proc. IEEE Int. Conf. Robotics and Automation,* 1998, pp. 21–26.

[8] D. Hsu, L. Kavraki, J.C. Latombe, R. Motwani, S. Sorkin, On finding narrow passages with probabilistic roadmap planners, in: P.K. Agarwal, L.E. Kavraki, M.T. Mason (eds.), *Robotics: The algorithmic perspective*, A.K. Peters, Natick, 1998, pp. 141–154.

[9] L. Kavraki, *Random networks in configuration space for fast path planning,* PhD thesis, Stanford University, 1995.

[10] L. Kavraki, M. Kolountzakis, J.C. Latombe, Analysis of probabilistic roadmaps for path planning, *Proc. IEEE int. Conf. Robotics and Automation,* 1996, pp. 3020–3025.

[11] L. Kavraki, J.C. Latombe, Randomized preprocessing of configuration space for fast path planning, *Proc. IEEE int. Conf. Robotics and Automation,* 1994, pp. 2138–2145.

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

[13] J.J. Kuffner, S.M. LaValle, RRT-connect: An efficient approach to single-query path planning, *Proc. IEEE Int. Conf. Robotics and Automation,* 2000, pp. 995–1001.

[14] F. Lamiraux, L. Kavraki, Path planning for elastic plates under manipulation constraints, *Proc. IEEE int. Conf. Robotics and Automation,* 1999.

[15] J-C. Latombe, *Robot motion planning*, Kluwer Academic Publishers, Boston, 1991.

[16] C. Nissoux, T. Siméon, J.-P. Laumond, Visibility based probabilistic roadmaps, *Proc. IEEE int. Conf. Intelligent Robots and Systems,* 1999, pp. 1316–1321.

[17] M.H. Overmars, *A random approach to motion planning,* Technical Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, October 1992.

[18] P. Švestka, *Robot motion planning using probabilistic roadmaps,* PhD thesis, Utrecht Univ. 1997.

[19] P. Švestka, M.H. Overmars, Motion planning for car-like robots, a probabilistic learning approach, *Int. Journal of Robotics Research* **16** (1997), pp. 119–143.

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

[21] S. Sekhavat, P. Švestka, J.-P. Laumond, M.H. Overmars, Multilevel path planning for nonholonomic robots using semiholonomic subsystems, *Int. Journal of Robotics Research* **17** (1998), pp. 840–857.

[22] S.A. Wilmarth, N.M. Amato, P.F. Stiller, MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space, *Proc. IEEE Int. Conf. Robotics and Automation,* 1999, pp. 1024–1031.