The Sampling-Based Neighborhood Graph: An Approach to Computing and Executing Feedback Motion Strategies

Libo Yang Steven M. LaValle

Abstract—
This paper presents a sampling-based approach to computing and executing feedback motion strategies by defining a global navigation function over a collection of neighborhoods in configuration space. The collection of neighborhoods and their underlying connectivity structure are captured by a Sampling-based Neighborhood Graph (SNG), on which navigation functions are built. The SNG construction algorithm incrementally places new neighborhoods in the configuration space using distance information provided by existing collision detection algorithms. A termination condition indicates the probability that a specified fraction of the space is covered. Our implementation illustrates the approach for rigid and articulated bodies with up to six-dimensional configuration spaces. Even over such spaces, rapid on-line responses to unpredictable configuration changes can be made in a few microseconds on standard PC hardware. Furthermore, if the goal is changed, an updated navigation function can be quickly computed without performing additional collision checking.

Keywords: Motion planning, Navigation functions, Potential fields, Feedback control

Submission Type: Regular paper

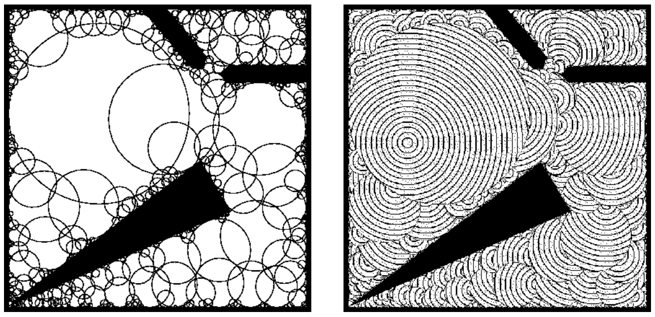
I. Introduction

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, such as the BUG paradigm [24], [25], [40], [39], [54] and potential field approaches [12], [22], [28], [29], [42], [50], [56].

We propose a sampling-based framework for generating feedback motion strategies (see Figure 1) for robots with many degrees of freedom. The work presented here incorporates and expands the work reported in [59], [60]. The key idea of this work is to fill the collision-free subset of the configuration space with overlapping neighborhoods, such as balls, and define a collision-free potential function on each neighborhood (a similar covering of neighborhoods was applied to trajectories in [47], [48]).

Libo Yang is with the Dept. of Computer Science, Iowa State University, Ames, IA 50011 USA. E-mail: lyang@cs.iastate.edu

Steven M. LaValle (corresponding author) is with the Dept. of Computer Science, University of Illinois, Urbana, IL 61801 USA. E-mail: lavalle@cs.uiuc.edu, Phone: +1-217-265-6313



(a) Coverage of $\mathcal{C}_{free}$ (b) A navigation function defined over it

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

Over the neighborhoods, the task is to compute a *navigation function*, which is a potential function that has only one local minimum, which is at the goal [49]. To achieve this, topological information is captured by an underlying connectivity graph. We refer to the resulting structure as a Sampling-based Neighborhood Graph (SNG); it has two main advantages:

1. A navigation function can be defined over most of the configuration space, enabling rapid response to unpredictable changes in the configuration. Such unpredictability could occur due to calibration errors, modeling errors, and disturbances. Based on our experiments, such responses are made in a few microseconds for a six-dimensional navigation function, implemented on a 500MHz PC. This is much faster than what can be obtained by replanning with existing path planning algorithms.
2. The navigation function is quickly reconfigurable for a given SNG. If the goal changes a small amount (remaining in the same neighborhood), the navigation function can be recomputed in constant time. This could be useful, for example, in target tracking applications. If the goal changes substantially, then a new navigation function can be quickly computed from scratch by performing graph search over the SNG without additional collision checking. This latter computation is similar to the benefit obtained in the multiple-query paradigm introduced in [26] for path planning.

To obtain the advantages above, efficient construction of an SNG of critical importance. We present a construction algorithm that has the following properties: 1) it exploits information computed from existing, efficient distance computation algorithms (e.g., [17], [18], [35], [44], [47]); 2) techniques from computational geometry can be exploited to efficiently construct the SNG with little sensitivity to dimension (for example, [43] enables $O(\lg n)$ expected running time to locate a configuration among a collection of balls, and $O(n \lg n)$ expected running time to compute the neighborhood-connectivity structure regardless of dimension); 3) a statistical stopping condition indicates that a specified fraction of the space is covered with a specified probability. Deterministic guarantees of coverage may alternatively be possible by applying other sampling methods [6], [33].

II. Related Work

Feedback strategies as funnels. In previous work, feedback motion strategies were considered as funnels that passively guide any state in the domain into the goal state. The notion of a funnel was introduced by Mason nearly two decades ago [41]. In his work, a funnel is a positive definite function that is centered at a goal point, which represents the only zero and unique local minimum over the funnel's domain of attraction. If from any state in the domain, the resulting feedback control law that follows the negative gradient of the funnel converges to the goal, then the funnel defines a Lyapunov function. Similar ideas have also been introduced by other researchers in robotics and fine-motion planning, including Ish-Shalom [21]; Lozano-Pérez, Mason, and Taylor [38]. Burridge, Rizzi, and Koditschek [10] extended Mason's idea. In their definition, a funnel represents an attraction region of a dynamical system within a known invariant region. Burridge's funnel recognizes the "obstacle", in which all states must be avoided, and adapts itself to a complicated domain that avoids the obstacle. Although each funnel represents a Lyapunov function, it may be difficult to find. Moreover, there is a large and important class of systems for which no single funnel (continuous stabilizing feedback control law) exists [8], [31].

The sequential composition of funnels: Deployments. Due to difficulty of finding a single funnel, a sequential composition of funnels was introduced. Lozano-Pérez, Mason, and Taylor [38] introduced the notion of preimage backchaining. There is a collection of controllers whose induced funnels each has a local goal set that is either contained in a another controller's domain or in the global task goal. In this case, it is possible to *backchain* away from the global task goal [14]. Such a backchain is also called a deployment [10]. A deployment partitions the state space into cells. Inside each cell, a different controller will become active. As each controller drives the system toward its local goal, the state crosses a boundary into a region of state space where another controller is active. This process is repeated until the state reaches the final cell, which is the only one that contains the goal. Using backchaining, a strategy is derived that switches between controllers to drive any initial state from the union of all cell domains into the goal state.

Potential fields. Artificial potential fields are an efficient approach to feedback motion generation. In traditional path planning approaches, the motion planning problem is usually decomposed into three parts: 1) plan the collision-free path; 2) transform this path into a trajectory that satisfies the dynamics; and 3) apply a feedback control law that guides the robot to the goal. Unlike traditional path planning approaches, the potential field approach solves the problem in one step. The potential field idea was originally developed as an on-line collision avoidance approach by Khatib in his Ph.D. dissertation [27]. Later, several approaches were developed to compute the potential field [1], [5], [57]. Potential field methods typically suffer from local minima and are often referred to as "local methods". Koditschek showed that a *global navigation function* does not exist in general [30]. However, it is possible to define an "almost global navigation function" (or *navigation function*) that has a minimum located at q_{goal} and a set of saddle points that are unstable configurations. Any small perturbation allows the planner to evade them. Much research has been done on computing a navigation function [4], [13], [29], [34], [49], [50], [56]. Rimon and Koditschek constructed a navigation function for a generalized sphere world for a point-mass robot [49], [50]. Sundar and Shiller proposed a pseudo-return function for a point robot in a workspace with circular obstacles [56]. Level-set methods have also been proposed to compute navigation functions [29]. LaValle and Konkimalla presented a dynamic programming approach to computing feedback strategies for nonlinear systems [34]. The main difficulty with general-purpose navigation function methods to date is that they are limited to low-dimensional problems.

Decomposition-based path planning. One straightforward approach of defining a navigation function is to use cell decomposition methods [3], [11], [53], [51], [52], [37], [9], [32], [15], [23], [61], by combining the partial potential functions each of which is defined over each cell. However, it is difficult to do so for the problems of interest in this paper. Usually, cell decomposition methods suffer in high dimensions, especially when the models of the robot and obstacle have high complexity. Another disadvantage is that the cells do not have overlapping interiors. This increases the difficulty of switching from one control law to another, which can be done in an open set for the case of overlapping regions.

An interesting approach that is similar to ours (developed in parallel) in that overlapping neighborhoods are used to build a potential function is the recent work of Brock and Kavraki [7]. This leads to fast performance in many cases, but the primary difference lies in the space in which neighborhoods are considered. They develop neighborhoods in the workspace, and in our case, we develop neighborhoods in the configuration space. The tradeoff is that our method is more expensive, but it is able to avoid local minima that arise from neglecting the $\mathcal{C}_{free}$ topology.

III. Problem Formulation

Assume that a robot moves in a compact 2D or 3D world, $\mathcal{W} \subset \mathbb{R}^N$, such that $N = 2$ or $N = 3$. An n-dimensional *configuration vector*, q, captures position, orientation, nd/or joint angles. Let $\mathcal{C}$ be the n-dimensional *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, closed *obstacle region.* Let $\mathcal{C}_{free}$ denote the set of configurations, q, such that $\mathcal{A}(q) \cap \mathcal{O} = \emptyset$.

We generally assume that there are some factors, such as external disturbances, calibration, or modeling errors, which prevent the motions of the robot from being completely predictable. The robot does, however, have an ideal sensor that can measure the current configuration. A discrete-time control problem is assumed in which a sensor provides measurements, and the robot chooses a control based on the sensor output. Thus, it can incorporate feedback to determine subsequent motion commands, even if the configuration drifts from expectations. The study of particular control systems is beyond the scope of this algorithmic paper. It will generally be assumed that the system is small-time controllable, and there exists a control law that can be implemented to perform gradient descent of a navigation function.

The robot control model can be described as $x_{k+1} = f(x_k, u_k)$, in which x_k is the state (usually taken from the tangent bundle of $\mathcal{C}$) at time step k, and u_k is an input chosen from a predetermined set. The model can be extended to explicitly account for prediction uncertainty as $x_{k+1} = f(x_k, u_k, \theta_k)$, in which θ_k represents an unknown disturbance chosen from a known set. At each time step, the robot must determine an action u_k; however, the future state is unpredictable due to the disturbance. In this case, an open-loop control that simply specifies the input sequence is insufficient.

