The Gaussian Sampling Strategy for Probabilistic Roadmap Planners*

Valérie Boor, Mark H. Overmars, A. Frank van der Stappen

Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands
Email: {*valerie,markov,frankst*}*@cs.uu.nl*

Abstract

Probabilistic roadmap planners (PRMs) form a relatively new technique for motion planning that has shown great potential. A critical aspect of 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. The approach uses only elementary operations which makes it suitable for many different planning problems. Experiments indicate that the technique is very efficient indeed.

1 Introduction

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 [11] for an overview of the situation up to 1990, and the proceedings of ICRA and WAFR for many more recent results.

The motion planning problem is normally formulated in terms of the configuration space, 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. 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 any obstacle, that is, it lies completely in the free part of the configuration space, denoted with $\mathcal{C}_{\text{free}}$.

The probabilistic roadmap planner (PRM), also called the probabilistic path planner (PPP), is a relatively new approach to motion planning, developed independently at different sites [3, 7, 9, 12, 13]. Globally speaking, the approach samples the configuration space for free configurations and tries to connect these configurations into a roadmap of feasible motions using a simple local planner. Over the past few years the method has been successfully applied in various motion planning areas, involving amongst others articulated robots [10], mobile robots with non-holonomic constraints [14, 16], multiple robots [15], and flexible objects [5]. The method turns out to be very efficient but, due to the probabilistic nature, it is very hard to analyse (see e.g. [8]).

Already in the earliest papers on probabilistic roadmap planners it was noticed that the random adding of free configurations is a bottleneck when small difficult 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. [12]), or by extending the roadmap at difficult places [10]. Recently some more general approaches have been proposed. Hsu et al. [6] 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). 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 subdivision of a scene in individual obstacles.

In this paper we describe a new, general sampling approach that we call the Gaussian sampler. It is based on the notion of blurring, used in image processing. The technique adds configurations in difficult

*This research has been partially supported by the ESPRIT LTR project MOLOG

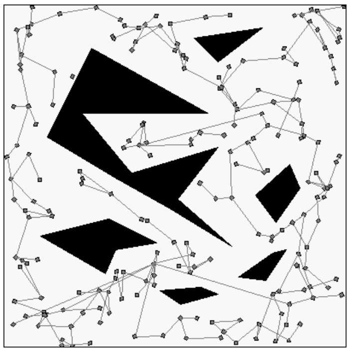

Figure 1: A typical graph produced by PRM.

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 reduces the number of samples required considerably, in this way improving the efficiency of PRM.

2 Probabilistic Roadmap Planners

Let us start with a very 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 [13].

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 very simple local motion planner is used to try to connect these configurations with 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 compute a short path in the absence of obstacles, and then check whether the path is collision-free. See Figure 1 for an example of a graph created with PRM.

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 is converted into a motion for the robot.

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.

In pseudo-code the algorithm for constructing the graph looks as follows:

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

The two time-consuming steps in this algorithm are line 3 where a free sample is generated, and line 8 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 3) and let T_a denote the time required to add it to the graph (lines 4–9). Then the total time for the algorithm is roughly

$$T = n(T_s + T_a)$$

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. 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. The different sampling schemes studied recently do exactly this.

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

$$T = n_t T_t + n_a T_a$$

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 a lot 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. 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).

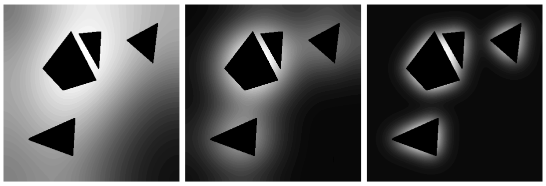

Figure 2: The sample distribution $g(c;\sigma)$ for a, from left to right, decreasing scale σ.

