Optimizations for Sampling-Based Motion Planning Algorithms

by

Joshua John Bialkowski

M.S., Massachusetts Institute of Technology (2010)
B.S., Georgia Institute of Technology (2007)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2014

Author .....
Department of Aeronautics and Astronautics
January 30, 2014

Certified by .....
Prof. Emilio Frazzoli
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Prof. Nicholas Roy
Associate Professor of Aeronautics and Astronautics

Certified by .....
Prof. Russ Tedrake
Associate Professor of Electrical Engineering and Computer Science

Accepted by .....
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Optimizations for Sampling-Based Motion Planning Algorithms

by

Joshua John Bialkowski

Submitted to the Department of Aeronautics and Astronautics
on January 30, 2014, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Sampling-basedalgorithms solve the motion planning problem by successively solving several separate suproblems of reduced complexity. As a result, the efficiency of the sampling-based algorithm depends on the complexity of each of the algorithms used to solve the individual subproblems, namely the procedures `GenerateSample`, `FindNearest`, `LocalPlan`, `CollisionFree`, and `AddToGraph`. However, it is often the case that these subproblems are quite related, working on common components of the problem definition. Therefore, distinct algorithms and segregated data structures for solving these subproblems might be costing sampling-based algorithms more time than necessary.

The thesis of this dissertation is the following: By taking advantage of the fact that these subproblems are solved repeatedly with similar inputs, and the relationships between data structures used to solve the subproblems, we may significantly reduce the practical complexity of sampling-based motion planning algorithms. Moreover, this reuse of information from components can be used to find a middle ground between exact motion planning algorithms which find an explicit representation ofthe collision-free space,and sampling-based algorithms which find no representation of the collision-free space, except for the zero-measure paths between connected nodes in the roadmap.

Thesis Supervisor: Prof. Emilio Frazzoli
Title: Professor of Aeronautics and Astronautics

Acknowledgments

First and foremost I would like to thank my thesis advisor, Emilio Frazzoli, for the years of guidance and collaboration that have led me through the exciting journey that has been my graduate career at MIT. It has been my priviledge to study under someone with such expertise, vision, and enthusiasm.

I am also indebted to my thesis committee, Russ Tedrake and Nick Roy, for their insight and encouragement on this thesis, and for what they have taught me. They have provided me with many of the tools that I will carry with me through my career.

I would also like to thank my collaborators, Sertac Karaman, Michael Otte, and Kyle Treleaven, and to my labmates. In the elaborate ritual of banging one's against a wall, it is easy to stray from reality in the search for the truth.

I am grateful beyond measure to Irene Chen, for all the many gifts she has given me, and to my family for their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A motion planning problem is, roughly speaking, a problem of finding a connected set of collision-free configurations that begin at a desired start region (or configuration) and end within a desired goal region. These problems arise in the development of automated systems including robotics in aerospace, underwater exploration, manufacturing, and warehouse management. These problems are also becoming relevant in in new industries such as personal transportation and personal robotics.

There are many *exact* motion planning algorithms which solve the motion planning problem (see, for instance, those in [56]), but the complexity of implementing them or the runtime required to solve the problem is often too high for practical use. In many cases, they require an exact decomposition of the collision-free configuration set, which may require storage exponential both in the representation of the obstacles and in the dimension of the workspace (as is the case in retraction planning on a triangulation [53]). LIDAR sensors, for instance, are capable of mapping a local environment as polygonal meshes with millions of triangles, meaning the obstacle set for far-field planning may be too large even to reside in computer memory at once. As a result, these algorithms are, in many cases, restricted to simple two or three dimensional problems with relatively small obstacle representations.

Alternatively, sampling-based motion planning algorithms forgo exact representations of the collision-free set for planning. Instead, they incrementally build a graph of paths, and thus require a collision checker which solves the much simpler problem of checking whether a single (often short or simply-represented) path is collision free. This approach has many advantages including conceptual simplicity, ease of implemen-

tation, and modularity. In many cases, the opportunistic approach of sampling-based algorithms can solve complex problems with less computational resources than would be required by an exact algorithm. It has proven to be an effective strategy even outside traditional motion planning problems in robotics, in the areas of computer graphics and synthetic biology [1, 20, 52]. Furthermore, the strategy can be used in so-called *any-time* applications, where an initial solution is found quickly and successively better solutions may be found by continuing to sample from the space.

As illustrated in the sequel, sampling-based algorithms solve the motion planning problem by successively solving several separate suproblems of reduced complexity. As a result, the efficiency of the sampling-based algorithm depends on the complexity of each of the algorithms used to solve the individual subproblems. However, it is often the case that these subproblems are quite related, working on common components of the problem definition. Therefore, distinct algorithms and segregated data structures for solving these subproblems might be costing sampling-based algorithms more time than necessary.

The thesis of this dissertation is the following: By taking advantage of the fact that these subproblems are solved repeatedly with similar inputs, and the relationships between data structures used to solve the subproblems, we may significantly reduce the practical complexity of sampling-based motion planning algorithms. In particular, as discussed below, we show that it is possible to improve the performance of (i) random sampling, (ii) nearest neighbor searching, and (iii) collision checking. Moreover, this reuse of information from components can be used to find a middle ground between exact motion planning algorithms which find an explicit representation of X_{free}, and sampling-based algorithms which find no representation of X_{free}, except for the zero-measure paths between connected nodes in the roadmap.

1.1 Problem statement

We are primarily concerned with the *motion planning* problem, which we may precisely define as in [45]. The *configuration space*, $X \subset \mathbb{R}^d$, is the set of all configurations of the robotic system. The *obstacle set*, *initial set*, and *goal set* are denoted X_{obs}, X_{start}, and X_{goal} respectively.

The *free space* is given by $X_{\text{free}} = \text{cl}\left(X \setminus X_{\text{obs}}\right)$ where $\text{cl}\left(\cdot\right)$ denotes set closure.

Given a function $\sigma : [0,1] \to \mathbb{R}^d$, its *total variation* is defined as follows [45]:

$$\text{TV}\left(\sigma\right) \triangleq \sup_{n \in \mathbb{N}, 0=t0<t1<\ldots<t_n=s} \sum_{i=1}^{n} \left|\sigma\left(t_i\right) - \sigma\left(t_{i-1}\right)\right|.$$

We say a function σ has *bounded variation* if $\text{TV}\left(\sigma\right) < \infty$, and a continuous function $\sigma : [0,1] \to \mathbb{R}^d$ with bounded variation is called a *path.*

A path is said to be *collision free* if it does not intersect the obstacle set, i.e., $\sigma\left(\tau\right) \in X_{\text{free}}$, $\forall \tau \in [0,1]$. A collision-free path is said to be *feasible* if it begins in the initial set and terminates in the goal set, i.e., $\sigma\left(0\right) \in X_{\text{start}}$, $\sigma\left(1\right) \in X_{\text{goal}}$. The *feasible motion planning problem* is to find a feasible path if one exists or to determine that one does not.

