A Framework for Planning Feedback Motion Strategies Based on a Random Neighborhood Graph

Libo Yang Steven M. LaValle

Dept. of Computer Science
Iowa State University
Ames, IA 50011 USA
{lyang, lavalle}@cs.iastate.edu

Abstract

Randomized techniques have led to the development of many successful algorithms for path planning in high-dimensional configuration spaces. This paper presents a randomized framework for computing feedback motion strategies, by defining a global navigation function over a collection of spherical balls in the configuration space. If the goal is changed, an updated navigation function can be quickly computed, offering benefits similar to the fast multiple queries permitted by the probabilistic roadmap approach to path planning. Our choice of balls is motivated in part by recent tools from computational geometry which compute point locations and arrangements efficiently without significant dependence on dimension. We present a construction algorithm that includes a Bayesian termination condition based on the probability that a specified fraction of the free space is covered. A basic implementation illustrates the framework for rigid and articulated bodies with up to five-dimensional configuration spaces.

1 Introduction

Determining a collision-free motion strategy is one of the most basic operations in robotics. One of the greatest challenges in the design of many robotic systems is to perform global, geometric reasoning while also allowing quick on-line responses to unexpected events. In a traditional view of robotics, an off-line path planning algorithm considers global, geometric issues to determine a collision-free path for a given robot and set of obstacles. The solution is then passed to an on-line control algorithm that attempts to follow the path, while hoping that localization errors, control errors, dynamical constraints, and responses to unexpected obstacles do not cause failure. Recognizing this difficulty, many interesting alternatives have been proposed. For example, in the BUG paradigm [8, 9, 19, 18, 27], the robot makes decisions based on local information and assumes the environment is mostly unknown, yet global convergence is guaranteed. As another example, potential field approaches have been proposed to allow the robot to use sensor feedback to determine the actual course during navigation [3, 7, 11, 12, 26, 28, 20].

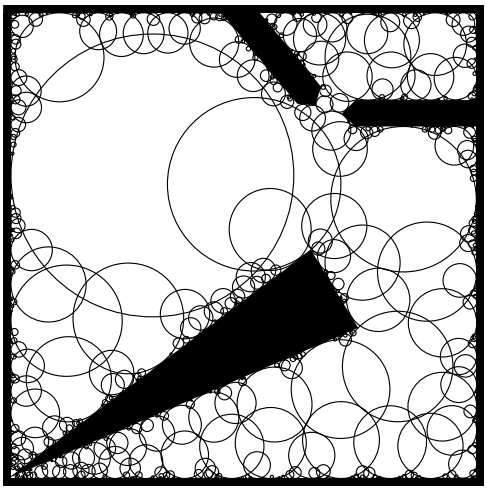

Figure 1: The configuration space is covered by balls on which navigation functions are defined.

In general, the task of constructing a collision-free feedback motion strategy is at least as difficult constructing a collision-free path. The computational expense required to solve general versions of either type of problem is prohibitive when there are many degrees of freedom. This difficulty has led to the development and success of randomized path planning algorithms, which are capable of solving many challenging and important practical planning problems [1, 2, 10, 25]. The tradeoff is that completeness is exchanged for a weaker, probabilistic convergence. Through clever use of

randomization, most of the high costs due to high dimensionality are eliminated. Randomized path planning methods have consequently generated great interest both in robotics, and beyond in areas such as virtual prototyping, computational chemistry, graphical animation, and architecture. Unfortunately, these applications contain problems that involve a static, off-line geometric reasoning task, which is often not a reasanoable solution for many kinds of robotics problems.