3 A Favourable Sample Distribution

What would be a good collection of samples for PRM? Clearly, we don't need many samples in large open regions in the configuration space. We do need samples that lie in difficult regions, close to obstacles (in the configuration space!). Therefore, we want the probability that a sample is added to the graph to depend on the amount of forbidden configurations nearby.

Borrowing from image processing terminology, we can formally describe this as follows: We define a Gaussian on the configuration space (with dimension d) as:

$$\phi(c;\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}^{d}} e^{-\frac{c^2}{2\sigma^2}}$$

σ is the scale (or width) of the Gaussian[1]. We define the function $Obs(c)$ to be 1 when in configuration c the robot intersects an obstacle and 0 otherwise. Let

$$f(c;\sigma) = \int Obs(y)\phi(c-y;\sigma)\mathrm{d}y$$

$f(c;\sigma)$ blurs the obstacles (in configuration space) with the Gaussian. To avoid forbidden configurations we define

$$g(c;\sigma) = \max(0, f(c;\sigma) - Obs(c))$$

that is, within the obstacles $g(c;\sigma) = 0$ and otherwise it is $g(c;\sigma) = f(c;\sigma)$. $g(c;\sigma)$ is the probability distribution we want.

The scale σ is an important parameter here. It indicates how close we like the configurations to stay to the obstacles. In Figure 2 an example is given of this sample distribution in a 2-dimensional configuration space for a very simple motion planning problem for a point robot, for three different scales. The lighter the color, the higher the probability a configuration will be sampled.

[1]Note that we might want to use a different scale for dimensions in the configuration space that correspond to translations and those that correspond to rotations. The formula can easily be adapted to this.

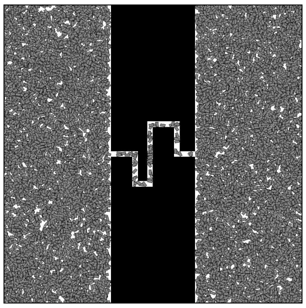

Figure 3: The collection of samples created by the random sampler (about 13000) before the corridor was adequately sampled.

4 Obtaining the Sample Distribution

We will now describe a simple algorithm to obtain a set of configurations, distributed according to the distribution 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 a configuration is free. Such an operation can be implemented very efficiently. Here is the algorithm:

loop
    $c_1 \leftarrow$ a random configuration
    $d \leftarrow$ a distance chosen according to a normal distribution
    $c_2 \leftarrow$ a random conf. at distance d from c_1
    if $c_1 \in \mathcal{C}_{\text{free}}$ **and** $c_2 \notin \mathcal{C}_{\text{free}}$ **then**
        add c_1 to the graph
    else if $c_2 \in \mathcal{C}_{\text{free}}$ **and** $c_1 \notin \mathcal{C}_{\text{free}}$ **then**
        add c_2 to the graph
    else
        discard both

So we only add a free configuration to the graph (in the way described in Section 2) if we found a forbidden configuration close by (according to the normal distribution). We call this sampling method the Gaussian sampler, because it produces a set of samples according to the distribution $g(c;\sigma)$.

Clearly, to obtain the favourable distribution, we pay a price in efficiency. Let us compare the Gaussian sampler with the standard random sampler where we simply take a random configuration and add it 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 7 we will give some experimental results, showing that the Gaussian sampler is indeed much more efficient in scenes with varying obstacle density.

5 Choosing the Parameter

As noted before there is one parameter that plays a crucial 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. If, on the other hand, we choose a very large standard deviation the configurations are almost uniformly distributed over the free space.

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 length of the robot from the obstacles. This keeps the configurations close to the obstacles, but, where possible, allows for rotation of the robot around its axis. It is easy to automatically pick a standard deviation that achieves this. The advantage of this is that no user-interaction is required.

Clearly, the rotational degree of freedom needs to be handled differently from the translational degrees of freedom. Our current approach is to pick only the position of the second sample according to the normal distribution, and to pick the orientation at random. This turns out to work well.