Problem 1 (Feasible Motion Planning)**.**

$$\begin{array}{ll} \textit{given} & X_{\text{free}},\ X_{\text{start}},\ X_{\text{goal}}, \\ \textit{find} & \sigma : [0,1] \to X_{\text{free}}, \\ \textit{subject to} & \sigma \in C^0, \\ & \text{TV}\left(\sigma\right) < 0, \\ & \sigma(0) \in X_{\text{start}}, \\ & \sigma(1) \in X_{\text{goal}}. \end{array}$$

Let us define Σ as the set of all solutions to a feasible motion planning problem. A functional $J : \Sigma \to \mathbb{R}_{\geq 0}$ is said to be a *cost functional* if it assigns a non-negative value (cost) to all non-trivial paths, i.e., $J\left(\sigma\right) = 0$ if and only if $\sigma\left(\tau\right) = \sigma\left(0\right)$, $\forall \tau \in (0,1]$, [45]. The *optimal motion planning problem* is to find a feasible path with minimum cost provided one exists, or determine if no feasible path exists.

Problem 2 (Optimal Motion Planning).

$$
\begin{aligned}
&\textit{given} && X_{\text{free}},\ X_{\text{start}},\ X_{\text{goal}},\ J,\\
&\underset{\tau}{\textit{minimize}} && J(\sigma),\\
&\textit{subject to} && \sigma : [0,1] \to X_{\text{free}},\\
& && \sigma \in C^0,\\
& && \mathrm{TV}(\sigma) < 0,\\
& && \sigma(0) \in X_{\text{start}},\\
& && \sigma(1) \in X_{\text{goal}}.
\end{aligned}
$$

It is often the case that the configuration space of the system is higher dimension then the physical (i.e., "real") space in which it operates. In this case we refer to the latter as the *workspace*, denoted $\mathcal{W} \subset \mathbb{R}^n$ (typically $n = [2, 3]$), and define a mapping $w : X \to \mathcal{W}$ from configurations to volumes of the workspace, i.e., given a configuration $\mathbf{x} \in X$, then $w(\mathbf{x}) \subset \mathcal{W}$ is the set of points in the workspace which are occupied by the system at configuration $\mathbf{x}$. If the obstacles are specified in terms of the workspace, i.e., $O_i \subset \mathcal{W}$, then free space is defined as $X_{\text{free}} = \{\mathbf{x} \in X \mid w(\mathbf{x}) \cap X_{\text{obs}} = \emptyset\}$.

1.2 State of the art

1.2.1 Complexity

There are several well known results regarding the fundamental complexity of motion planning problems. The generalized movers problem is PSPACE–Complete (PSPACE–Hard [86], PSPACE [17]). Even the somewhat simpler warehouse movers problem is PSPACE–hard [39], as is the game Sokoban (a simplified warehouse movers problem) [22].

1.2.2 Complete algorithms for specialized problems

The PSPACE complexity class makes complete and exact algorithms for complex (high dimensional, large obstacle representation) and general problem instances computationally impractical. Nevertheless, there are efficient algorithms for solving specializations of the motion planning

problem with two or three degrees of freedom. Many such algorithms are described in the books by Latombe, Goodman, and Lavalle [33, 53, 56].

1.2.3 Sampling-based motion planning

In contrast to exact algorithms, sampling-based motion planning algorithms are algorithms which solve the feasible or optimal motion planning problem by incremental construction of a roadmap of plans, through either random or quasi-random sampling sequences. As a general definition, a sampling-based algorithm takes as input a motion planning problem $(X_{\text{free}}, X_{\text{start}}, X_{\text{goal}})$, and an integer $n \in \mathbb{N}$, and return a graph $G_n = (V_n, E_n)$ where $V_n \subset X_{\text{free}}$, $\text{card}\,(V_n) \leq n$, $E_n \subset V_n \times V_n$. Sampling-based algorithms map a sequence of sampled configurations ω to a sequence of graphs. Let $G_n(\omega) = (V_n(\omega), E_n(\omega))$ denote the output of the algorithm, given ω.

Sampling-based algorithms address the complexity of motion planning problems by forgoing the traditional concepts of completeness and optimality in favor of *probabilistic* completeness or *asymptotic* optimality. Traditionally a *complete* algorithm returns a solution to the problem if one can be found, or returns an indication that no solution exists. Likewise an *optimal* algorithm (in the sense of optimal solutions, not of optimal complexity) returns an optimal solution if one exists or an indication that no solution exists. For sampling-based algorithms we consider the possible outcomes of an algorithm in terms of the event determining the sequence of samples, ω. We say that a sampling-based algorithm is *probabilistically complete* if, as the number of samples goes to infinity, the probability of finding a solution (given that a *robustly feasible*[45] solution exists) goes to unity.

$$\lim_{n\to\infty} P\,(\text{Sol'n Found} \mid \text{Sol'n Exists}\ ) = 1;$$

Likewise, if an optimal motion planning problem has a *robustly optimal*[45] solution with cost J^*, and the best solution found after n samples is σ_n^*, then we say that a sampling-based algorithm is *asymptotically optimal* if, as the number of samples goes to infinity, the cost of the best path $J\,(\sigma_n^*)$ approaches J^*, with probability one. Formally:

$$P\left(\lim_{n\to\infty} J\,(\sigma_n^*) = J^*\right) = 1.$$

Although they suffer reduced guarantees, sampling-based algorithms have proven to be quite effective at efficiently finding solutions to several difficult problems.

1.2.4 Sampling-based algorithms

One of the earliest sampling-based planning algorithms was the Randomized Path Planner (RPP) [8] which is based on the computation of a potential field and adds a random walk procedure to escape from local minima.

The "Ariadne's clew" algorithm [11] was developed based on placement of landmarks, or intermediate configurations. An explore subalgorithm iterates on the placement and number of these landmarks while a search subalgorithm attempts to find at least one landmark which can reach the goal by a simple Manhattan motion. The Ariadne's Clew algorithm utilizes genetic algorithms to optimize the location of landmarks and perform local search for Manhattan motions that reach the goal.