Inspired by both the success of randomized path planning techniques and sensor-feedback motion strategies, we propose a randomized framework for generating feedback motion strategies (see Figure 1) for robots with high degrees of freedom. We hope that this framework can facilitate the development of motion strategies for problems that involve multiple queries, dynamic obstacles, sensing and prediction uncertainties, nonholonomic planning, tracking controller design, or safety clearance restrictions. The key idea is to fill the collision-free subset of the configuration space with overlapping spherical balls, and define collision-free potential functions on each ball. A similar idea has been developed for collision detection in [23], and for navigation in [24]. Topological information is captured by an underlying connectivity graph called a Random Neighborhood Graph (RNG). The following desirable properties are achieved during execution once an RNG has been constructed: 1) a global potential function (with no local minima) can be defined over most of the configuration space; 2) if the goal changes, the RNG can be quickly reconfigured to represent a new global potential function that guides the robot to the new goal (this is analogous to the multiple-query property in path planning algorithms [10]); 3) the path taken by the robot can be significantly adjusted and adapted on-line, while easily avoiding collisions; 4) efficient algorithms from computational geometry can be exploited to quickly extract motion commands in real time [21]. We present an RNG construction algorithm that has the following properties: 1) it exploits information computed from existing, efficient distance computation algorithms (e.g., [4, 5, 16, 22, 23]); 2) the use of randomization overcomes many of the pitfalls of high dimensionality; 3) techniques from computational geometry can be exploited to efficiently construct the RNG with little sensitivity to dimension; 4) a Bayesian stopping condition guarantees that a specified fraction of the space is covered with a specified probability.

2 Problem Formulation

Assume that a robot moves in a bounded 2D or 3D world, $\mathcal{W} \subset \Re^N$, such that $N = 2$ or $N = 3$. An n-dimensional *configuration vector*, q, captures position, orientation, joint angles, and/or other information for robot. Let $\mathcal{C}$, be the configuration space (i.e., the set of all possible configurations). Let $\mathcal{A}(q)$ denote the set of points in $\mathcal{W}$ that are occupied by the robot when it is in configuration q. Let $\mathcal{O} \subset \mathcal{W}$ denote a static *obstacle region* in the world. Let $\mathcal{C}_{free}$ denote the set of configurations, q, such that $\mathcal{A}(q) \cap \mathcal{O} = \emptyset$.

The task is to find a motion strategy that uses feedback and guides the robot to any goal configuration from any initial configuration, while avoiding collisions. For a given goal, q_g, this can be accomplished by defining a real-valued *navigation function*, $U : \mathcal{C}_{free} \rightarrow \Re$ that has a single local minimum, which is at q_g. A navigation function can be considered as a special case of an artificial potential function [11] that avoids the pitfalls of becoming trapped in local minima. The robot is guided to the goal by following directions given by the negative gradient of U.

The task can generally be divided into two stages: constructing a representation of $\mathcal{C}_{free}$, and construction a navigation function over this representation. the first stage is performed only once, while the second stage may be iterated many times, for a variety of changing goals. In general, it is too difficult to obtain a navigation function defined over all of $\mathcal{C}_{free}$. Instead, the task is to build navigation functions over as much of $\mathcal{C}_{free}$ as possible. Assume $\mathcal{C}_{free}$ is bounded. Let $\mu(X)$ denote the measure (or n-dimensional volume) of a subset of $\mathcal{C}_{free}$ (obviously the measure is sensitive to the parameterization of the configuration space). For a given $\alpha \in (0, 1)$, and a probability, P, the first phases consists of building a data structure that fills $\mathcal{C}_{free}$ with a set $\mathcal{B} \subset \mathcal{C}_{free}$, such that $\mu(\mathcal{B})/\mu(\mathcal{C}_{free}) \geq \alpha$ with probability P. As goals are changed, it must be possible to efficiently recompute a new navigation function. This operation motivates our construction of a Random Neighborhood Graph, which is described in Section 3.

3 Random Neighborhood Graph

A Random Neighborhood Graph (RNG) is an undirected, graph, $G = (V, E)$, in which V is the set of vertices and E is the set of edges. Each vertex represents an n-dimensional ball that lies entirely in $\mathcal{C}_{free}$. For any vertex, v, let c_v denote the center of its corresponding ball, r_v denote the radius of its ball, and let B_v be the set of points,

$$B_v = \{q \in \mathcal{C} \mid \|q - c_v\| \leq r_v\}.$$

We require that $B_v \subset \mathcal{C}_{free}$. The definition of B_v assumes that $\mathcal{C}$ is an n-dimensional Euclidean space; however, minor modifications can be made to include other frequently-occurring topologies, such as $\Re^2 \times S^1$ and $\Re^3 \times P^3$.