6 Experimental Setup

Here we briefly mention the choices we made regarding the implementation and setup of our experiments. Our experiments deal with a two-dimensional workspace, consisting of the unit square, in which the robot moves (translating and rotating) from a given start to a given goal configuration. These configurations are automatically added to the graph, so it can be checked whether the roadmap contains enough nodes, simply by checking whether start and goal are in the same connected component of the graph. We ran both the random sampler and the Gaussian sampler, for various sizes of sigma, until there were enough samples in the corridor to find the connection.

The default value of sigma is chosen to be equal to the distance of the reference point of the robot to its furthest vertex.

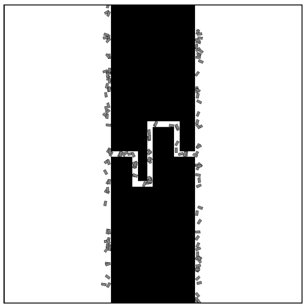

Figure 4: The collection of samples created by the Gaussian sampler (about 150).

In the experiments, the nodes chosen to try to connect to from a new configuration c, are the "neighbour nodes", as described in [13]. This means that of those within a certain distance ($d_{max} = \frac{1}{2}$) from c, we consider only the closest per graph component. As distance measure we use the Euclidian distance in configuration space, with the rotational dimension weighed appropriately according to the robot geometry.

The local planner is a simple potential-field-like planner, which has been shown to work well for free-flying robots, see [12]. Collisions are tested by taking small steps along the path and testing the slightly grown robot 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 good enough for comparing the different sampling approaches.)

As testcases we consider three difficult 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 distributed with wide corridors, random sampling will be more efficient[2].

7 Experimental Results

We will now describe some experimental results on the Gaussian sampler. All timing results come from a Pentium II PC running at 400 Mhz. We have tested various sizes of sigma. As expected, the running time of the algorithm increases when sigma is very small, making it hard to find a pair of nodes that has both a forbidden and a free configuration, so the ratio n_a/n_t decreases (deteriorating performance). Also when sigma

[2]For example, the scene in Figure 1, where the robot is a small rectangle, was solved by the random sampler in about half the time needed by the Gaussian sampler.

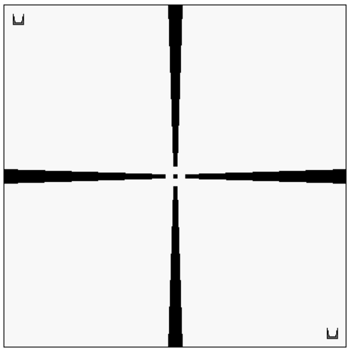

Figure 5: A scene requiring a difficult twist of the robot.

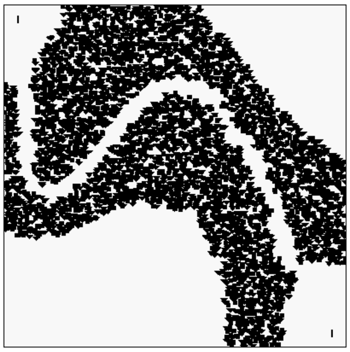

Figure 6: A scene with 5000 obstacles.

was very large, and the output of the sampler started to approximate random sampling, performance deteriorated. Here we only report results for the default value of σ, even though this was not necessarily the optimal value for the specific problem at hand. More extensive results can be found in the full version of this paper.

Our first testcase involves a scene with a small twisting corridor between two large open areas. The robot is a rectangle that has three degrees of freedom (translation and rotation). The ratio between robot width, robot length, and width of the corridor is 2:4:5, so it is difficult for the robot to get around the corners.

In Figure 3 you can see the collection of samples created by the random sampler. In Figure 4 you find the samples created by the Gaussian sampler. On average, the random sampler requires almost 70 times more nodes to obtain enough samples in the narrow corridor. The time T_t, needed for creating a sample, is about twice as high for the Gaussian sampler (that needs to test 2 samples) as for the random sampler. However, T_a is about equal *per node* for both sampling methods. The total running time using Gaussian sampling is almost 60 times shorter than with random sampling.