The Probabilistic Roadmap (PRM), introduced in [48], operates in two phases. In the learning phase, a roadmap is constructed with nodes corresponding to collision-free configurations, and edges corresponding to feasible paths between these configurations. In the query phase, a given start and goal configuration are added to this map, along with appropriate edges, and then a graph search algorithm is used to find a feasible path from the start configuration to the goal configuration.

The Rapidly Exploring Random Tree (RRT) [58, 57], is similar to the PRM but for single query problems. A roadmap is constructed similarly to PRM, but the graph is grown incrementally from the start configuration, and new configurations are only added as nodes to the roadmap if they can feasibly connect to a configuration already in the roadmap.

Following the introduction of these algorithms, many variations of PRM and RRT have been developed. However, despite significant diversity among sampling-based planning algorithms, they share a common high level structure based on the solution to several subproblems. They are generally composed of the following components: (i) a sampling scheme; (ii) a collision-checking procedure that determines if a point or path is in collision, given an obstacle set, (iii) a proximity search

procedure that returns "neighbors" to a point, given a point set; and (iv) a local planning procedure that returns a path between two given points. This structure represents the incremental computation of the graph G_{n+1} from the graph G_n, and is illustrated in pseudo code in Algorithm 1.1.

```
   Data: G_n = (E_n, V_n)
1  G_{n+1} ← G_n ;
2  x_sample ← GenerateSample ;
3  X_near ← FindNear (x_sample) ;
4  for x ∈ X_near do
5  |  σ ← LocalPlan (x, x_sample) ;
6  |  if CollisionFree (σ) then
7  |  |  G_{n+1} ←
   |  |  AddToGraph (σ, G_n) ;
8  return G_{n+1}
```

Algorithm 1.1: `ExtendGraph` (G), Many sampling-based motion planning algorithms share a common high level structure.

Variations of these algorithms such as RRT* and PRM* are shown to be asymptotically optimal and are designed to have the same asymptotic complexity bounds [45].

1.3 Components of sampling-based motion planning

As illustrated in Algorithm 1.1, sampling-based algorithms solve the motion planning problem by successively solving several separate suproblems of reduced complexity, namely the procedures `GenerateSample`, `FindNearest`, `LocalPlan`, `CollisionFree`, and `AddToGraph`.

1.3.1 Random Sampling

In most implementations the *true* randomness of the sampling sequence is non-crucial, and so-called pseudo random number (PRN) sequences are sufficient for the sampling process[15]. A pseudo random number

sequence is generated by a deterministic computational process which is chaotic, relatively equidistributed, uncorrelated, and has a long period. There are many popular algorithms for generating a PRN sequence such as BBS [12], Multiply-with-carry [65], Linear Congruential Generators [59] (the most common generator in programming languages), XOR-shift [66], Linear Feedback Shift, and of more recent popularity, Mersenne Twister (MT) [67] (which can be efficiently implemented on parallel hardware [101]). Each of these algorithms have complexity linear in the number of bits and several of them provide sequences with long periods and sufficient distributions to provide adequate notions of randomness to sampling-based planners.

Each of these algorithms generate sequences which are approximately uniformly distributed. Combined with the common choice of a hyperrectangular configuration space, they are a suitable source for random selection of points uniformly distributed over the workspace $\mathcal{W}$. However, it is the goal of sampling-based planners to build a tree of collision-free trajectories, so an ideal sample generator would sample uniformly from X_{free}.

In general practice, this is achieved by rejection sampling of X. This increases the complexity of the `GenerateSample` procedure by a factor proportional to the relative volume of X_{obs} with respect to X times the complexity of performing a collision check for a single configuration.

In addition to this fact, especially in the case of optimal motion planning, an independent sampling algorithm does not leverage information currently obtained about the environment by the sampling sequence.

1.3.2 Nearest neighbors

The `FindNearest` procedure in sampling-based motion planning generally solves the (k-)Nearest Neighbor problem or the Range Search problem. The k-Nearest Neighbor problem is defined as follows: Given a set of points $\mathbf{X}$, a query point $\mathbf{x}_{\text{q}}$, a distance function $d : X \times X \to \mathbb{R}_{\geq 0}$, and an integer k, find the set $\mathbf{X}_{\text{nearest}} \subset \mathbf{X}$ satisfying

$$\begin{aligned} \text{card}\left(\mathbf{X}_{\text{nearest}}\right) &= k, \\ \max_{\mathbf{x} \in \mathbf{X}_{\text{nearest}}} d\left(\mathbf{x}, \mathbf{x}_{\text{q}}\right) &\leq \min_{\mathbf{x} \in \mathbf{X} \setminus \mathbf{X}_{\text{nearest}}} d\left(\mathbf{x}, \mathbf{x}_{\text{q}}\right). \end{aligned}$$

The Nearest Neighbor problem, then, is equivalent to the k-Nearest Neighbor problem with $k = 1$. The Range Search problem is defined as follows: Given a set of points $\mathbf{X}$, a query point $\mathbf{x}_{\mathrm{q}}$, a distance function, and a parameter distance d, find the set $\mathbf{X}_{\mathrm{range}} = \{\mathbf{x} \in \mathbf{X} \mid d(\mathbf{x}, \mathbf{x}_{\mathrm{q}}) < d\}$.

A naïve algorithm for solving either of these problems by exhaustive search will take time $O(n)$ where $n = \mathrm{card}(\mathbf{X})$. However, there are many data structures (and with them algorithms) which yield much better complexity.

One of the most important of these data structures is the Voronoi diagram [6] (and its dual, the Delaunay graph). A hierarchy of Voronoi diagrams [23] can be searched in expected time $O(n \log n)$, while a persistent search tree [92] over the diagram can be searched in deterministic time $O(\log n)$, which is optimal for the Nearest Neighbor problem. However, it is known that there are degenerate configurations of points $\mathbf{X} \in \mathbb{R}^d$ such that the size of the diagram is $O(n^d)$, giving construction of the Voronoi diagram exponential complexity in the worst case. On the other hand, for a set of points $\mathbf{X}$ which are sampled at random from a uniform distribution, the expected number of Delaunay neighbors is $O(\alpha^d)$ for a constant α [24], meaning that on average the cost of incrementally constructing the Voronoi diagram will have an incremental complexity which is constant for constant dimension. In fact, if each point $\mathbf{x} \in \mathbf{X}$ is independent and identically distributed, then the Voronoi diagram can be constructed in linear expected time [25].

The problem of constructing a Voronoi diagram from a point set is well studied [70]. Methods of incremental construction [69, 51] are particularly useful for sampling-based motion planning algorithms since the point set itself is built incrementally.