The task is to find a motion strategy that uses feedback from a configuration sensor, and computes inputs that guide the robot to a goal configuration while avoiding collisions. For a given goal, q_{goal}, this can be accomplished by defining a real-valued *navigation function*, $\gamma : \mathcal{C}_{free} \rightarrow \mathbb{R}$ that has a single stable local minimum, which is at q_{goal}. It is assumed that the controller can use this function to determine the appropriate input. The robot is guided to the goal in each iteration by choosing an input, u_k, from the current configuration, q_k, that reduces $\gamma(q_{k+1})$ as much as possible. In this work, we do not explicitly model the disturbance, θ_k. This is common in classical feedback control; from the system formulation, it appears that feedback is unnecessary; however, it is still assumed that unpredictabilities can occur. To explicitly model bounded disturbances, our approach could be modified by appropriately "growing" the obstacles by an amount that ensures disturbances will not lead to collision.

Finally, we assume that q_{goal} is frequently changed for a fixed environment. In this case, we would like a *multiple-query* approach to building navigation functions. This will be accomplished by the SNG in a manner that avoids using collision detection if the goal is changed.

IV. A General Framework

There are three separate phases of computation to keep in mind.

Precomputation: Construction an SNG for a new environment.
Building a Navigation Function: Given the SNG and a goal, construct a navigation function.
Execution: Given a navigation function over an SNG, iteratively compute motion commands from an initial state until the goal is reached.
For the first two phases, the philosophy is similar to the multiple-query approach used for the probabilistic roadmap (PRM) [26]. We precompute a data structure under the assumption that the goal configuration will frequently change, while the geometry remains the same. However, instead of computing a path in the second phase, our approach computes a navigation function. Also, note that we focus on a third phase, which is where the key advantages of the SNG emerge. Rapid response to unpredictable changes in configuration can be made because we represent a navigation function. Thus, a feedback motion strategy is obtained, as opposed to an open-loop path.

Each of the subsections below describes one of the three phases. The construction of the SNG is by far the most difficult of the three; therefore, the SNG is defined in IV-A, and the construction algorithm is deferred until Sections V to VII. Sections IV-B and IV-C describe the remaining two phases and their associated algorithms. These primarily emphasize the use of the SNG, assuming that it has already been precomputed.

A. *The Sampling-based Neighborhood Graph*

Our approach can be considered as a method to compute a deployment in complicated, high-dimensional configuration spaces. The first phase is to compute a collection of domains that cover as much of $\mathcal{C}_{free}$ as possible. The Sampling-based Neighborhood Graph (SNG) is defined as follows:

- Associated with the SNG is an undirected graph, $G = (V, E)$, in which V is the set of vertices and E is the set of edges.
- Each vertex, $v \in V$, represents a unique convex, open *neighborhood*, $B_v \subset \mathcal{C}_{free}$, and a point, $q_v \in B_v$, which is called the *center* of B_v. Nonconvex neighborhoods could be considered, as long as it is simple to ensure that they are collision free and to define a navigation function over them.
- It is assumed that no neighborhood is a subset of another neighborhood.
- An edge, $e \in E$, exists for each pair of vertices, v_i and v_j, if and only if their neighborhoods intersect, $B_i \cap B_j \neq \emptyset$.
- Let $\mathcal{B}$ be the union of all neighborhoods,

$$\mathcal{B} = \bigcup_{v_i \in V} B_i \ .$$

The *sampling-based* term arises from the fact the center of each neighborhood will be chosen according to some infinite sequence of samples that lie in $\mathcal{C}_{free}$. In practice, this sequence is terminated after a finite number of iterations, and the resulting SNG is considered as an approximate representation of $\mathcal{C}_{free}$ that is both volumetric and captures its connectivity.

We prefer sample sequences that enable convergence in measure in the following sense. Suppose that an SNG has been constructed using k samples. Let γ represent a bounded, real-valued navigation function over $\mathcal{C}_{free}$. Consider approximating γ with a function, γ_k, such that $\gamma(q) = \gamma_k(q)$ if $q \in \mathcal{B}$, and $\gamma(q) = 0$, otherwise. We want to use sample sequences that make γ_k converge in measure to γ as k tends to infinity. To achieve this, any sequence that is dense in $\mathcal{C}_{free}$ will suffice. This could, for example, be a deterministic sequence, such as Halton, Sobol', Faure, Niederreiter-Xing, or the extensible grid in [36]. Using a smooth probability density function over $\mathcal{C}_{free}$, a sequence of samples simulated from the density function will be dense with probability one. Although many reasonable choices are possible, we focus the implementation and analysis in this paper on uniform, random sampling to obtain probabilistic coverage bounds.

B. Constructing Navigation Functions

We now assume that an SNG and goal, $q_{goal} \in \mathcal{C}_{free}$, have been given, and the task is to construct a navigation function, $\gamma : \mathcal{B} \rightarrow \mathbb{R}$. Assume that $G(V, E)$ is augmented to include finite weights on all edges. These could all be unit weights, or values that correspond to desirability of the neighborhood transition, based on distance, clearance, etc.

Once q_{goal} has been given, the computation proceeds as follows:

1. The first step is to find a neighborhood, B_g, that contains it. Let v_g denote its corresponding vertex in V.
2. Starting with v_g, run an exhaustive search algorithm over G, such as breadth-first, depth first, or Dijkstra's algorithm. This results in *cost-to-go* values that are stored at every vertex. Let $c(v)$ denote the cost-to-go value at a vertex, v.
3. Consider the partial ordering on V obtained using the cost-to-go values. Construct a mapping, π, on V that assigns a unique positive integer to each vertex, in such as way that if $c(v_1) \leq c(v_2)$, then $\pi(v_1) < \pi(v_2)$. The mapping π can be considered as an assignment of strict *priorities* over the vertices in V (we interpret a lower value as having higher priority).
4. Over each neighborhood, B_v, with center q_v, we define a *local navigation function*, $\gamma_v : B_v \rightarrow [0, \infty)$. This could, for example, be any Lyapunov function or optimal cost-to-go function for a particular system. The zero value for γ_v is placed q_{vi}, which is chosen as a point in the highest-priority neighborhood that intersects B_v (see Figure 2). Thus, the local navigation function will try to guide the configuration into a neighborhood that is closer to the goal (closer in the sense of path cost in G). There is one special exception. The local navigation function for B_g places its zero point at q_{goal}, because there are no other balls with higher priority.
5. A global navigation function is defined over $\mathcal{B}$ by combining the local navigation functions and information from priorities. A configuration, q, may lie in one or more neighborhoods. Let $\mathcal{B}_q$ denote this collection of neighborhoods. The global navigation function utilizes the local navigation function from the highest-priority neighborhood in $\mathcal{B}_q$. Let $B_h(q)$ denote this neighborhood. To obtain a navigation function, one simply has to "activate" γ_h (defined for $B_h(q)$) for any $q \in \mathcal{B}$.

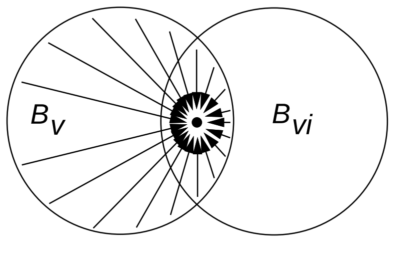


Fig. 2. Executing motions that decrease the values of a local navigation function sends the configuration to a higher-priority neighborhood.

To provide an example of an actual navigation function, we define a simple quadratic function, $\gamma_v(q) = \|q - q_{vi}\|^2$, which can be applied for any $v \in V$ over its corresponding B_v. For the neighborhood that contains the goal, we let $q_{vi} = q_{goal}$. This notation neglects topological issues which must also be considered, and assumes that for any neighborhood, a smooth coordinate chart exists that covers it. Let w denote an upper bound on the maximum value possible of γ_v, over all $v \in V$. This is proportional to the square of the maximum Euclidean distance between any pair of points within the same neighborhood. Using the priorities, a global navigation function can be defined as:

$$\gamma(q) = w\pi_h(q) + \|q - q_{hi}\|^2,$$

in which the h subscripts are derived from B_h, the highest-priority neighborhood that contains q. The $w\pi_h$ term is added to ensure that each time a higher-priority ball is reached, the navigation function will decrease.

The navigation function as defined might seem to cause difficulties due to discontinuities. This is also permitted in the deployment model of [10]. This is acceptable is that every nonempty neighborhood intersection contains an open set, within which switching between controllers can occur gracefully. This is the approach taken in many hybrid systems (see, for example, [16]). If such discontinuities are still a problem, then interpolation schemes could be applied over the open set. Such issues are interesting, but beyond the scope of our algorithmic framework.

We provide the following proposition based on the computation steps given above:

Proposition 1: An SNG with an underlying graph, $G(V, E)$, can compute a navigation function for a specified goal in time $O(|V| + |E|)$.

Proof: Assuming that point location within a single neighborhood can be performed in $O(1)$ time, Step 1 takes $O(|V|)$ time to naively locate a neighborhood that contains q (this can be dramatically improved for many neighborhood systems). Applying breadth-first or depth-first search to G in Step 2 requires $O(|V|+|E|)$ time. Once the cost-to-go values have been assigned, the priorities can be assigned in $O(|V|)$ time in Step 3. Step 4 requires $O(|V| + |E|)$ time to determine all highest-priority neighbors in G. It is assumed that q_{vi} can be identified from a neighborhood-pair intersection that can be computed in $O(1)$ time. Step 5 is actually computed during execution because it depends on q. Thus, once the first four steps have been completed, the global navigation function has been defined for any q. Combining the total running time from all of the steps yields $O(|V| + |E|)$ time to compute a navigation function. ■

We make one final remark about the computation of navigation functions. Suppose that the goal changes continuously over time. In most cases, the same neighborhood, B_g, will contain the goal. When this occurs, only the local navigation function for B_g needs to be changed, to ensure convergence to the new goal. This can be performed in constant time, which could be convenient in applications such as tracking a moving target.