An edge, $e \in E$, exists for each pair of vertices, v_i, and v_j, if and only if their balls intersect, $B_i \cap B_j \neq \emptyset$. Assume that no balls are contained within another ball, $B_i \not\subseteq B_j$, for all v_i and v_j in V. Let $\mathcal{B}$ represent the subset of $\mathcal{C}_{free}$ that is occupied by balls,

$$\mathcal{B} = \bigcup_{v \in V} B_v.$$

Suppose that the graph G has been given; an algorithm that constructs G is presented in Section 4. For a given goal, the RNG will be used to represent a feedback strategy, which can be encoded as a real-valued navigation function, $\gamma : \mathcal{B} \rightarrow \Re$. This function will have only one minimum, which is at the goal configuration. If the goal changes, it will also be possible to quickly "reconfigure" the RNG to obtain a new function, γ', which has its unique minimum at the new goal.

Let G be a weighted graph in which $l(e)$ denotes the cost assigned to an edge $e \in E$. Assume that $1 \leq l(e) < \infty$ for all $e \in E$. The particular assignment of costs can be used to induce certain preferences on the type of solution (e.g., maximize clearance, minimize distance traveled). Let B_{v_g} denote any ball that contains the goal, q_{goal}, and let v_g be its corresponding vertex in G. Let $L^*(v)$ denote be the optimal cost to reach v_g from v. The optimal costs can be recomputed for each vertex in V in $O(V^2)$ or $O(V \lg V + E)$ time using Dijkstra's algorithm; alternatively, an all-pairs shortest paths algorithm can be used to implicitly define solutions for all goals in advance.

Assume that G is connected; if G is not connected, then the following discussion can be adapted to the connected component that contains v_g. Define a strict linear ordering, $<_v$, over the set of vertices in V using $L^*(v)$ as follows. If $L^*(v_1) < L^*(v_2)$ for any $v_1, v_2 \in V$, then $v_1 <_v v_2$. If $L^*(v_1) = L^*(v_2)$, then the ordering of v_1 and v_2 can be defined in an arbitrary way, while ensuring that $<_v$ remains a linear ordering. The ordering $<_v$ can be adapted directly to the set of corresponding balls to obtain an ordering $<_b$ such that: $B_{v_1} <_b B_{v_2}$ if and only if $v_1 <_v v_2$. Note that the smallest element with respect to $<_b$ always contains the goal.

For a given goal, the RNG will be used to represent a mapping $\gamma : \mathcal{B} \rightarrow \Re$ that serves as a global potential or navigation function. For each vertex, $v \in V$, let $\gamma_v : B_v \rightarrow \Re$ represent a *partial strategy*. Among all balls that intersect B_v, let B_{v_m} denote the ball that is minimal with respect to $<_b$. It is assumed that γ_v is a differentiable function that attains a unique minimum a point in the interior of $B_v \cap B_{v_m}$. Intuitively, each partial strategy guides the robot to a ball that has lower cost.

The partial strategies are combined to yield a global strategy in the following way. Any configuration, $q \in \mathcal{B}$, will generally be contained in multiple balls. Among these balls, let B_v be the minimal ball with respect to $<_b$ that contains q. The navigation function at q is given by $\gamma_v(q)$, thus resolving any ambiguity. Note that the robot will typically not reach the minimum of a partial strategy before "jumping" to a ball that has a lower cost with respect to $<_b$.

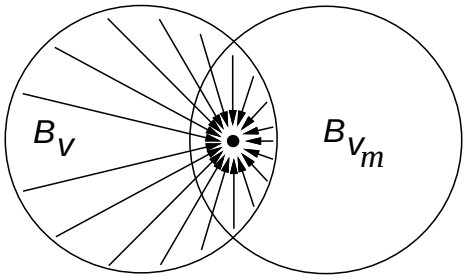

Figure 2: The negative gradient of a partial navigation function sends the robot to a lower-cost ball.

```
GENERATE_RNG(α,P_c)
 1   G.init(q_init);
 2   while (TerminationUnsatisfied(G,α,P_c) do
 3       repeat
 4           q_new ← RandomConf(G);
 5           d ← DistanceComputation(q_new);
 6       until ((d > 0) and (q_new ∉ B))
 7       r ← ComputeRadius(d);
 8       v_new ←G.AddVertex(q_new, r);
 9       G.AddEdges(v_new);
10   G.DeleteEnclaves();
11   G.DeleteSingletons();
12   Return G
```

Figure 3: This algorithm constructs the RNG and determines automatically when to terminate based on estimated coverage of $\mathcal{C}_{free}$.