In particular, it has been shown that there is a linear-time mapping from the problem of building a Voronoi diagram of a point set $\mathbf{X} \subset \mathbb{R}^d$ to the problem of building the convex hull of a point set $\mathbf{X}' \subset \mathbb{R}^{d+1}$ [16]. Thus, all of the results for algorithms and complexity of convex hulls applies to that of Voronoi diagrams. The problem is particularly well studied for points in two and three dimensions, such as the $O(n \log n)$ algorithm of [80]. In [94], a deterministic algorithm with complexity $O(n^2 + F \log n)$ for fixed dimension, and $F = O(n^{\lfloor d/2 \rfloor})$ is given. An optimally output-sensitive randomized algorithm constructing convex hulls in time $O(n^{\lfloor d/2 \rfloor})$ is given in [95]. An algorithm based on Random

Incremental Construction (RIC) is given in [19]. Another popular algorithm is the quickhull algorithm [7] and its randomized version [104].

While Voronoi diagrams are particularly useful in analysis, their worst-case complexity tends to discourage their use in many applications. There are many alternative spatial indexes including other spatial decompositions and search trees [90]. On of the most common search trees for nearest neighbor queries is the *kd*-tree [9, 10] or one of its many variants and derivatives such as the K-D-B tree [87], the hB-tree [63], divided *kd*-trees [103], and the Bkd-tree [81]. The *kd*-tree is particularly useful because it is very simple to implement, and for the case of incremental insertion of points independently sampled and identically distributed (i.i.d.) it is known to be asymptotically balanced such that nearest neighbor queries are $O(\log n)$.

More recent work on neighbor searching in very high dimension has yielded algorithms based on Locality Sensitive Hashing (LSH) [31]. Of particular note is the hashing function described in [4] for Euclidean distances. The most recent LSH algorithm is described in [5] and a nice survey is given in [3]. Parallel implementations of LHS for nearest neighbor searching using GPUs are described in [74, 75].

1.3.3 Collision checking

Collision checking for sampling-based algorithms is important for both the `GenerateSample` and `CollisionFree` procedures. There is an important distinction between the two. Much of the literature on collision checking focuses on the less complex problem of *discrete collision checking*, namely determining if a single configuration $\mathbf{x}$ is in collision, i.e., whether $w(\mathbf{x}) \cap X_{\text{obs}} \neq \emptyset$. More recent work has been focused on the problem of *continuous collision checking*: given a (usually infinite) set of configurations X, determine if any configuration is in collision, i.e., $\exists \mathbf{x} \in X \mid w(\mathbf{x}) \cap X_{\text{obs}} \neq \emptyset$.

Static collision checking is useful in rejection sampling for the `GenerateSample` procedure, but continuous collision checking is required for the `CollisionFree` procedure, as the parameter of that procedure is a continuous path through the configuration space.

In collision checking, it is generally assumed that the obstacles and the workspace volume of a configuration are both convex. While this assumption is not always realistic, it is required for the efficient compu-

tation of set distances and other spatial properties [14]. Furthermore, for many systems of interest an efficient convex decomposition of obstacles or configuration volumes is available.

Collision checking is generally based on the combination of two distinct components: *interference testing* and *spatial indexing*. An interference test checks if a configuration or set of configurations is in collision with a single obstacle. Spatial indexing is a strategy for creating a data structure which can reduce the number of obstacles that must be considered for interference testing. These components are often called broad-phase (navigating the index) and narrow-phase (interference testing) collision checking. A survey summarizing many of the following techniques is given in [44].

Interference testing

Discrete interference testing between two convex objects is very often implemented by the GJK algorithm [29]. Efficient exact interference testing in 3d for a convex polytope undergoing smooth motion among convex polytope obstacles is given in [18] by an algebraic representation of the interference predicates for configurations represented by a translation vector and a rotation quaternion. If the motion between two configurations is described by a polynomial vector function of the configuration the resulting predicates are polynomials and interference testing resolves to root-finding in a polynomial. In the special case that the motion is linear in a parameter s, the interference test requires finding the solution of a polynomial cubic in s. A derivative of this method which replaces actual motions between two configurations with an arbitrary surrogate that is sufficient for small time instances is given in [82]. For the case that the motion can be approximated by a sequence of screw motions (infinitesimal translations/rotations), a method is given in [49]. A combination of the two with the addition of conservative bounds given by interval arithmetic is given in [85].

Given that these methods rely on low-order representations of the path between configuration pairs, they tend to have less utility for systems that evolve with complex dynamics, or systems with a large number kinematic constraints (especially robotic manipulators). Alternative strategies are based on inexact continuous checking by discrete sampling, in many cases applying a static collision check at each of

the sample points. A method of adaptive segment checking is presented in [93]. Methods based on conservative certificates and interval arithmetic are described in [83, 84]. An extension using Taylor models to approximate incremental motion is described in [107]. A method based on refitting bounding spheres is given in [47].

Spatial indexing

Data structures used as spatial indexes generally fall under the category of spatial decompositions and bounding volume hierarchies [90]. As in the case of nearest neighbor searching, the Voronoi diagram plays an important role in collision checking. Especially in the case of two-dimensional systems and obstacles, the Voronoi diagram of convex sites yields an optimal data structure for set distances [68]. In the case where the system and obstacles are polygons the Voronoi diagram of line segments can be used [37, 46].

Voronoi diagrams provide an optimal spatial decomposition for set distance queries, but the complexity of constructing the diagram can prove to be unreasonable, especially in higher dimensions. One alternative based on non-optimal decomposition is a binary space partition (BSP) [102].

Bounding volume hierarchies (BVHs), especially those based on R-trees [36] are extremely popular for collision checking. One of the most common is the Axis-Aligned Bounding Box tree (AABB-tree). A combined method based on the R-tree of the Voronoi diagram is given in [96].

Voronoi diagrams, BSPs and BVHs are shown to have, at least in expectation, $O(\log n)$ search time (and thus, number of interference tests) for a static collision query, where n is the size of the obstacle representation, usually number of vertices or faces of the obstacle mesh.

Parallel hardware

By its nature, collision checking is a computationally intensive process. Interference testing is $O(n_f)$ in the number of features n_f and spatial index searching is often $O(\log n_o)$ in number of obstacles n_o. For many practical applications the values of n_f and n_o may be very large compared to other data structures in a planning algorithm. How-

ever, the low level mathematical operations to interference testing and index searching tend to be elementary for hardware arithmetic units. As such, new computational hardware designed with reduced capability but many more logical processors, namely, graphics co-processors (GPUs), are particularly well suited towards solving the collision checking problem.