C. The Execution Phase

We finally consider the execution phase. Here is it assumed that an SNG has been given with a navigation function computed over it. The goal of the first two phases was to precompute data structures that expedite the execution phase as much as possible. Therefore, the execution phase is straightforward. Assume that discrete-time control is used for the robot, and that at any time increment, it measures the configuration. Initially, the highest-priority neighborhood that contains the initial configuration must be found. This takes time $O(|V|)$ using a naive algorithm, but can be significantly improved for most neighborhood systems of interest by using point location data structures.

In each time increment, the highest-priority neighborhood that contains the current configuration must be determined. It is assumed that the configuration changes only by a small amount. Therefore, most of the time, the highest-priority neighborhood will not change. Local graph computations can be performed to determine the collection of neighborhoods that need to be checked for priority. In our experiments, this takes a few microseconds on a standard PC, even for high-dimensional configuration spaces. The advantages are similar to those of "almost constant time," performance for incremental distance collision checking due to *temporal coherency* [35], [44].

Although one could easily imagine rapidly iterating along a path obtained from a standard path planning algorithm, replanning would have to be performed if the configuration drifts unpredictably from the path. This almost always happens in practice, which causes the need for path clearance assumptions, tracking assumptions, etc. In the case of the SNG, motion commands can continue to be applied to the robot regardless of the configuration that it enters, as long as it stays within $\mathcal{B}$. In this sense, it represents a configuration-feedback motion strategy.

We provide a simple example to help understand the philosophy. In our experiments we have considered the simple transition equation

$$q_{k+1} = q_k + u_k + \theta_k, \qquad (1)$$

in which $dim(u_k) = dim(\theta_k) = n$, and u_k is a directional heading input. The θ_k term represents a small disturbance parameter that is bounded, but unpredictable. At each iteration, u_k is chosen in the direction that most decreases the navigation function. If the disturbances are chosen as simulations of a random process, each execution will yield different paths; however, the same navigation function is used in every case. Obviously, our simple formulation here does not take into account differential constraints; however, such consideration is possible as long as a Lyapunov or other return functions can be defined over each neighborhood. It is beyond the scope of this paper to address the control of particular systems.

V. SNG Construction Algorithm

This section contains two parts. Section V-A presents the general components of the algorithm, and Section V-B presents and analyzes the probabilistic termination condition, which is applied in the general algorithm.

A. Algorithm Overview

Recall from Section IV-A that the SNG construction algorithm can be defined using any sample sequence that is dense in $\mathcal{C}_{free}$. In this section, we assume that uniform random sampling is used, which leads to a probabilistic termination criterion. Figure 3 gives an outline of the algorithm, which has two inputs, $\alpha \in (0, 1)$ and $P_c \in (0, 1)$. This enables the user to prescribe the quality of the $\mathcal{C}_{free}$ approximation. The algorithm will terminate when the probability is at least P_c that 100α percent of $\mathcal{C}_{free}$ has been covered. In other words, $\mu(\mathcal{B})/\mu(\mathcal{C}_{free}) \geq \alpha$ with probability at least P_c, in which μ denotes Lebesgue measure in $\mathcal{C}_{free}$. Alternatively, a deterministic, dense sequence could be used to obtain deterministic guarantees of coverage; however, this case is not considered here.

```
GENERATE_SNG($\alpha$,$P_c$)
1   G.init($q_{init}$);
2   while (TerminationUnsatisfied(G,$\alpha$,$P_c$) do
3       repeat
4           $q_{new}$ ← RandomConf();
5           $d$ ← DistanceComputation($q_{new}$);
6       until (($d > 0$) and ($q_{new} \notin \mathcal{B}$))
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
```

Fig. 3. This algorithm constructs the SNG using uniform random sampling, and determines automatically when to terminate based on estimated coverage of $\mathcal{C}_{free}$.

Each execution of Lines 3-9 corresponds to the addition of a new neighborhood, $B_{v_{new}}$, to the SNG. This results in a new vertex in V, and new edges that each corresponds to a neighborhood that intersects $B_{v_{new}}$. Neighborhoods are added to the SNG until the probabilistic termination condition is met, causing TerminationUnsatisfied to return FALSE. The **repeat** loop from Lines 3 to 6 generates a new sample in $\mathcal{C}_{free} \setminus \mathcal{B}$, and 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 $\mathcal{O}$ [17], [18], [35], [44], [47],

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

If d is not positive, then q_{new} is in collision, and another configuration is chosen. Also, q_{new} must lie outside of $\mathcal{B}$ before the *repeat* loop terminates. This forces the SNG to quickly expand into $\mathcal{C}_{free}$, and leads to fewer edges per vertex in V. Finally, Lines 10 and 11 perform some simple cleaning to remove neighborhoods strictly contained within other neighborhoods, and to remove neighborhoods that intersect no others.

There are three costly parts to the algorithm at each iteration: 1) the distance computation in Line 5, 2) testing whether q_{new} lies in $\mathcal{B}$ in Line 6, and 3) determining the new edges in Line 9. The distance computation cost depends, of course, on the complexity of the robot and obstacles. Distance computation algorithms are very efficient in practice, and their existence is essential to our approach. In our implementation, we reply on the PQP package from the University of North Carolina. The neighborhood size is computed in Line 7 using this distance information; this is presented in Section VI.

The other two expensive parts of the algorithm can be greatly accelerated by exploiting efficient algorithms from computational geometry. Theoretically, it is possible to perform these operations in logarithmic expected time in terms of the number of neighborhoods, regardless of dimension. The random geometric separators technique [43] can provide $O(\lg n)$ expected running time to locate a new configuration in the SNG, and $O(n \lg n)$ expected running time to make all edges. While many theoretical algorithms have superior complexity, an implementation is often impractical or inefficient for the range of problems of interest. For this reason, we have adapted the Approximate Nearest Neighbors package from the University of Maryland to our application [45], [2], even though the theoretical complexity is somewhat higher (it is still much improved over naive computations), and nearest-neighbor searching is not precisely the problem of interest. The algorithms are based on Kd-trees and related data structures. The work in [2] extended the algorithms and package to work for topological spaces that usually arise in motion planning.

The acceleration of Lines 6 and 9 using nearest neighbor techniques proceeds as follows. For determining whether $q_{new} \in \mathcal{B}$, nearest neighbor searching is performed on the neighborhood centers. A prescreening using k neighborhoods that have the nearest centers to q_{new} is performed. In practice, we have selected $k = 30$. If q_{new} lies in any of these neighborhoods, it is quickly rejected. Otherwise, there are two different ways to implement the algorithm. At this point, q_{new} can be checked for containment among all neighborhoods. This will not be too costly because as the algorithm runs for a while, most of the time is wasted evaluating new configurations that lie in $\mathcal{B}$. The nearest neighbor technique eliminates most of this cost. The other way to implement the algorithm is simply to allow q_{new} to be added if it is not contained in one of the nearest neighbors. In practice, this leads to a little inefficiency by introducing a small number of neighborhoods that are not strictly needed. For constructing edges, we only attempt to connect the new neighborhood to the neighborhoods that correspond to the k nearest centers (an intersection test must also be performed with each one). It might be possible that some other neighborhoods intersect the new neighborhood; however, this does not cause any practical difficulty. Simple connected component analysis can be to ensure that the connectivity is being preserved. In fact, it might make sense to limit the total number of edges per vertex, as long as $\mathcal{C}_{free}$ topology is preserved. Fewer edges will lead to solutions that are further from optimal, but this might not be an important concern in many applications.

B. Termination Condition

In this section, the termination condition from Line 2 of the algorithm in Figure 3 is derived. The ability of the algorithm to estimate the quality of the coverage is one of the advantages of our approach. A similar condition appears in the Visibility PRM approach [55]. The basic idea behind our termination condition is to derive estimates of $\mu(\mathcal{B})/\mu(\mathcal{C}_{free})$ by analyzing the statistics gathered from attempts to find a new configuration at random which lies in $\mathcal{C}_{free} \setminus \mathcal{B}$. As $\mu(\mathcal{B})/\mu(\mathcal{C}_{free})$ becomes closer to one, we expect the number of failed attempts to find a new configuration to increase. Our analysis concludes with a condition that uses the inputs, α and P_c, of the algorithm to assert that 100α percent of the volume of $\mathcal{C}_{free}$ has been covered with probability P_c.

Let Y be a random variable corresponding to a new configuration in an iteration of the SNG construction algorithm. The experiment of Y has two possible outcomes: either $q_{new} \in \mathcal{B}$, which is a "failure", or $q_{new} \in \mathcal{C}_{free} \setminus \mathcal{B}$, which is a "success". Let $Y = 0$ denote failure, and let $Y = 1$ denote success. Since each q_{new} is chosen uniformly at random from $\mathcal{C}_{free}$, Y has a Bernoulli distribution,

$$P[Y = 0] = \theta \ \ and \ \ P[Y = 1] = 1 - \theta,$$

in which $\theta = \mu(\mathcal{B})/\mu(\mathcal{C}_{free})$.

The following lemma shows that the trials of Y can be used to estimate θ. Let $Y_1, Y_2, \ldots, Y_m$ represent a sequence of independent trails of Y, and let $\bar{Y}$ be the sample mean. Note that $1 - \bar{Y}$ is an unbiased estimator of θ.