4 RNG Construction Algorithm

An outline of the RNG construction algorithm is given in Figure 3. The inputs are $\alpha \in (0, 1)$ and $P_c \in (0, 1)$ (the obstacle and robot models are implicitly assumed). For a given α and P_c, the algorithm will construct an RNG such that with probability P_c, the ratio of the volume of $\mathcal{B}$ to the volume of $\mathcal{C}_{free}$ is at least α.

Each execution of Lines 3-9 corresponds to the addition of a new ball, $B_{v_{new}}$, to the RNG. This results in a new vertex in G, and new edges that each corresponds to another ball that intersects $\mathcal{B}_{v_{new}}$. Balls are added to the RNG until the Bayesian termina-

tion condition is met, causing TerminationUnsatisfied to return FALSE. The Bayesian method used in the termination condition, RelativelyUnexplored, is presented in Section 4.2. The **repeat** loop from Lines 3 to 6 generates a new sample in $\mathcal{C}_{free} \setminus \mathcal{B}$, which might require multiple iterations. Collision detection and distance computation are performed in Line 5. Many algorithms exist that either exactly compute or compute a lower bound on the closest distance in $\mathcal{W}$ between $\mathcal{A}$ and $\mathcal{O}$ [16, 22, 23]

$$d(q_{new}) = \min_{a \in \mathcal{A}(q_{new})} \min_{o \in \mathcal{O}} \|a - o\|$$

If d is not positive, then q_{new} is in collision, and another configuration is chosen. The new configuration must also not be already covered by the RNG before the *repeat* loop terminates. This forces the RNG to quickly expand into $\mathcal{C}_{free}$, and leads to few edges per vertex in G.

Distance computation algorithms are very efficient in practice, and their existence is essential to our approach. The distance, d, is used in Line 7 by the ComputeRadius function, which attempts to select r to create the largest possible ball that is centered at q_{new} and lies entirely in $\mathcal{C}_{free}$. A general technique for choosing r is presented in Section 4.1.

The number of iterations in the **while** loop depends on the Bayesian termination condition, which in turn depends on the outcome of random events during execution and the particular $\mathcal{C}_{free}$ for a given problem. The largest two computational expenses arise from the distance computation and the test whether q_{new} lies in $\mathcal{B}$. Efficient algorithms exist for both of these problems. Hierarchical representations [23] and Voronoi structure [16, 17, 22] can be exploited to efficiently compute distance. The test whether q_{new} lies in $\mathcal{B}$ can be considered as a *point location* problem, for which very efficient algorithms exist, which approach computation time logarithmic in the number of balls, regardless of dimension [21].

4.1 Radius selection

For a given q_{new}, the task is to select the largest radius, r, such that the ball

$$B_v = \{q \in \mathcal{C} \mid \|q_{new} - q\| \leq r\}$$

is a subset of $\mathcal{C}_{free}$. If DistanceComputation(q_{new}) returns d, then $\max_{a \in \mathcal{A}} \|a(q_{new}) - a(q)\| < d$ for all $q \in B_v$ implies that $B_v \subset \mathcal{C}_{free}$. For many robots one can determine a point, a_f, in $\mathcal{A}$ that moves the furthest as the configuration varies. For a rigid robot, this is the point that would have the largest radius if polar or spherical coordinates are used to represent $\mathcal{A}$. There exists a positive constant, r, such that $\|q_{new} - q\| < r$ implies $\|a_f(q_{new}) - a_f(q)\| < d$. The goal is to make r as large as possible to make the RNG construction algorithm more efficient. The largest value of r is greatly affected by the parameterization of the kinematics. For example, if a_f is far from the origin, points on the robot will move very quickly as the rotation angle changes. For a 2D rigid robot with translation and rotation, $\mathcal{C} = \Re^2 \times S^1$, and the following homogeneous transformation can be used for the kinematics