In [34] and [35], implementations of bounding volume strategies for GPUs are presented. A BVH construction technique is presented in [54] and collision checking based on that construction in [55]. A parallel method based on maintenance of the colliding-front of the bounding volume tree is presented in [100]. In [40] an alternative to breadth-first construction order for GPU bounding volume hierarchies is proposed. A modern general implementation for production use is described in [76, 73, 77].

1.4 Organization

The remainder of this document is organized as follows: chapter 2 demonstrates the use of proximity search structures as a cache for collision information, allieviating the collision-checking bottleneck in c-space planning algorithms. Chapter 3 extends the algorithm of chapter 2 to workspace planning problems. Chapter 4 describes an algorithm for efficiently sampling from the configuration free space by storing emperical data in a proximity search bounding volume hierarchy. Chapter 5 extends the algorithm of chapter 4 to triangulations. Chapter 6 describes efficient proximity search structures for some common configuration spaces.

Chapter 2

Efficient Collision Checking in Sampling-based Motion Planning

Collision checking is widely considered the primary computational bottleneck of sampling-based motion planning algorithms (e.g., see [56]). This chapter demonstrates that this does not have to be the case. We introduce a novel collision checking implementation that has negligible amortized complexity vs. the proximity searches that are already at the core of sampling-based motion planning. As a consequence, proximity searches are identified as the main determinant of complexity in sampling-based motion planning algorithms, rather than collision checking.

Traditionally, collision checking is handled by passing a point query to a "Boolean black box" collision checker that returns either true or false depending on if the point or path is in collision with obstacles or not. In this chapter, we place a stronger requirement on the collision-checking procedure, which is now assumed to return a lower bound on the minimum distance to the obstacle set. Although computing such a bound is harder than checking whether a point is in collision or not, leveraging this extra information allows us to skip explicit collision checks for a large fraction of the subsequent samples. Indeed, the probability of calling the explicit collision checking procedure, and hence the amortized computational complexity due to collision checking, converges to zero as the number of samples increases.

It is useful to think of this method as *caching* discrete collision check

results inside the proximity search data structure. As proximity queries precede path collision checking in sampling-based algorithms, the cache fetch is implicit and occurs no additional work when used to accelerate continuous collision queries. If the two end configurations of a path both lie inside the certificate region, we may consider this a *cache hit* whereas if they do not, then this is a *cache miss*. Furthermore, if proximity queries are cheaper than discrete collision queries (i.e., the graph size is small enough), the proximity query may be performed instead before the discrete collision check to accelerate rejection sampling as well.

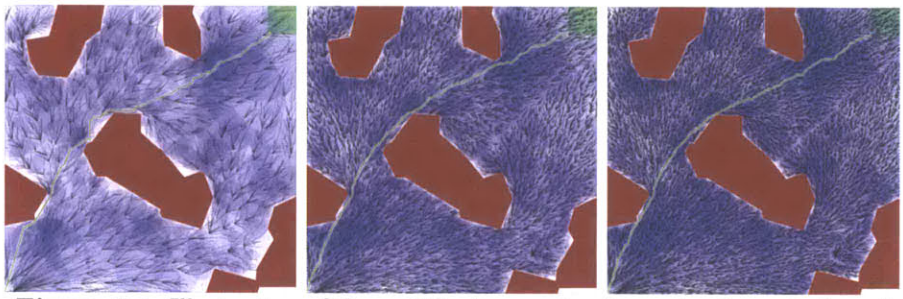

Figure 2-1: Illustration of the certificate covering for an implementation of RRT* for a planar point robot. As the number of samples increase (images from left to right) the likelihood of sampling from outside the certificates goes down.

The rest of this chapter is organized as follows: In Section 2.1 we introduce notation and state our problem and main results. In Section 2.2 we present our new collision checking method. In Section 2.3 we analyze the expected fraction of samples that will require a collision distance query and calculate the expected runtime savings of the proposed approach (e.g., for RRT, PRM, and many of their variants). In Section 2.4 we demonstrate performance improvement when used with RRT and RRT*. In Section 2.5 we draw conclusions and discuss directions for future work.

2.1 Notation, Problem Statement, and Main Results

Let $X = (0,1)^d$ be the configuration space, where $d \in \mathbb{N}$, $d \geq 2$ is the dimension of the space. Let X_{obs} be the obstacle region, such that $X \setminus X_{\text{obs}}$ is an open set, and denote the obstacle-free space as

$X_{\text{free}} = \text{cl}\,(X \setminus X_{\text{obs}})$, where $\text{cl}\,(\cdot)$ denotes the closure of a set. The initial configuration $\mathbf{x}_{\text{init}}$ is an element of X_{free}, and the goal region X_{goal} is an open subset of X_{free}. For the sake of our discussion, we will assume that sampling-based algorithms take as input a motion planning problem as problem 1 or problem 2, and return a sequence of graphs as in section 1.2.4. Let $G_n\,(\omega) = (V_n\,(\omega)\,, E_n\,(\omega))$ denote the output of the algorithm, given a sample sequence ω.

The problem we address in this chapter is how to reduce the complexity of sampling-based motion planning algorithms, i.e., the (expected) number of operations needed to compute G_n. In particular, we consider incremental complexity: the expected difference in the total number of operations needed to compute $G_{n+1}(\omega)$ and $G_n(\omega)$. In what follows, we characterize complexity based on the number of samples n and the environment description X_{obs}, while keeping the dimension d of the space a constant. For convenience, we assume stochastic sampling that draws uniformly and independently from X. Our results extend to deterministic and/or non-uniform sampling procedures, as long as the sampling procedure satisfy the technical conditions required for probabilistic or resolution completeness [56].

The computational complexity of a sampling-based algorithm can be decomposed in terms of the complexity of its primitive operations. The complexity of drawing a sample is bounded by a constant c_{sample}. The complexity of a collision check is bounded by $c_{\text{cc}}(X_{\text{obs}})$ a function that depends only on the description of the environment—note that the expected complexity of checking collisions with n_{obs} convex obstacles is $O\left(\log(n_{\text{obs}})^d\right)$ [90]. The complexity of (approximate) proximity searches that return the $O\,(\log n)$ nearest elements to a query point from a set of cardinality n is $O\,(\log n)$ [90]. The latter applies to k-nearest neighbor searches (k fixed or scaling as $\log n$), and range searches among uniform random samples for those points within a ball of volume scaling as $\log(n)/n$.

The complexity of the local planner is bounded by a constant c_{plan} that does not depend on the environment description. For convenience we also include in c_{plan} all other bookkeeping operations that are constant per candidate connection (cost updates, graph rewiring, etc.). Each candidate path generated by the local planner must also be checked for collisions, with complexity bounded by $c_{\text{path}}(X_{\text{obs}})$ a function that

depends only on the description of the environment.