Lemma 1: For given fraction α and $T \in (0, 1)$, the probability, P_c, that $1-\bar{Y} \geq T$ implies $\theta \geq \alpha$ is

$$P_c = 1 - \alpha^{m+1} \sum_{i=0}^{\lfloor m(1-T) \rfloor} \binom{m}{i} \left(\frac{1}{\alpha^i (m-i+1)} - \frac{1}{m+1} \right). \tag{2}$$

Let P_c be called the *confidence level.*

Proof: There are two kinds of hypothesis errors: rejection of the null hypothesis if it is true is called a *Type I* error; acceptance of the null hypothesis if it is false is called a *Type II* error. More precisely, $1-\bar{Y} < T$ while $\theta \geq \alpha$ causes a Type I error, and $1-\bar{Y} \leq T$ while $\theta < \alpha$ causes a Type II error. Let P_I and P_{II} represent the probabilities that Type I and Type II errors occur, respectively. Using this notation, $P_I = P\left[1-\bar{Y} < T\ ;\ \theta \geq \alpha\right]$ and $P_{II} = P\left[1-\bar{Y} \geq T\ ;\ \theta < \alpha\right]$. Note that,

$$P_c = 1 - P_{II}. \tag{3}$$

Let $y_1, y_2, \ldots, y_m$ represent the outcomes of a sequence of experiments of Y. Let X be a random variable that denotes the number of trials of Y at which the k successes occur, i.e.,

$$\sum_{i=1}^{m} y_i = k.$$

Notice that X has a binomial distribution,

$$P[X=k] = \binom{m}{k} \theta^{m-k} (1-\theta)^k.$$

Thus, P_{II} can be written as

$$\begin{aligned} P_{II} &= P\left[(1-\bar{Y}) \geq T\ ;\ \theta < \alpha\right] \\ &= P\left[X \leq \lfloor m(1-T) \rfloor;\ \ \theta < \alpha\right] \\ &= \int_0^{\alpha} P\left[X \leq \lfloor m(1-T) \rfloor\right]\ d\theta \\ &= \int_0^{\alpha} \sum_{i=0}^{\lfloor m(1-T) \rfloor} \binom{m}{i} \theta^{m-i} (1-\theta)^i\ d\theta \\ &= \alpha^{m+1} \sum_{i=0}^{\lfloor m(1-T) \rfloor} \binom{m}{i} \left(\frac{1}{\alpha^i (m-i+1)} - \frac{1}{m+1} \right). \end{aligned}$$

Therefore, we have,

$$P_c = 1 - \alpha^{m+1} \sum_{i=0}^{\lfloor m(1-T) \rfloor} \binom{m}{i} \left(\frac{1}{\alpha^i (m-i+1)} - \frac{1}{m+1} \right),$$

which completes the proof. ■

Equation 2 implies that P_c increases as m increases. For given fraction α and P_c, Lemma 1 provides a method to estimate whether the fraction of $\mathcal{B}$ over $\mathcal{C}_{free}$ is at least α with confidence level at least P_c. However, it is not the only method to do so.

The following lemma presents another way to estimate θ. Once again, consider random trials $Y_1, Y_2, \ldots, Y_m$. Assume the first success occurs at Y_m.

Lemma 2: For given fraction α and $M > 0$, the probability, P_c, that $m \geq M$ implies $\theta \geq \alpha$ is

$$P_c = 1 - \frac{\alpha^M}{M}. \tag{4}$$

Proof: Borrowing from the proof of Lemma 1, we start with (3). Let $y_1, y_2, \ldots, y_m$ represent the outcomes of a sequence of experiments of Y. Let X be a random variable that denotes the number of trials of Y at which the first success occurs at y_m,

$$y_i = 0\ ,\ i = 1, \cdots, m-1\ , y_m = 1.$$

Note that X has a geometric distribution,

$$P[X=m] = \theta^{m-1}(1-\theta).$$

Thus, P_{II} can be written as:

$$\begin{aligned} P_{II} &= P\left[X \geq M\ ;\ \theta < \alpha\right] \\ &= \int_0^{\alpha} P\left[X \geq M\right]\ d\theta \\ &= \int_0^{\alpha} \left(1 - P\left[X < M\right]\right)\ d\theta \\ &= \int_0^{\alpha} \left(1 - \sum_{i=1}^{M-1} \theta^{i-1}(1-\theta)\right)\ d\theta \\ &= \frac{\alpha^M}{M} \end{aligned}$$

Therefore, we have,

$$P_c = 1 - \frac{\alpha^M}{M},$$

which establishes the lemma. ■

Each of Lemma 1 and Lemma 2 provides a method to estimate θ, based on the different statistical hypotheses. Two termination conditions can be derived from them, respectively:

- h_1: terminate based on m trials of Y, at which k successes have been counted.
- h_2: terminate based on m consecutive failures followed by the first success.

The following proposition shows how h_1 and h_2 can be used to make the termination decision.

Proposition 2: For given a fraction α and P_c, the probability of the fraction $\theta \geq \alpha$ is at least P_c, i.e.,

$$P[\theta \geq \alpha] \geq P_c,$$

if the termination condition satisfies either one of the following:

- h_1:

$$P_c \geq 1 - \alpha^{m+1} \sum_{i=0}^{k} \binom{m}{i} \left(\frac{1}{\alpha^i(m-i+1)} - \frac{1}{m+1} \right) \quad (5)$$

- h_2:

$$m \geq \frac{\ln(1-P_c)}{\ln \alpha} - 1 \quad (6)$$

Proof: For condition h_1, let the confidence level be at least P_c as shown in (2). Substitute $\lfloor m(1-T) \rfloor$ with the expected number of success k to immediately obtain (5).

For condition h_2, we use (4) to obtain

$$P_c = 1 - \frac{\alpha^M}{M} \geq 1 - \alpha^M. \quad (7)$$

Equation 7 yields a lower bound on P_c. Letting the confidence level be at least P_c will yield a lower bound on m as provided in (6). ■

Although both h_1 and h_2 are feasible termination conditions in practice, they have different performance, as indicated by the following proposition.

Proposition 3: For the same $\bar{Y}$, the termination condition h_1 has a better or equal confidence level than h_2.
Proof: For the same $\bar{Y}$, the size of the sample sequences differ, $m_{h_1} \geq m_{h_2}$. Therefore, according to (2) and (4), it follows that $P_c^{h_1} \geq P_c^{h_2}$. ■

According to Proposition 3, it seems h_1 is always better than h_2. However, this is not the whole story. To achieve the same $\bar{Y}$, h_1 needs more samples than h_2. This yields more computation time because in each trial, we have to check for collision and check whether the new configuration is inside $\mathcal{B}$, which are expensive. The computation time increase of h_1 may be significant. A better strategy is to terminate the algorithm if either h_1 is satisfied or h_2 is satisfied.

While the construction algorithm iterates, we open a sample window with a fixed window size m in the sample process. Inside the window, k successes are observed. In each iteration of adding a new configuration, we advance the sample window to include the new experiment and apply the termination condition h_1 on it. This process continues until a success is found. We count the number of successive failures before this success and apply the termination condition h_2 on it. The algorithm terminates if either one of h_1 and h_2 is satisfied. In general, if a larger m is chosen, then a more accurate decision has been made, but more computation time is needed. Note that inside the sample window, θ is not fixed. Each time a new configuration is found, a new neighborhood is added into $\mathcal{B}$, which increases θ. This means that the termination conditions are somewhat conservative, but if α and P_c are close to 1, we expect this effect to be negligible.

VI. Neighborhood Selection

Up to now, the SNG has been presented in terms of generic neighborhoods. This section addresses the critical issue of how to construct neighborhoods from information provided by a distance computation algorithm. The goal of selecting a new neighborhood is to have it enclose as much of $\mathcal{C}_{free}$ as possible, while keeping its geometry as simple as possible. Generally it is hard to satisfy both of those two conditions, which leads to tradeoffs. In this section, we will focus on the simplest neighborhood shape, an n-dimensional ball, and discuss how to determine the radius of a ball in $\mathcal{C}_{free}$ based on the distance information obtained from collision detection in $\mathcal{W}$. We will also address using the union of n-dimensional balls and n-dimensional axis-aligned cylinders, which is often more efficient than just using balls.

A. Guaranteed Collision-Free Balls

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

$$B_{new} = \{q \in \mathcal{C}_{free} \mid \|q_{new} - q\| \leq r\},$$

is a subset of $\mathcal{C}_{free}$. The definition of B_{new} assumes that $\mathcal{C}$ is an n-dimensional Euclidean space; however, minor notational modifications can be made to include other frequently-occurring topologies, such as $\mathbb{R}^2 \times S^1$ and $\mathbb{R}^3 \times P^3$. The following proposition indicates how to select r based on distance information in $\mathcal{W}$.

Proposition 4: Given a robot, $\mathcal{A}$, for any two configurations q and q', and any real number $d > 0$, there exists a positive number r such that

$$\|q - q'\| \leq r \Rightarrow \max_{a \in \mathcal{A}} \|a(q) - a(q')\| < d, \quad (8)$$

in which $a(q)$ refers to the position in $\mathcal{W}$ of a point $a \in \mathcal{A}$, transformed to configuration q.
Proof: Let $f : \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}^m$ denote the expression of the kinematics of $a \in \mathcal{A}$, which is assumed to be a smooth mapping from an n-dimensional configuration space to an m-dimensional world (see Figure 4). Let $\tau(a, q, q', t)$ be the following curve that sends $f(a, q)$ to $f(a, q')$:

$$\tau(a, q, q', t) = f(a, q + t(q' - q)),$$

for $t \in [0, 1]$. Using this definition,

$$\begin{aligned}
\|\Delta f\| &\equiv \|f(a,q) - f(a,q')\| \\
&= \|\tau(a,q,q',0) - \tau(a,q,q',1)\| \\
&= \left\| \int_0^1 \frac{\partial \tau(a,q,q',t)}{\partial t} dt \right\| \\
&\leq \int_0^1 \left\| \frac{\partial \tau(a,q,q',t)}{\partial t} \right\| dt \\
&\leq \int_0^1 \left\| \frac{\partial f(a, q + t(q'-q))}{\partial q} \right\| \|q' - q\| \, dt
\end{aligned}$$

in which $\frac{\partial f(a,q)}{\partial q}$ is a $m \times n$ matrix that represents the Jacobian of the kinematics in terms of a particular point on the robot. One can determine a point, $a_f \in \mathcal{A}$ that maximizes $\frac{\partial f(a,q)}{\partial q}$ over all $a \in \mathcal{A}$. In other words, a_f moves most