$$\begin{pmatrix} x_{new} \\ y_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(q_3/r_m) & -\sin(q_3/r_m) & q_1 \\ \sin(q_3/r_m) & \cos(q_3/r_m) & q_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1}$$

in which $r_m = \|a_f(0)\|$, q_1 is x-translation, q_2 is y-translation, and q_3 is a scaled rotation. Using this representation it can be shown that $r = d$. If the standard parameterization of rotation was used, the effects of rotation would dominate, resulting in a smaller radius, $r = d/r_m$. Although the relative fraction of S^1 that is covered is the same in either case, the amount of $\Re^2$ that is covered is increased substantially.

Although many alternatives are possible, one general methodology for selecting r for various robots and configuration spaces is to design a parameterization by bounding the arc length. Let $f : \Re^n \to \Re^m$ denote the expression of the kinematics that maps points from an n-dimensional configuration space to an mD world (a simple of example of f is (1)). In general, arc length in the world, based on differential changes in configuration, is specified by a metric tensor

$$ds^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{\partial f_i}{\partial q_j} \frac{\partial f_i}{\partial q_k} \, dq_j \, dq_k.$$

If the transformation f is orthogonal, then the arc length simplifies to

$$ds^2 = \sum_{i=1}^{n} \left\| \frac{\partial f(x_1, \ldots, x_m)}{\partial q_i} \right\|^2 dq_i^2, \tag{2}$$

in which each term represents the squared magnitude of a column in the Jacobian of f. Using the bound $d^2 < ds^2$, (1) expresses the equation of a solid ellipsoid in the configuration space. If every term is one, then $c = d$, which occurred by design using (1). The key is to choose kinematic expressions that keep the eccentricity as close as possible to representing a sphere. For problems that involve articulated bodies, it is preferable to derive expressions that consider the distance in the world of each rigid body. For the results in Section 5, we use this general technique for 2D rigid bodies, and 2D articulated bodies.

4.2 A Bayesian termination condition

The algorithm in Figure 3 decides to terminate based on a statistical estimate of the fraction of $\mathcal{C}_{free}$

that is covered by the RNG. The volumes of $\mathcal{C}_{free}$ and $\mathcal{B}$, denoted by $\mu(\mathcal{C}_{free})$ and $\mu(B)$ are assumed unknown. Although it is theoretically possible to incrementally compute $\mu(B)$, it is generally too complicated to compute. A Bayesian termination condition can be derived based on the number of samples that fall into $\mathcal{B}$, as opposed to $\mathcal{C}_{free} \setminus \mathcal{B}$. For a given α and P_c, the algorithm will terminate when 100α percent of the volume of $\mathcal{C}_{free}$ has been covered by the RNG with probability P_c.

Let $p(x)$ represent a probability density function that corresponds to the fraction $\mu(\mathcal{B})/\mu(\mathcal{C}_{free})$. Let $y_1, y_2, \ldots, y_k$ represent a series of k observations, each of which corresponds for a random configuration, drawn drawn from $\mathcal{C}_{free}$. Each observation has two possible values: either the random configuration, q_{new}, is in $\mathcal{B}$ or in $\mathcal{C}_{free} \setminus \mathcal{B}$. Let $y_k = 1$ denote $q_{new} \in \mathcal{B}$, and let $y_k = 0$ denote $x_{new} \in \mathcal{C}_{free} \setminus \mathcal{B}$.

For a given α and P_c, we would like to determine whether $P[x > \alpha] \geq P_c$. The left side can be computed from

$$P[x > \alpha] = \int_{\alpha}^{1} p(x \mid y_1, \ldots, y_k). \tag{3}$$

The integrand above is given by starting with a prior density function $p(x)$, and iteratively applying Bayes' rule for each sample. This yields

$$p(x \mid y_1, \ldots, y_k) = \frac{p(y_k|x)\ p(x \mid y_1, \ldots, y_{k-1})}{\int_0^1 p(y_k \mid x)\ p(x \mid y_1, \ldots, y_{k-1}) dx}.$$