The expected incremental complexity of sampling-based motion planning algorithms can be evaluated by examining the work induced by each new sample:

- RRT: Find the nearest neighbor, generate a new point, check the new point for collision, and connect the new point to the nearest neighbor.
- k-PRM: Collision check the sample, find and connect to its k-nearest neighbors.
- RRT*, PRM*: Collision check the sample, find the neighbors within a ball of volume scaling as $\log(n)/n$—or, equivalently, the $(k \log n)$-nearest neighbors, for a fixed k—and connect to them.

Table 2.1 summarizes the main contributions to the expected incremental complexity of these algorithms, in their standard implementation, based on collision-checking queries. Note that the local planning and path checking only occur if the new sample is not in collision with obstacles.

It is important to note that the asymptotic bounds may be misleading. In practice, the cost of collision checking the new sample and a new candidate connection $c_{\text{cc}}(X_{\text{obs}}) + c_{\text{path}}(X_{\text{obs}})$ often effectively dominates the incremental complexity, even for very large n. Hence, collision checking is typically regarded as the computational bottleneck for sampling-based motion planning algorithms. The cost of collision checking is even more evident for algorithms such as RRT* and PRM*, where collision checks are carried out for each of the $O(\log n)$ candidate connections at each iteration. Moreover, the ratio of the incremental complexity of RRT* to that of RRT depends on the environment via $c_{\text{path}}(X_{\text{obs}})$, and is potentially very large, although constant with respect to n.

In our approach, we replace standard collision checks with approximate minimum-distance computations. The complexity of computing a non-trivial lower bound on the minimum distance of a point from the obstacle set is also bounded by a function that depends on the description of the environment. In particular, we will require our collision checking procedure to return a lower bound $\bar{d}$ on the minimum distance d^* that satisfies $\alpha d^* \leq \bar{d} \leq d^*$, for some $\alpha \in (0, 1]$. The computational

cost of such a procedure will be indicated with $c_{\text{dist}}(X_{\text{obs}})$; in many cases of interest such a lower bound can be computed with a relatively small overhead vs. collision checking.

Algorithm	Proximity Search	Point Collision Checking	Local Planning	Path Collision Checking
RRT	$O(\log n)$	$c_{\text{cc}}(X_{\text{obs}})$	c_{plan}	$c_{\text{path}}(X_{\text{obs}})$
RRT (Modified)	$O(\log n)$	$O(\log n) + p_{\text{cc}}(n) c_{\text{cc}}(X_{\text{obs}})$	c_{plan}	$p_{\text{cc}}(n) c_{\text{path}}(X_{\text{obs}})$
k-PRM	$O(\log n)$	$c_{\text{cc}}(X_{\text{obs}})$	$k\, c_{\text{plan}}$	$k\, c_{\text{path}}(X_{\text{obs}})$
k-PRM (Modified)	$O(\log n)$	$O(\log n) + p_{\text{cc}}(n)\, c_{\text{cc}}(X_{\text{obs}})$	$k\, c_{\text{plan}}$	$k\, p_{\text{cc}}(n)\, c_{\text{path}}(X_{\text{obs}})$
RRT*, PRM*	$O(\log n)$	$c_{\text{cc}}(X_{\text{obs}})$	$O(\log n)\ c_{\text{plan}}$	$O(\log n)\ c_{\text{path}}(X_{\text{obs}})$
RRT*, PRM* (Modified)	$O(\log n)$	$O(\log n) + p_{\text{cc}}(n)\, c_{\text{cc}}(X_{\text{obs}})$	$O(\log n)\ c_{\text{plan}}$	$O(\log n)\, p_{\text{cc}}(n)\, c_{\text{path}}(X_{\text{obs}})$

Table 2.1: Asymptotic bounds on the expected incremental complexity of some standard sampling-based motion planning algorithms and of those same algorithms modified according to the proposed approach. p_{cc} is a function, such that $\limsup_{n\to\infty} p_{\text{cc}}(n) = 0$

The output of the minimum-distance computations is stored in a data structure that augments the standard graph constructed by sampling-based algorithms. Using this information, it is possible to reduce the number of explicit collision checks and minimum-distance computations that need to be performed. As we will later show, using our approach to modify the standard algorithms results in the asymptotic bounds on expected incremental complexity summarized in table 2.1, where p_{cc} is a function such that $\limsup_{n\to\infty} p_{\text{cc}}(n) = 0$. Again, the local planning and path checking only occur if the new sample is not in collision with obstacles.

Inspection of table 2.1 reveals that the incremental complexity of our version of RRT and k-PRM is less sensitive to the cost of collision checking. Moreover, the asymptotic ratio of the incremental complexity of our version of RRT* and RRT is a constant that does not depend on the environment.

2.2 Proposed Algorithm

Before describing our proposed modification to the collision checking procedure in section 2.2.1, we first formalize its primitive procedures and data structure.

Sampling: Let $\texttt{Sample} : \omega \mapsto \{\texttt{Sample}_i(\omega)\}_{i\in\mathbb{N}} \subset X$ be such that the random variables $\texttt{Sample}_i$, $i \in \mathbb{N}$, are independent and identically distributed (i.i.d.). The samples are assumed to be from a uniform

distribution, but results extend naturally to any absolutely continuous distribution with density bounded away from zero on X.

Nearest Neighbors: Given a finite point set $S \subset X$ and a point $x \in X$, the function `Nearest` : $(S, x) \mapsto s \in S$ returns the point in S that is closest to x,

$$\texttt{Nearest}\,(S, x) \triangleq \operatorname*{argmin}_{s \in S} \|x - s\|,$$

where $\|\cdot\|$ is a metric (e.g., Euclidean distance—see [57] for alternative choices). A set-valued version is also considered, `kNearest` : $(S, x, k) \mapsto \{s_1, s_2, \ldots, s_k\}$, returns the k vertices in S that are nearest to x with respect to $\|\cdot\|$. By convention, if $\text{card}\,(S) < k$, then the function returns S.