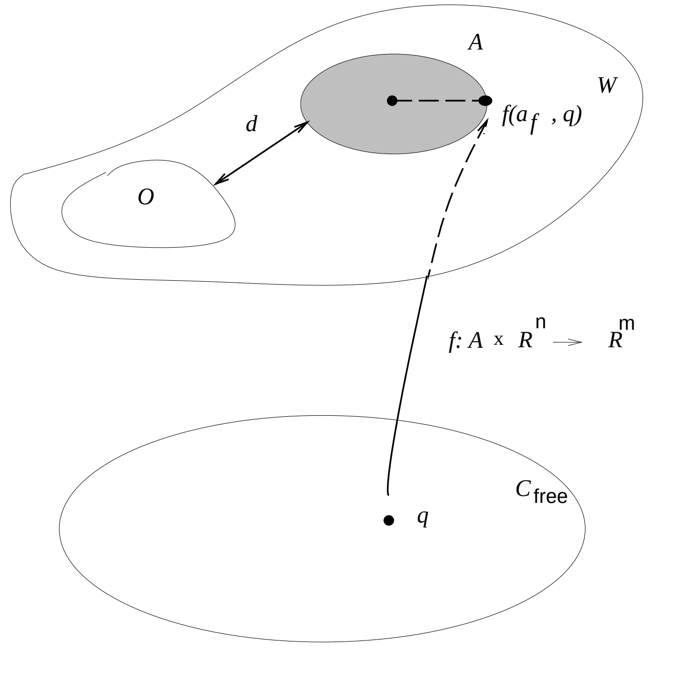


Fig. 4. The effect on distance of the kinematic mapping from an n-dimensional configuration space to an m-dimensional world.

rapidly as the configuration varies along the path from q to q'. Let

$$D = \max_{t\in[0,1]} \left\| \frac{\partial f(a_f(t), q + t(q' - q))}{\partial q} \right\|.$$

We will use a_f to denote the $a_f(t)$ that achieves the maximum above. Note that $\|\frac{\partial f}{\partial q}(a, q)\|$ is uniformly upper bounded by D over $\mathcal{A} \times \mathcal{C}$:

$$\left\| \frac{\partial f(a, q)}{\partial q} \right\| \le D \;\; \forall a \in \mathcal{A},\ \forall q \in \mathcal{C}.$$

Thus, for any $a \in \mathcal{A}$,

$$\|f(a, q') - f(a, q)\| \le D\, \|q' - q\|,$$

which leads directly to (8) by choosing $r = \frac{d}{D}$. ■

Proposition 4 implies that, if Line 5 of the algorithm in Figure 3 returns d, a ball,

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

can be constructed with assurance that $B_v \subset \mathcal{C}_{free}$, and $r = d/D$. Since the radius is inversely proportional to D, the maximum Jacobian magnitude, it appears desirable to have D small. We next consider nonstandard kinematics formulations that attempt to minimize the adverse effects of D.

B. Benefits of Nonstandard Kinematics Parameterizations

The size of the collision free ball depends greatly on the particular parameterization of the kinematics. To understand the issue, suppose a_f is far from the origin. In this case, small rotations induce large displacements of a_f. Even if the robot is far from obstacles, a small ball will have to be used because a small amount of change in rotation could cause collision. One could use an ellipsoid instead of a ball to solve the problem, but then the SNG could have to incorporate ellipsoids of varying eccentricities, which would be more expensive in other parts of the algorithm.

Luckily, there is a way out of this problem by carefully formulating the kinematics. It turns out that there exist nonstandard parameterizations that lead to improved performance. We present a general method for reformulating kinematic equations to improve the ball size. The method is critical to the success of the SNG, although other methods may be possible.

We consider expressions for arc length of the path traversed by a point, $a \in \mathcal{A}$, in the world, $\mathcal{W}$, as the configuration is changed from q to some q'. In general, differential arc length in $\mathcal{W}$, based on differential changes in configuration, is specified by the *metric tensor*,

$$ds^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} g_{ij}\, dq_i\, dq_j,$$

in which

$$g_{ij} = \sum_{k=1}^{m} \frac{\partial f_k}{\partial q_i} \frac{\partial f_k}{\partial q_j}.$$

In the expressions above, the q_i is the i^{th} component of q (which corresponds to an abuse of notation with respect to other subscripts applied to q). The total arc length is given by $\int ds$, which can be evaluated by a suitably-parameterized path integral. Consider a pair, q, q', of configurations such that $\|q - q'\| = d$, for some positive constant, d. Consider using a linear interpolation function, τ, that connects q to q', as defined in Proposition 4.

Suppose that the g_{ij} are constant, and that M denotes an $n \times n$ symmetric positive definite matrix of constant metric tensor components. In this case, the arc length for the path τ is given by the quadratic form $(q-q')^T M (q-q')$. This yields the following bound

$$\|f(a, q') - f(a, q)\|^2 \le (q' - q)^T M (q' - q) = d^2. \qquad (9)$$

The left side above indicates the amount of displacement in $\mathcal{W}$ of a, which is certainly bounded by the total arc length of the path traversed by a. The right side represents the equation of an ellipsoid whose eccentricity is given by differences between eigenvalues of M.

Suppose now that d is the value given by the distance computation algorithm at some particular q. In trying to place a ball around q, we would like any configuration, q', of distance d from q to be collision-free. This will be assured if (9) is satisfied. Since we are only allowed to use spheres, instead of ellipsoids, we are forced to use the largest ball contained in the ellipsoid. This means that the largest eigenvalue of M will dominate. To fix this problem, we reparameterize the kinematics so that all eigenvalues of M are of equal value.

Our approach is based on an assumption that the g_{ij} are constant. This method can also be used when the g_{ij} are not constant, but poorer performance is obtained. In this case, we utilize worst-case bounds over all of $\mathcal{C}$ to treat the metric tensor as a constant.

Rigid Robots. To illustrate the approach, consider a 2D rigid robot with translation and rotation, leading to $\mathcal{C} = \mathbb{R}^2 \times S^1$, and a_f is selected as the point in $\mathcal{A}$ that is furthest from the origin, O (it would have maximum magnitude in polar coordinates). Suppose rotation is parameterized from 0 to 2π; the effects of rotation would dominate the metric tensor if the translation coordinates have a large range, such as 0 to 1000. Let r_m be the Euclidean distance from a_f to O. If a scaled rotation, $q_3 = r_m\theta$, is used, then (9) will represent the equation of a sphere. Although the fraction of S^1 that is covered is the same in either case, the amount of $\mathbb{R}^2$ that is covered is increased substantially. This subtle relationship between the parameterization and the size of spheres is quite important to the success of the SNG, and therefore is discussed in detail in the remainder of this section for different kinds of robots.

As mentioned above, for 2D rigid robots we use scaled rotation, $q_3 = r_m\theta$, instead of standard parameterization of rotation θ. The following homogeneous transformation can be used for the kinematics

$$\begin{pmatrix} x_{new} \\ y_{new} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(q_3/r_m) & -\sin(q_3/r_m) & q_1 \\ \sin(q_3/r_m) & \cos(q_3/r_m) & q_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{10}$$

in which q_1 is x-translation, q_2 is y-translation, and q_3 is scaled rotation. Equation 10 provides an easy way to calculate the arc length, if a_f is fixed and known in advance. Consider substituting (10) into (9). In this case, M simplifies to the identity matrix, and (9) yields the equation of a sphere with radius d. Therefore, we have a 3-dimensional ball $B_v \subset \mathcal{C}_{free}$ such that

$$B_v = \{q' \in \mathcal{C}_{free} \mid \|q' - q\| \leq d\},$$

For a 3D rigid robot with translation and rotation, $\mathcal{C} = \mathbb{R}^3 \times S^1 \times S^1 \times S^1$, the same result can be obtained if roll, pitch, and yaw are used to represent rotation. The reason for not using quaternions is to keep the metric tensor constant. Once again, scaled rotation is used. Let q_1, q_2, and q_3 be x, y, and z translations, respectively. Let $q_4 = r_{m\gamma}\gamma$, $q_5 = r_{m\beta}\beta$, and $q_6 = r_{m\alpha}\alpha$ be scaled rotations corresponding to roll (γ), pitch (β), and yaw (α), respectively. Let $r_{m\gamma}, r_{m\beta}, r_{m\alpha}$ be the maximum radii, if we rotate $\mathcal{A}$ around the axis X (roll), Y (pitch), and Z (yaw), respectively. Thus, a 6-dimensional ball, $B_v \subset \mathcal{C}_{free}$, is generated with radius $r = d$, centered at q.

Articulated Robots. For problems that involve articulated bodies, similar expressions can be derived based on the distance from the robot to the obstacles in the world. For example, suppose we have a 2D articulated body which has configuration space $\mathcal{C} = \mathbb{R}^2 \times S^1 \times S^1 \times S^1$ (see Figure 5). As the configuration varies, a_f is fixed on the last body of the chain, as shown in Figure 5. Given q, as θ_1 varies, the furthest distance the a_f can travel is by following a circle

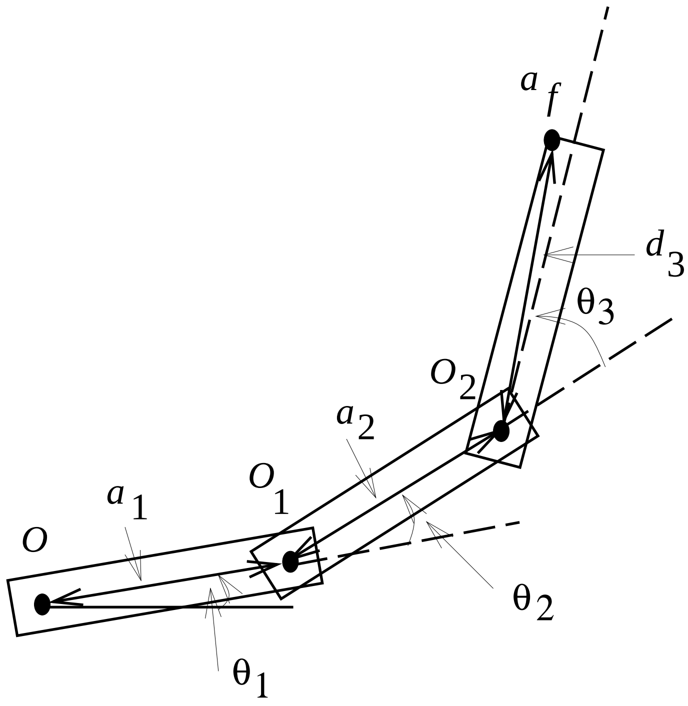


Fig. 5. A 2D articulated chain of bodies with $\mathcal{C} = \mathbb{R}^2 \times S^1 \times S^1 \times S^1$. In this case, a_f is a fixed point on the last body.