Assume that the prior $p(x)$ is a uniform density over $[0, 1]$. For any k, if $y_k = 1$, then $p(y_k|x) = x$. Otherwise, $p(y_k|x) = 1 - x$. For a sequence of k samples, suppose $y_k = 1$ for i of these samples, and $y_k = 0$ for the remaining $k - i$. Then

$$p(x \mid y_1, \ldots, y_k) \propto x^i (1 - x)^{k-i}$$

The constant of proportionality is somewhat complicated, except in the case in which all samples lie in $\mathcal{B}$. For simplicity, we use this this set of observations, even though other observation sets can theoretically be used (in fact there are a variety of other methods that would allow more efficient termination). Thus, $y_k = 1$ for a chain of k successive samples. The posterior probability density is $p(x|y_1, \ldots, y_k) = (k + 1)x^k$. Using (3), $P[x > \alpha] = 1 - \alpha^{k+1}$.

The algorithm terminates when the number of successive samples that lie in $\mathcal{B}$ is k, such that $\alpha^{k+1} \leq 1 - P_c$. One can solve for k and the algorithm will terminate when $k = \frac{\ln(1-P_c)}{\ln \alpha} - 1$. During execution, a simple counter records the number of consecutive samples that fall into $\mathcal{B}$ (ignoring samples that fall outside of $\mathcal{C}_{free}$).

5 An Implementation with Examples

We have implemented the RNG construction algorithm in Gnu C++ using the LEDA library on a Pentium III 500 Mhz PC running Linux. A variety of experiments have been performed for robots in 2D environments and up to five degrees of freedom.

Figure 1 shows the balls of the RNG for a point robot in a 2D environment. Figure 4.a shows the RNG edges as line segments between ball centers. The RNG construction required 23s, and the algorithm terminated after 500 successive failures ($k = 500$) to place a new ball. The RNG contained 535 nodes, 525 of which are in a single connected component. There were 1854 edges, resulting in an average of only 3.46 edges per vertex. We have observed that this number remains low, even for higher-dimensional problems. This is an important feature for maintaining efficiency because of the graph search operations that are needed to build navigation functions. Computation time can be significantly improved by weakening the requested percentage of coverage. For example, when $k = 50$, only 0.2s was needed to construct the RNG. Most of the computation time is wasted when the space is mostly covered, and it is very difficult find new ball centers at random that do not lie in $\mathcal{B}$. Part of the problem is that we perform naive $O(n)$ time point location, which can be easily improved. We also believe that specialized sampling methods can be developed to attempt to obtain ball centers that are not already covered. Figures 4.a and 4.b show level sets of two different potential functions that were quickly computed for two different goal (each in less than 10ms). The first goal is in the largest ball, and the second goal is in the upper right corner. Each ball will guide the robot into another ball, which is one step closer to the goal. Using this representation, the particular path taken by the robot during execution is not critical. Figure 4.d shows another result for a 2D point robot. In this case, the robot travels into a narrow corridor, and the RNG contains a single connected component.

For higher-dimensional configuration spaces, we only show robot trajectories, even though much more information is contained in the RNG. Figure 5 shows a complicated path followed by the robot for an RNG that was computed for a 2D rigid robot that can rotate and translate in a 2D environment. The RNG took an hour to construct. Figure 6 shows paths that were obtained by constructing an RNG in a 5D configuration space for an articulated robot that consists of three links. The computation time to construct this RNG was about twenty minutes. Significant improvement can be made to the computation times by designing a better termination condition and by performing more efficient point location.

Recall that using the RNG, potential functions with no local minima can be constructed for any configura-