Near Vertices: Given a finite point set $S \subset X$, a point $x \in X$, and a positive real number $r \in \mathbb{R}_{>0}$, the function `Near` : $(S, x, r) \mapsto S_{\text{near}} \subseteq S$ returns the vertices in S that are inside a ball of radius r centered at x,

$$\texttt{Near}\,(S, x, r) \triangleq \{s \in S \,|\, \|s - x\| \leq r\}\,.$$

Set distance: Given a closed set $S \subset X$ and a point $x \in X$, `SetDistance` returns a non-trivial lower bound on the minimum distance from x to S, i.e., for some $\alpha \in (0, 1]$,

$$\alpha \min_{s \in S} \|s - x\| \leq \texttt{SetDistance}\,(S, x) \leq \min_{s \in S} \|s - x\|.$$

Segment Collision Test: Given points $x, y \in X$, `CFreePath`(x, y) returns `True` if the line segment between x and y lies in X_{free}, i.e., $[x, y] \subset X_{\text{free}}$, and `False` otherwise.

Data Structure: We achieve efficient collision checking by storing additional information in the graph data structure that is already used by most sampling-based algorithms. Specifically, we use the "augmented graph" $AG = (V, E, S_{\text{free}}, S_{\text{obs}}, \text{Dist})$, where $V \subset X_{\text{free}}$ and $E \subset V \times V$ are the usual vertex and edge sets. The sets $S_{\text{free}} \subset X_{\text{free}}$ and $S_{\text{obs}} \subset X_{\text{obs}}$ are sets of points for which an explicit collision check has been made. For points in S_{free} or S_{obs} the map $\text{Dist} : S_{\text{free}} \cup S_{\text{obs}} \to \mathbb{R}_{\geq 0}$ stores the approximate minimum distance to the obstacle set or to the free space, respectively. The vertices V and edges E are initialized according to the particular sampling-based algorithm in use. The sets

S_{free} and S_{obs} are initialized as empty sets.

2.2.1 Modified Collision Checking Procedures

Using the augmented graph data structure allows us to modify the collision checking procedures as shown in algorithm 2.1 and 2.2, respectively for points and paths. For convenience, we assume that the augmented data structure AG can be accessed directly by the collision checking procedures, and do not list it as an input.

Point Collision Test: Given a query point $\mathbf{x}_q \in X$, the function `CFreePoint`($\mathbf{x}_q$) returns `True` if $\mathbf{x} \in X_{\text{free}}$, and `False` otherwise. When a new sample $\mathbf{x}_q$ is checked for collision, we first check if we can quickly determine whether it is in collision or not using previously computed information, lines 2.1.1-2.1.6. In particular, we use the configuration $\mathbf{x}_{\text{free}} \in S_{\text{free}}$ that is nearest to $\mathbf{x}_q$ (found on line 2.1.1). If $\mathbf{x}_q$ is closer to $\mathbf{x}_{\text{free}}$ than the $\mathbf{x}_{\text{free}}$ is to an obstacle, then clearly $\mathbf{x}_q$ is collision free, lines 2.1.3-2.1.4. Otherwise, we find the configuration $\mathbf{x}_{\text{obs}} \in S_{\text{obs}}$ that is nearest to $\mathbf{x}_q$, line 2.1.2. If $\mathbf{x}_q$ is closer to $\mathbf{x}_{\text{obs}}$ than $\mathbf{x}_{\text{obs}}$ is to the free space, then clearly $\mathbf{x}_q$ is in collision, line 2.1.5-2.1.6.

If these two checks prove inconclusive, then a full collision check is performed using set distance computations. First, one can compute the approximate minimum distances from $\mathbf{x}_q$ to the obstacle set, and to the free set, respectively indicated with d_{obs} and d_{free}, lines 2.1.7-2.1.8. If $d_{\text{obs}} > 0$, $\mathbf{x}_q$ is in X_{free} and is added to the set S_{free} of vertices that have been explicitly determined to be collision-free, lines 2.1.9-2.1.10. Furthermore, Dist($\mathbf{x}_q$) is set to be equal to d_{obs}, line 2.1.11. Otherwise, $\mathbf{x}_q$ is in X_{obs}, and is added to the set S_{obs}, with Dist($\mathbf{x}_q$) = d_{free}, lines 2.1.13-2.1.15.

Path Set Collision Test: Given a vertex set $S \subseteq V$ and query point $\mathbf{x}_q$, the function `BatchCFreePath`($S, \mathbf{x}_q$) returns a set of edges $H \subseteq (V \cup \{\mathbf{x}_q\}) \times (V \cup \{\mathbf{x}_q\})$ such that for all $h = (x, y) \in H$, `CollisionFreePath`(x, y) evaluates to `True`. The form of S depends on the particular planning algorithm that is being used, e.g.,

- RRT: $S =$ `Nearest`$(V, \mathbf{x}_q)$ is the single nearest point to $\mathbf{x}_q$ in V.
- k-PRM: $S =$ `kNearest`(V, x, k) contains up to k nearest neighbors to $\mathbf{x}_q$ in V.

```
1  x_free ← Nearest(S_free, x_q);
2  x_obs ← Nearest(S_obs, x_q);
3  if ‖x_q − x_free‖ ≤ Dist(x_free)
   then
4  |  return True
5  else if ‖x_q − x_obs‖ ≤ Dist(x_obs)
   then
6  |  return False
7  d_obs ← SetDistance(X_obs, x_q) ;
8  d_free ← SetDistance(X_free, x_q) ;
9  if d_obs > 0 then
10 |  S_free ← S_free ∪ {x_q};
11 |  Dist(x_q) ← d_obs;
12 |  return True
13 else
14 |  S_obs ← S_obs ∪ {x_q};
15 |  Dist(x_q) ← d_free;
16 |  return False
```

Algorithm 2.1: `CFreePoint`($\mathbf{x}_q$)

```
1  H ← ∅;
2  x_near ← Nearest(S_free, x_q);
3  foreach x ∈ S do
4  |  if ‖x − x_near‖ ≤ Dist(x_near)
   |  then
5  |  |  H ← H ∪ {(x, x_q)};
6  |  else if CFreePath(x, x_q) then
7  |  |  H ← H ∪ {(x, x_q)};
8  return H
```

Algorithm 2.2: `BatchCFreePath`($S, \mathbf{x}_q$)

```
1  d ← (Ax_q − c) ;
2  d_min = −‖ − d‖_∞ ;
3  d_max = ‖d‖_∞ ;
4  if d_max > 0 then
5  |  return (False, d_max) ;
6  else
7  |  return (True, −d_min) ;
```

Algorithm 2.3: `ApproxCDistPolytope`($A, \mathbf{c}, \mathbf{x}_q$)

- RRT* and PRM*: S contains $O(\log n)$ points—either the points within a ball of volume scaling as $\log(n)/n$ centered at $\mathbf{x}_q$, or the $(k \log n)$-nearest neighbors to $\mathbf{x}_q$.