that is centered at O with radius $a_1 + a_2 + d_3$. Similarly, as θ_2 varies, the furthest distance a_f can move is by following the circle that is centered at O_1 with radius $a_2 + d_3$. As θ_3 varies, the furthest distance a_f can move is following the circle that is centered at O_2 with radius d_3. Thus, if we use the scaled rotation again, the furthest distance that a_f can move is bounded by

$$\sum_{i=1}^{5} (q_i' - q_i)^2 \leq \|f(a_{max}, q') - f(a_{max}, q)\|^2 < d^2, \tag{11}$$

in which q_1 is x-translation, q_2 is y-translation, and $q_3 = \theta_1(a_1 + a_2 + d_3)$, $q_4 = \theta_2(a_2 + d_3)$, and $q_5 = \theta_3 d_3$ are scaled rotations. This yields a 5-dimensional ball in $\mathcal{C}_{free}$ that is centered at q, and has radius d.

C. Using Unions of Balls and Cylinders

Performance will be much better if cylinders can be used, where appropriate, in addition to balls. The reason is that when the robot is far away enough from the obstacles, it might be able to rotate freely with respect to a fixed axis without colliding. In such a case, instead of using a ball, we should place a cylinder along this axis, which will significantly increase the efficiency of covering $\mathcal{C}_{free}$. Thus, each free variable in q can be left unconstrained. The remaining variables will be constrained with a sphere equation to obtain solid, axis-aligned cylinders. The cross section of each cylinder will be a k dimensional ball, if there are $n - k$ free parameters.

Consider a 2D rigid robot with $\mathcal{C} = \mathbb{R}^2 \times S^1$. For a given q, assume $d > r_m$; in this case, the robot cannot collide no matter how it is rotated. Thus, we can place a

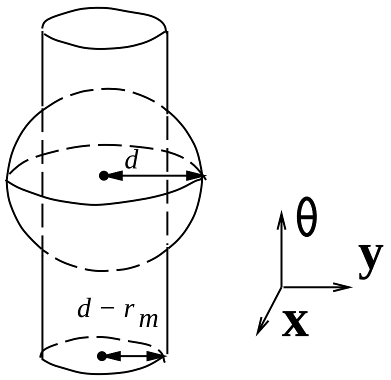


Fig. 6. Using unions of cylinders and balls in the case of a 2D rigid robot with $\mathcal{C} = \mathbb{R}^2 \times S^1$.

cylinder centered at q_1, q_2 with radius $d - r_m$ along the axis $q_3 = \theta r_m$ (see Figure 6). We can also place a ball centered at q with radius d. Therefore, the new neighborhood B_v has the form

$$B_v = B_v^c \cup B_v^b \ ,$$

in which B_v^c is a cylinder,

$$B_v^c = \{q' \in \mathcal{C}_{free} \mid (q'_1 - q_1)^2 + (q'_2 - q_2)^2 \leq (d - r_m)^2\},$$

along the q_3 axis, and B_v^b is a ball,

$$B_v^b = \{q' \in \mathcal{C}_{free} \mid \|q' - q\| \leq d\}.$$

In the case of a 3D rigid robot with $\mathcal{C} = \mathbb{R}^3 \times S^1 \times S^1 \times S^1$ (assuming yaw, pitch, and roll are used), a similar result is obtained. For example, suppose $d > r_{m\gamma}$ and $d > r_{m\beta}$. Since $d > r_{m\gamma}$, the robot cannot collide while rotating about the X-axis. Likewise, $d > r_{m\beta}$, the robot cannot collide while rotating about the Y-axis. Thus, two cylinders centered at q, can be placed along axes $q_4 = r_{m\gamma}\gamma$ and $q_5 = r_{m\beta}\beta$, with radii $d - r_{m\gamma}$ and $d - r_{m\beta}$, respectively. The neighborhood is $B_v = B_v^{cX} \cup B_v^{cY} \cup B_v^b$, in which B_v^{cX} and B_v^{cY} are cylinders along X and Y axes respectively, and B_v^b is a ball. Using unions of cylinders and balls can significantly reduce the size and construction time of the SNG. We will present some experimental results that show the benefit of using unions of cylinders and balls in Section VIII.

VII. Sampling Enhancement

For many problems, it might be advantageous to apply nonuniform sampling, especially for problems that involve narrow corridors in $\mathcal{C}_{free}$. In the context of sampling-based path planning, one interesting approach to alleviate this difficulty is sampling onto the medial axis [19], [46], [58]. This general idea is particularly well-suited to the SNG because maximizing distance from obstacles will yield larger neighborhoods, which ultimately reduces the size of the SNG.

In this section we briefly present a sampling-based method that attempts to perturb a new configuration, q_{new} from Figure 3, towards the medial axis. The approach is based on two steps:

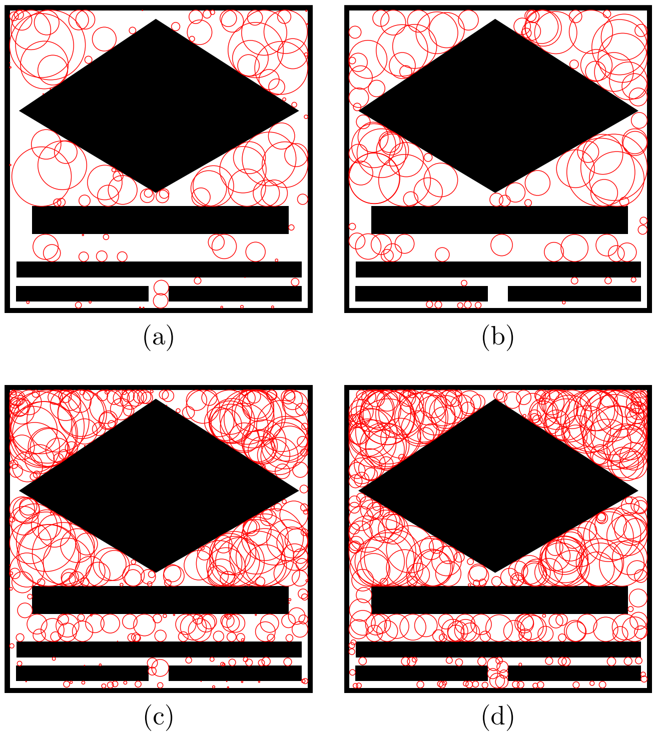


Fig. 7. The comparison of generating the SNG both with and without sampling enhancement in a 2D point robot example: (a), (c) show sampling without enhancement with 100 and 300 nodes, respectively; (b), (d) show sampling with randomized perturbations with 100 and 300 nodes, respectively.

1. The first step is to choose a direction along which q_{new} can be adjusted to rapidly increase d, the distance between the robot and obstacles in $\mathcal{W}$. Rather than performing a costly optimization, we employ a simple, but effective approach. Several direction vectors are generated at random. In each direction, a new sample is generated at a fixed distance from q_{new}. The direction, v, along which the maximum increase in d occurs is saved. An alternative approach is to calculate v by using least-squares algorithms. For example, one can find v by following the gradient of the distance function d, which in turn can be calculated by solving linear equations formed by the projection of the gradient in k selected directions. In our implementation, however, we used the simpler sampling approach and obtained satisfactory results.
2. The second step iteratively computes improved configurations along the direction of v from q_{new}. In each iteration two conditions are checked: the enhanced configuration, q_e, must increase d, and q_e must yield a neighborhood that contains q_{new}. The second condition is requred to ensure the convergence of the planner; it is guaranteed that some amount of new coverage is obtained in each iteration. It also avoids checking whether $q_e \in \mathcal{B}$. In each iteration, several distances along v from the current q_e are attempted by starting with a large distance and repeatedly halving until the conditions for progress are satisfied. If the conditions are not met, even for a very small distance, then the enhancement algorithm is terminated, and q_e is returned.

In combination with the SNG, we only apply the enhancement algorithm to the samples that yield distance values below some threshold. Samples that already yield large neighborhoods are left alone. Figure 7 illustrates the advantage of the enhancement algorithm through a simple 2D point robot example. Figures 7.a and 7.c show sampling without enhancement with 100 and 300 nodes, respectively. Figures 7.b and 7.d show corresponding results with the same number of nodes, but applying enhancement. Most of the tiny balls in Figures 7.a and 7.c have been enlarged by using enhancement. The coverage of $\mathcal{C}_{free}$ has also been increased significantly. After adding 300 nodes to the G, the maximum number of random trials to put a new sample in $\mathcal{C}_{free} \setminus \mathcal{B}$ was 16 using enhancement, but only 7 without (more trials fail as the a larger fraction of the $\mathcal{C}_{free}$ is covered). Since the random perturbation is only an approximation to sampling onto the medial axis, there is no guarantee that every tiny ball will be enlarged. This fact can be found in Figures 7.b and 7.d; there are still a few tiny balls remaining. Nevertheless, are observed substantial performance improvements for problems that involve narrow corridors; more sample enhancement results are presented in Section VIII.

VIII. An Implementation with Examples

We have implemented the three phases of computation described in Section IV: precomputation, building a navigation function, and execution. These were implemented in Gnu C++ on a 500Mhz PC running Linux. A variety of experiments have been performed for robots in 2D and 3D environments with up to six degrees of freedom.