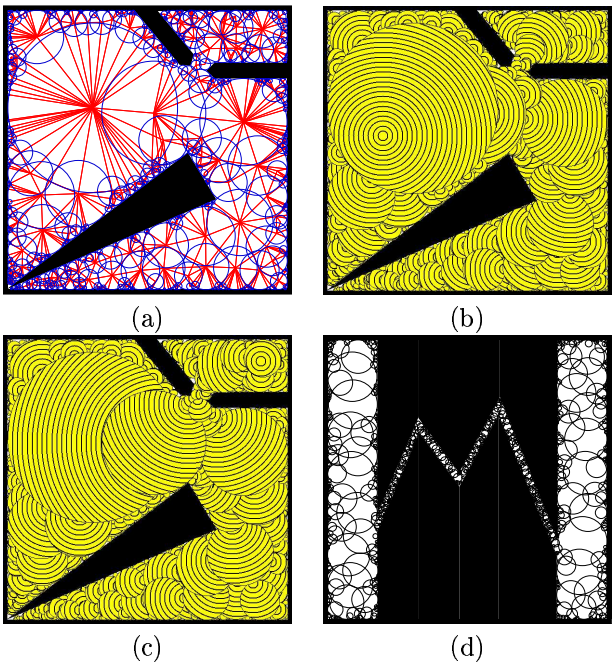

Figure 4: (a) The RNG for a 2D point robot (b), (c) two navigation functions computed from a single RNG; (d) another example for a 2D point robot.

tion that is covered by the balls. Thus, it is important not to compare the computation times with those obtained by standard path planning algorithms. Instead of determining a path or a network of paths, we build a data structure that can be repeatedly used many times for different execution tasks.

6 Discussion

We have introduced a randomized planning method that computes feedback motion strategies based on the Random Neighborhood Graph. The choice of simple geometric primitives enables the efficient construction of a volumetric representation of $\mathcal{C}_{free}$, and a simple Bayesian termination condition. Because navigation functions can be quickly computed and recomputed on an RNG, our work is expected to have applications to robotics systems that involve changing environments, moving obstacles, unexpected obstacles, sensing uncertainties, and uncertainties in control.

Our preliminary implementation and experimental results have been encouraging; however, more experimentation is needed, especially in higher-dimensional configuration spaces. This transition to higher dimensions is expected to proceed smoothly because our representation is not sensitive to dimension [21], in comparison to a traditional cell decomposition method such as 2^n trees [13]. The key difficulty will be selecting a good parameterization of $\mathcal{C}$ to increase the likelihood that large balls can be placed at random in $\mathcal{C}_{free}$. One possible improvement for higher dimensions is to combine cylinders with balls; for example, a region might exist in $\mathcal{C}$ in which the robot is able to rotate freely without collision. In this case, it might be preferable to allow any value for orientation, which generates a cylinder. Another reasonable alternative is to define rectangular neighborhoods, as opposed to spheres.

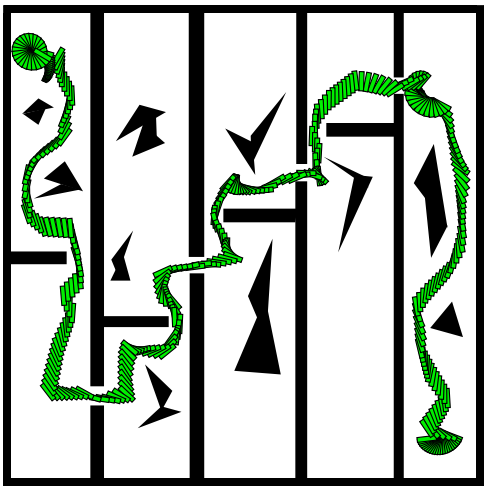

Figure 5: A path constructed from a 3D RNG, for a rigid robot.

Our method is expected to suffer, along with other randomized planning methods, from the narrow passage problem [6]. Although our algorithm converges in volume to a covering of $\mathcal{C}_{free}$, it does not indicate the probability that the topology of $\mathcal{B}$ will follow the topology of $\mathcal{C}_{free}$. Randomized path planning techniques, such as probabilistic roadmaps [10], rapidly-exploring random trees [15, 14], or even random walks might be helpful to locate locations for balls that will improve RNG performance in narrow passages. Another issue to address is the increasing number of wasted samples that are generated as the RNG covers a larger fraction of $\mathcal{C}_{free}$. With improved sampling methods, it might be possible to significantly improve computation times.

Acknowledgments

This work was funded in part by NSF CAREER Award IRI-9875304 (LaValle). We thank Ahmad Massoud for a helpful discussion.