Although this first example is difficult for the random sampler, it can be solved rather easily using for example geometric node adding [13] or the techniques from Amato et al.[1]. Our second example, shown in Figure 5, is more difficult. Here a U-shaped robot has to twist to get through the narrow gap in the center. The robot just fits in. Again we compare the random sampler with the Gaussian sampler. To connect the start and goal configuration (as shown in Figure 5) the random sampler took about 13 times longer than the Gaussian sampler, and needed about 10000 nodes. The Gaussian sampler required about 750 nodes.

Both of these examples deal with small sets of obstacles. The method can easily be used for much more complicated scenes. Figure 6 shows a scene with 5000 (intersecting) obstacles. The obstacle density varies largely throughout the scene. The Gaussian sampler needed 85 nodes to connect start and goal, and did this about 4 times as fast as the Random sampler. The random sampler required over 450 nodes.

8 Extensions

The Gaussian sampler can be extended in many ways. Here we want to briefly mention one such extension.

When all obstacles are convex, difficult places lie close to at least two obstacles. However, the Gaussian sampler creates samples along all obstacles. For example, in Figure 4 the only difficult part is the corridor, but the Gaussian sampler also places samples along the long vertical walls. There is a simple way to avoid this: Rather than two, we pick three samples that lie close to reach other according to the normal distribution. If one lies in $\mathcal{C}_{\text{free}}$ and the other two intersect different obstacles, we keep the free sample. See Figure 7 for the effect. The number of nodes is reduced to about 100.

Even though this might look like an interesting improvement it is currenlty unclear whether this is indeed a useful extension. The number of samples reduces but the time to find them increases. In the example, this was not compensated by the reduction in time for adding the nodes to the graph. Hence, the total time did increase rather than decrease. (In 3-dimensional workspaces, where the time required for adding the nodes increases, the approach might be advantageous.) Also, the collection of samples becomes dependent on the subdivision of the scene in obstacles; something we wanted to avoid.

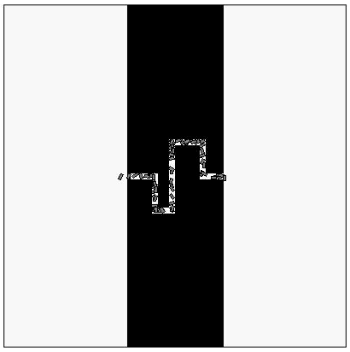

Figure 7: A smaller set of samples by using triples rather than pairs.

9 Conclusions

In this paper we have given some first results that we obtained with the Gaussian sampler. The technique is simple, general, and we demonstrated that it results in large improvements over random sampling. This indicates that it is a promising technique to study further.

A large number of issues remain. We like to get a better understanding of the effect of the standard deviation σ^2. Also, we want to apply the approach to more general robots. We are currently implementing the Gaussian sampler for a three-dimensional workspace in which the moving object can have up to six degrees of freedom. The approach carries over without major modifications. We expect that the results will be even more significant, because in a three-dimensional workspace the time required for adding a node to the graph (T_a) is huge compared to the time required for testing a sample (T_s). So it is even more important to add as few nodes as possible to the graph. It would also be interesting to see how the approach works for e.g. articulated robots.

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, MA, 1998.

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

[3] N. Amato, Y. Wu, A randomized roadmap method for path and manipulation planning, *Proc. IEEE Conf. Robotics and Automation,* Minneapolis, 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] C. Holleman, L. Kavraki, J. Warren, Planning paths for a flexible surface patch, *Proc. IEEE Conf. Robotics and Automation,* Leuven, 1998, pp. 21–26.

[6] 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, MA, 1998.

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

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

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

[10] 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.

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

[12] 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.

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

[14] 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.

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

[16] 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.