For the first set of examples, no enhancement or nearest neighbor-based acceleration was performed. Figure 1 shows the balls of the SNG for a point robot in a 2D environment. Figure 8.a shows the SNG edges as line segments between ball centers. The SNG construction required 23s, and the algorithm terminated after 500 successive failures (termination condition h_2 is used and $m = 500$) to place a new ball. The SNG 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. Figures 8.b and 8.c show level sets of two different navigation functions that were quickly computed for two different goals (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 guides 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 8.d shows another result for a 2D point robot. In this case, the robot travels into a narrow corridor, and the SNG contains a single connected component.

For higher-dimensional configuration spaces, we only show robot trajectories, even though much more information is contained in the SNG. Figure 9 shows a complicated

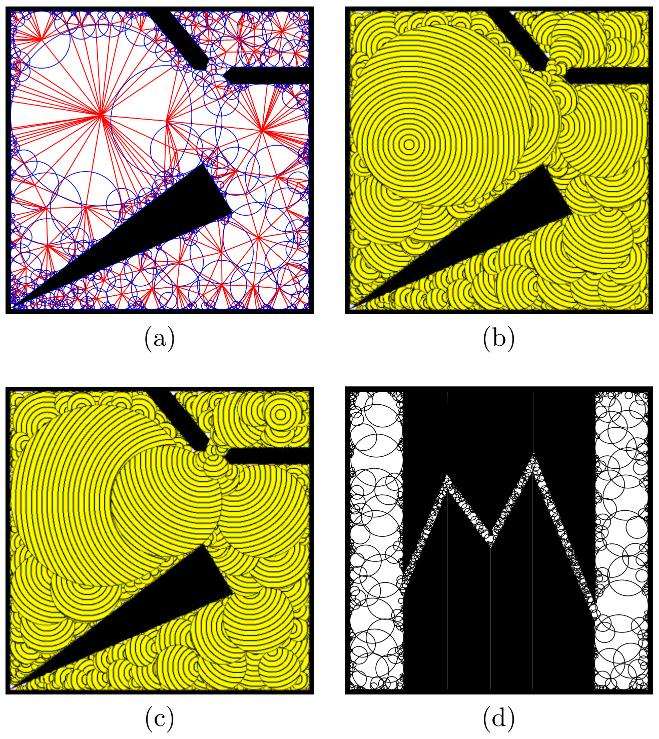


Fig. 8. (a) The SNG for a 2D point robot (b), (c) two navigation functions computed from a single SNG; (d) another example for a 2D point robot.

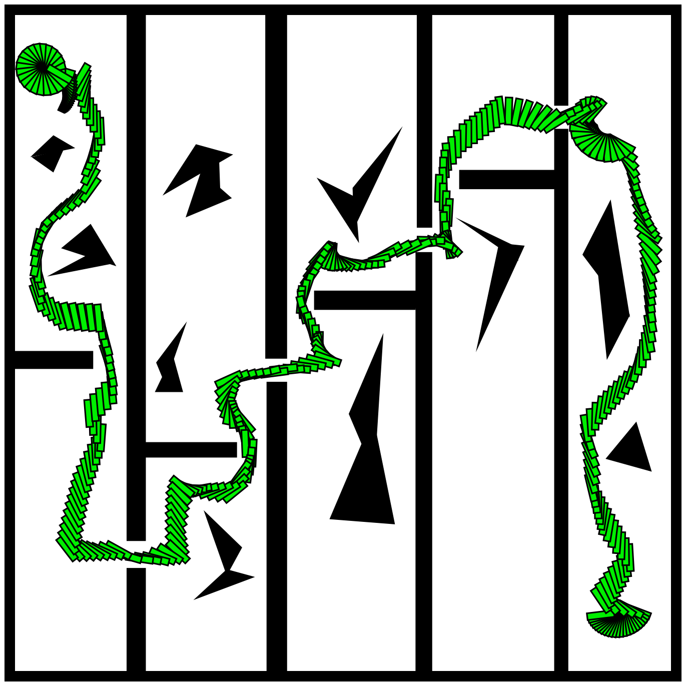


Fig. 9. A path constructed from a 3D SNG, for a rigid robot.

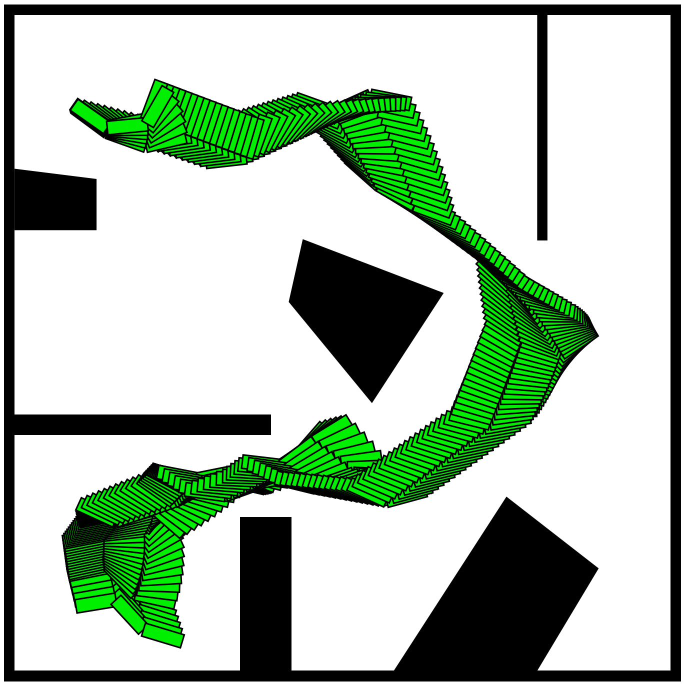


Fig. 10. A path for an articulated robot, constructed from a 5D SNG.

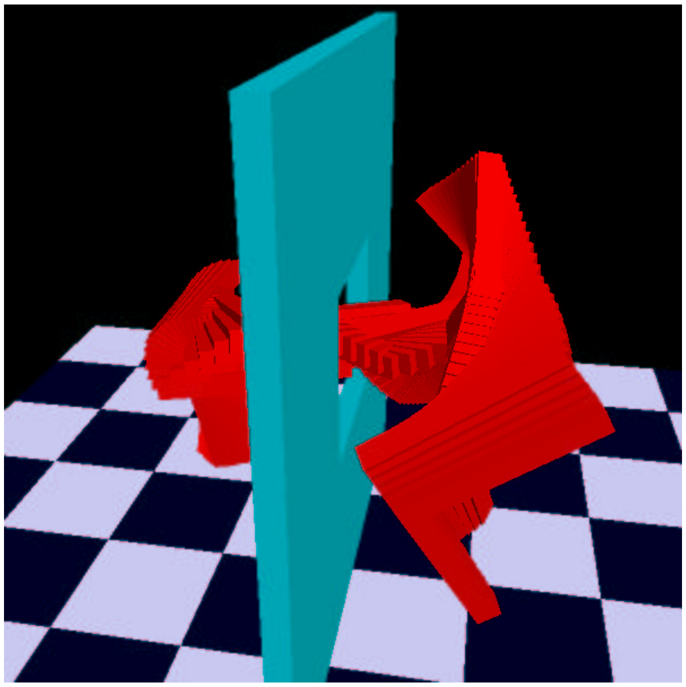


Fig. 11. A path constructed from a 6D SNG for a 3D L-shape robot in 3D environment.

path followed by the robot for an SNG that was computed for a 2D rigid robot that can rotate and translate in a 2D environment. Figure 10 shows paths that were obtained by constructing an SNG in a 5D configuration space for an articulated robot that consists of three free-floating bodies, joined by revolute joints. Figure 11 shows paths that were obtained by following the 6D navigation function that was computed for a 3D L-shape robot that can rotate and translate in a 3D environment. Figure 12 shows a path that moves a long thin feather into a coffee mug, obtained from a navigation function on a 6D SNG. The computation times of constructing SNGs for the examples above are listed in Figure 13.

As expected, smaller α or P_c values result in faster SNG construction time, as shown in Figure 14, in which termination condition h_2 is used. Observe that the size of the SNG and the construction time increase significantly if α or P_c increases. Larger α and P_c imply higher confidence that a larger percentage of C_{free} has been covered, which represents the tradeoff between the requested quality of coverage and the construction time. In other experiments (not shown here), we observed slightly better performance by using both h_1 and h_2 together.

As discussed in Section VI, unions of n-dimensional cylinders and n-dimensional balls significantly improves efficiency. Figure 15 compares the number of neighborhoods needed to achieve the same coverage of $\mathcal{C}_{free}$. Compared to using only balls, up to three times improvement is observed.

The SNG construction is slower when dealing with narrow passages, as mentioned in Section VII. By applying the enhancement algorithm from Section VII, we obtain fewer nodes in the SNG and lower computation times. Figure

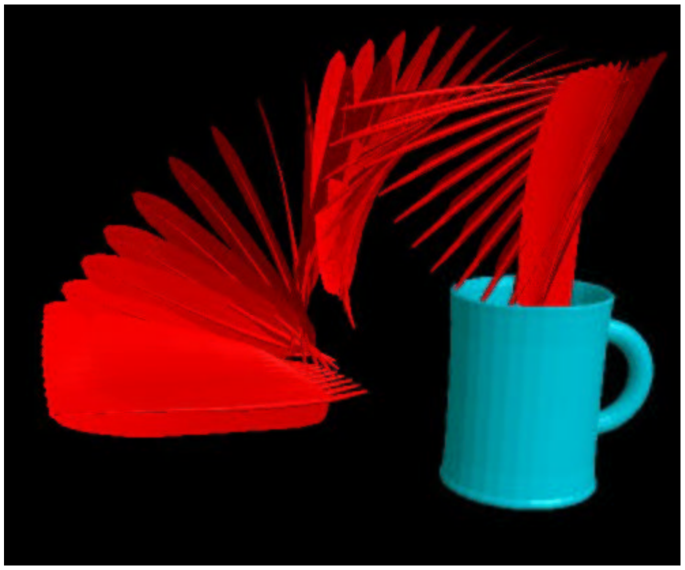


Fig. 12. A path constructed from a 6D SNG: placing a feather into a cup.