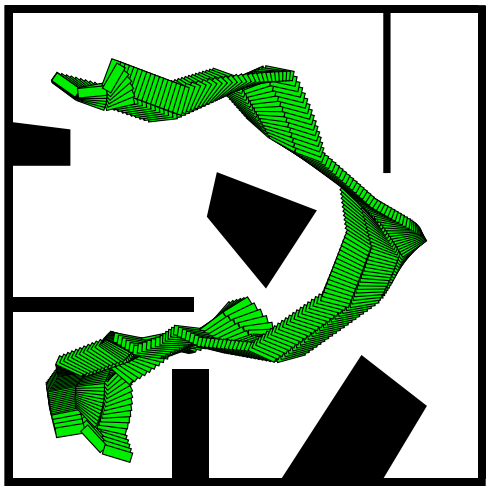

Figure 6: A path for an articulated robot, constructed from a 5D RNG.

References

[1] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE Int. Conf. Robot. & Autom.*, pages 113–120, 1996.

[2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, December 1991.

[3] C. Connolly and R. Grupen. The application of harmonic potential functions to robotics. *J. Robotic Systems*, 10(7):931–946, 1993.

[4] E. G. Gilbert and D. W. Johnson. Distance functions and their application to robot path planning in the presence of obstacles. *IEEE Trans. Robot. & Autom.*, 1(1):21–30, March 1985.

[5] L. J. Guibas, D. Hsu, and L. Zhang. H-Walk: Hierarchical distance computation for moving convex bodies. In *Proc. ACM Symposium on Computational Geometry*, pages 265–273, 1999.

[6] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In et al. P. Agarwal, editor, *Robotics: The Algorithmic Perspective*, pages 141–154. A.K. Peters, Wellesley, MA, 1998.

[7] H. Jacob, S. Feder, and J. Slotine. Real-time path planning using harmonic potential functions in dynamic environment. In *IEEE Int. Conf. Robot. & Autom.*, pages 874–881, 1997.

[8] I. Kamon and E. Rivlin. Sensory-based motion planning with global proofs. *IEEE Trans. Robot. & Autom.*, 13(6):814–822, December 1997.

[9] I. Kamon, E. Rivlin, and E. Rimon. Range-sensor based navigation in three dimensions. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[10] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. & Autom.*, 12(4):566–580, June 1996.

[11] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[12] R. Kimmel, N. Kiryati, and A. M. Bruckstein. Multivalued distance maps for motion planning on surfaces with moving obstacles. *IEEE Trans. Robot. & Autom.*, 14(3):427–435, June 1998.

[13] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[14] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Computer Science Dept., Iowa State University. <http://janowiec.cs.iastate.edu/papers/rrt.ps>, Oct. 1998.

[15] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, 1999. To appear.

[16] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Int. Conf. Robot. & Autom.*, 1991.

[17] M. C. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 129–142. A K Peters, Wellesley, MA, 1997.

[18] V. J. Lumelsky and T. Skewis. A paradigm for incorporating vision in the robot navigation function. In *IEEE Int. Conf. Robot. & Autom.*, pages 734–739, 1988.

[19] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[20] A. Massoud. Robot navigation using the vector potential approach. In *IEEE Int. Conf. Robot. & Autom.*, pages 1:805–811, 1993.

[21] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM*, 44(1):1–29, January 1997.

[22] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electronics Research Laboratory, 1997.

[23] S. Quinlan. Efficient distance computation between nonconvex objects. In *IEEE Int. Conf. Robot. & Autom.*, pages 3324–3329, 1994.

[24] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *IEEE Int. Conf. Robot. & Autom.*, pages 802–807, 1993.

[25] S. Ratering and M. Gini. Robot navigation in a known environment with unknown moving obstacles. In *IEEE Int. Conf. Robot. & Autom.*, pages 25–30, 1993.

[26] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Trans. Robot. & Autom.*, 8(5):501–518, October 1992.

[27] A. M. Shkel and V. J. Lumelsky. Incorporating body dynamics into sensor-based motion planning: The maximum turn strategy. *IEEE Trans. Robot. & Autom.*, 13(6):873–880, December 1997.

[28] S. Sundar and Z. Shiller. Optimal obstacle avoidance based on the Hamilton-Jacobi-Bellman equation. *IEEE Trans. Robot. & Autom.*, 13(2):305–310, April 1997.