16 compares performance with and without sampling enhancement for two examples. Note that the purpose of sampling enhancement is to improve the sampling quality in narrow corridors, not for samples in uncluttered regions. Figure 11 involves a narrow corridor, and Figure 12 is relatively easy. This is why more performance improvement is observed for the problem in Figure 11, as opposed to the problem in Figure 12.

One advantage of the SNG approach is it enables real-time control of the robot without being forced to track a preplanned path. For the simple system in (1), Figure 17 shows the average running time to compute the mo-

	α/P_c	# of Fails	# of Nodes	# of Edges	Construction Time
9	.95/.99	100	20000	78676	87.0s
10	.90/.65	10	12000	74432	25.0s
11	.90/.88	20	83922	1792958	168.8s
12	.90/.99	40	6059	335630	13.2s

Fig. 13. The SNG construction time for various examples. Nearest neighbor-based accelerations were performed, in which 30 neighbors per iteration were checked. No sample enhancement was performed.

α/P_c	# of Fails	# of Nodes	Construction Time
.90/.65	10	247	0.61s
.90/.88	20	6901	21.73s
.90/.99	40	49074	193.69s

Fig. 14. The comparision of using different α and P_c. The experiment is run in a 6D SNG, shown as Figure 12, by using termination condition h_2.

tion command at each step on a Pentium III 500 Mhz PC running Linux. Several different values for Δt are shown, which are proportional to the amount of distance traveled by the robot in each iteration. If the robot travels a smaller distance, then the highest-priority ball is more likely to remain unchanged, resulting in less computation time. We found that over a very large range of Δt values, the execution times remain similar. For 6D configuration spaces, the execution algorithm uses the SNG to compute the next motion command for the robot in several microseconds on a slow PC. Thus, the SNG can yield motion commands at a rate much faster than typical feedback control rates. It is also orders of magnitude faster than what can be achieved by using existing path planning algorithms to replan paths from new configurations.

IX. Conclusions

We have introduced an algorithmic framework for efficiently computing and executing navigation functions on high-dimensional configuration spaces. There are three phases to the computation: 1) precomputation, in which the SNG is constructed for a given environment, 2) the computation of a navigation function for a given goal, and 3) the execution of a feedback motion strategy by using the navigation function. Based on our implementation, we conclude that the method is practical for problems of up to six degrees of freedom. On a standard PC, the SNG can be constructed in seconds, and feedback motion strategies can be executed with a response time of several microseconds.

Several topics remain for further research. It would be interesting to attempt more-challenging problems, either with more degrees of freedom or more environment complexity. Our method is expected to suffer, along with sampling-based path planning methods, from the narrow corridor problem [20]. Although our algorithm converges in volume to a covering of $\mathcal{C}_{free}$, it does not indicate the probability that $\mathcal{B}$ will capture the topology of $\mathcal{C}_{free}$. Although the enhancement algorithm helps to improve the sampling quality, it would be desirable to have better sampling techniques. It might be feasible to develop an SNG construction algorithm that can both add and delete neighborhoods. As better neighborhoods are discovered, poorer ones can be deleted. Finally, deterministic sample sequences should be carefully considered and analyzed in this context.

	α/P_c	Balls Only	Unions of Balls and Cylinders	Improvement factor
11	.90/.88	259807/619.3s	83922/168.8s	3.10/3.67
12	.90/.99	20349/58.9s	6059/13.18s	3.36/4.47

Fig. 15. Improvements are obtained by using unions of cylinders and balls, as opposed to only balls. The left and right sides of "/" list the number of SNG nodes and total construction time, respectively.

	α/P_c	No Enhancement	With Enhancement	Improvement factor
11	.90/.88	83922/168.8s	11073/10.9s	7.56/15.5
12	.90/.99	6059/13.2s	4952/11.1s	1.22/1.19

Fig. 16. Sample enhancement yields significant improvement. The left and right sides of "/" list the number of SNG nodes and the construction time, resectively.

Δt	2D Problem: Fig. 8	6D Problem: Fig. 12
0.001s	1.389μs	2.70μs
0.008s	1.383μs	2.86μs
0.064s	1.380μs	3.98μs
0.128s	1.389μs	5.15μs

Fig. 17. The average running time to compute the motion command using the navigation function stored in the SNG.

Acknowledgments

This work was funded in part by NSF awards 9875304, 0118146, and 0208891. We thank Ahmad Massoud, Sunder Sethuraman, and Yan-Bin Jia for their helpful comments and suggestions.

References

[1] M. D. Adams, H. Hu, and P. J. Roberts. Towards a real-time architecture for obstacle avoidance and path planning in mobile robots. In *IEEE Int. Conf. Robot. & Autom.*, pages 584–589, May 1990.

[2] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 632–637, 2002.

[3] F. Avnaim, J. D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. Technical Report 890, INRIA, 1988.

[4] J. Barraquand and J.-C. Latombe. A Monte-Carlo algorithm for path planning with many degrees of freedom. In *IEEE Int. Conf. Robot. & Autom.*, pages 1712–1717, 1990.

[5] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Trans. Syst., Man, Cybern.*, 19(5):1179–1187, 1989.

[6] M. Branicky, S. M. LaValle, K. Olsen, and L. Yang. Quasi-randomized path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1481–1487, 2001.

[7] O. Brock and L. Kavraki. Decomposition based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *IEEE Int. Conf. Robot. & Autom.*, 2001.

[8] R. W. Brockett. Asymptotic stability and feedback stabilization. In R. W. Brockett, R. S. Millmann, and H. J. Sussmann, editors, *Differential Geometric Control Theory*. Birkháuser, 1983.

[9] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. In *Proc. of the*

8th International Conference on Artificial Intelligence, pages 799–806, 1983.

[10] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behavious. *Int. J. Robot. Res.*, 18(6):534–555, 1999.

[11] W. Choi and J.-C. Latombe. A reactive architecture for planning and executing robot motions with incomplete knowledge. In *Proc. of the International Conference on Intelligent Robots and Systems*, volume 1, pages 24–29, 1991.

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

[13] C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using laplace's equation. In *IEEE Int. Conf. Robot. & Autom.*, pages 2102–2106, May 1990.

[14] M. A. Erdmann. On motion planning with uncertainty. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, August 1984.

[15] B. Faverjon. Object level programmimg of industrial robots. In *Proc. of IEEE Int. Conf. Robot. & Autom.*, pages 1406–1412, 1986.

[16] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance and Control*, 25(1):116–129, 2002.

[17] 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.

[18] 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.

[19] C. Holleman and L. Kavraki. A framework for using the workspace medial axis in PRM planners. In *IEEE Int. Conf. Robot. & Autom.*, 2000.

[20] 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.

[21] J. Ish-Shalom. The cs language concept: A new approach to robot motion design. In *Proc. IEEE Conf. Decision & Control*, pages 760–767, 1984.

[22] 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.

[23] S. Kambhampati and L. S. Davis. Multiresolution path planning for mobile robots. *IEEE Trans. Robot. & Autom.*, RA-2(3):135–145, 1986.

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

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

[26] 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.

[27] O. Khatib. *Commande dynamique dans l'espace opérational des robots manipulateurs en présence d'obstacles.* PhD thesis, Ecole Nationale de la Statistique et de l'Administration Economique, France, 1980.

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

[29] 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.

[30] D. E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *Proc. IEEE Int. Conf. Robot. & Autom.*, pages 1–6, 1987.

[31] D. E. Koditschek. Task encoding: Toward a scientific paradigm for robot planning and control. *J. Robot. Autonomous Sys.*, 9:5–39, 1992.

[32] C. Laugier and F. Germain. An adaptative collision-free trajectory planner. In *Proc. of the International Conference on Advanced Robotics*, 1985.

[33] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, December 2002.

[34] S. M. LaValle and P. Konkimalla. Algorithms for computing numerical optimal feedback motion strategies. *International Journal of Robotics Research*, 20(9):729–752, September 2001.

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

[36] S. R. Lindemann and S. M. LaValle. Incremental low-discrepancy lattice methods for motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, 2003.

[37] T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Trans. Syst., Man, Cybern.*, 11(10):681–698, 1981.

[38] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic systhesis of fine-motion strategies for robots. *Int. J. Robot. Res.*, 3(1):3–24, 1984.

[39] 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.

[40] 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.

[41] M. T. Mason. The mechanics of manipulation. In *Proc. IEEE Int. Conf. Robot. & Autom.*, pages 544–548, 1985.

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

[43] 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.

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

[45] D. Mount and S. Arya. ANN: Library for Approximate Nearest Neighbor Searching. Available at http://www.cs.umd.edu/ mount/ANN/, 1998.

[46] C. Pisula, K. Hoff III, M. C. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In *Workshop on the Algorithmic Foundations of Robotics*, pages 279–291, 2000.

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

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

[49] E. Rimon and D. E. Koditschek. The construction of analytic diffeomorphisms for exact robot navigation on star worlds. In *IEEE Int. Conf. Robot. & Autom.*, pages 21–26, May 1989.

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

[51] J. T. Schwartz and M. Sharir. On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[52] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniqies for computing topological properties of algebraic manifolds. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[53] J. T. Schwartz and M. Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies. *Int. J. Robot. Res.*, 2(3):97–140, 1983.

[54] 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.

[55] T. Simeon, J.-P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000.

[56] 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.

[57] R. B. Tilove. Local obstacle avoidance for mobile robots based on the method of artificial potentials. In *IEEE Int. Conf. Robot. & Autom.*, pages 566–571, May 1990.

[58] S. A. Wilmarth, N. M. Amato, and P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis

of the free space. In *Proc. of the 1999 IEEE Int. Conf. Robot. & Autom.*, pages 1024–1031, 1999.

[59] L. Yang and S. M. LaValle. A framework for planning feedback motion strategies based on a random neighborhood graph. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 544–549, 2000.

[60] L. Yang and S. M. LaValle. An improved random neighborhood graph approach. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 254–259, 2002.

[61] D. J. Zhu and J. C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. Robot. & Autom.*, 7(1):9–20, February 1991.