We assume `CFreePoint`($\mathbf{x}_q$) is called prior to `BatchCFreePath`($S, \mathbf{x}_q$), and thus $\mathbf{x}_q$ is collision-free. For each pair (s, x), $s \in S$, the first step is to check whether the segment $[s, x]$ can be declared collision-free using only the information already available in AG, lines 2.2.2-2.2.5. Let $\mathbf{x}_{\text{near}} \in S_{\text{free}}$ be the vertex in S_{free} that is nearest to $\mathbf{x}_q$, line 2.2.2. If both s and $\mathbf{x}_q$ are closer to $\mathbf{x}_{\text{near}}$ than $\mathbf{x}_{\text{near}}$ is to the obstacle set, then clearly the segment $[s, \mathbf{x}_q]$ is collision free, lines 2.2.4-2.2.5 ($\|\mathbf{x}_q - \mathbf{x}_{\text{near}}\| \leq \text{Dist}(v_{\text{near}})$ is automatic, given that `CFreePoint`($\mathbf{x}_q$) has already been called). If this check is inconclusive, then a full collision check is performed, lines 2.2.6-2.2.7.

Approximate set collision distance: The set distance computation method depends on the choice of an obstacle index. For two dimensional polygonal obstacles, a segment Voronoi diagram is an index for which set distance computation is efficient and exact. For polyhedral obstacles with halfspace representation $\{x : Ax \leq c\}$, algorithm 2.3 computes a non-trivial lower bound on the set distance in $O(n_f d)$ time for n_f faces in d dimensions. If J^* is the true set distance, then algorithm 2.3 will return a distance J satisfying $\alpha J^* \leq J \leq J^*$ where

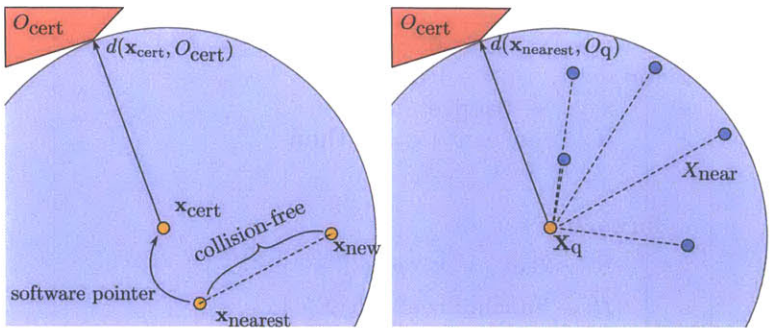

Figure 2-2: (left) If the query point and its nearest neighbor are both within the certificate, then the path between them is collision free. (right) Likewise, if any candidates for connection also lie within the same certificate, then the path between is implicitly collision free.

$\alpha > 0$ depends on the smallest angle between two adjacent faces. For polytopes obstacles represented by vertex sets, GJK [29] provides similar approximate set distances and can be extended to provide exact set distances.

Efficient collision checking: We now illustrate how to improve the efficiency of standard sampling-based motion planning algorithms. As an example, Algorithms 2.4 and 2.5 show modified versions of the RRT and PRM* (pseudo-code based on [45]). AG is initialized on lines 2.4.1/2.5.1. Standard point collision checks are replaced with `CFreePoint` (i.e., Algorithm 2.1), lines 2.4.4/2.5.4. The code generating neighbor connections from a new sample is modified to generate those connections in a batch manner via a single call to `BatchCFreePath` (i.e., Algorithm 2.2), lines 2.4.6/2.5.8.

```
1  V ← {x_init}; E ← ∅; S_free ← ∅; S_obs ← ∅ ;
2  for i = 1, ..., n − 1 do
3  |   x_rand ← Sample_i (ω);
4  |   if CFreePoint(x_rand) then
5  |   |   x_nearest ← Nearest (E, x_rand);
6  |   |   H ← BatchCFreePath ({x_nearest} , x_rand);
7  |   |   if H ≠ ∅ then
8  |   |   |   V ← V ∪ x_rand;
9  |   |   |   E ← E ∪ H;
10 return G = (V, E);
```

Algorithm 2.4: Modified RRT

```
1  V ← {x_init}; E ← ∅; S_free ← ∅; S_obs ← ∅ ;
2  for i = 1, ..., n − 1 do
3  |  x_rand ← Sample_i (ω);
4  |  if CFreePoint (x_rand) then
5  |  |  V ← V ∪ {x_rand} ;
6  foreach x ∈ V do
7  |  S ← Near (V, x, γ_PRM (log(n)/n)^{1/d}) ;
8  |  H ← BatchCFreePath (S \ {x}, x);
9  |  E ← E ∪ H;
10 return G = (V, E);
```

Algorithm 2.5: Modified PRM*

2.3 Analysis

Suppose ALG is a sampling-based motion planning algorithm. Let $I_{\text{point}}(n)$ denote the indicator random variable for the event that the `SetDistance` procedure in algorithm 2.1 is called by ALG during the iteration in which the n-th sample is drawn. Similarly, define $I_{\text{path}}(n)$ as the indicator random variable for the event that the `CFreePath` procedure in algorithm 2.2 is called by ALG in the iteration in which the n-th sample is drawn. Define $I(n) \triangleq I_{\text{point}}(n) + I_{\text{path}}(n)$, and let $p_{\text{cc}}(n) = \mathbb{E}[I(n)]$. We prove our main result for algorithms satisfying the following assumptions:

(A1) ALG calls the `CFreePoint` procedure $O(1)$ times per iteration;

(A2) At the n-th iteration, ALG calls the batch `CFreePath` procedure with $O(\log n)$ paths, and all such paths lie inside a ball of radius $o(n^{-1/(d+1)})$ as $n \to \infty$.

Our main result is stated in the following theorem.

Theorem 1. *Suppose ALG satisfies Assumptions (A1) and (A2). Then ALG, implemented using the proposed collision-checking algorithm, has zero asymptotic expected incremental set-distance complexity, i.e.,*

$$\limsup_{n \to \infty} p_{\text{cc}}(n) = 0.$$

Note that all algorithms in Table 2.1 satisfy Assumptions (A1) and (A2). Hence, the results summarized in Table 2.1 can be deduced from Theorem 1.

The rest of this section is devoted to the proof of Theorem 1. The key idea behind the proof is to divide up the state space into several cells. This particular discretization allows us to compute a bound on the expected number of samples that require Algorithm 2.1 to call the set-distance procedure. We show that this number grows slower than n, which implies Theorem 1.

Consider a partitioning of the environment into cells, where the size and number of cells used in the partition grows with the number of samples n. More precisely, divide up the configuration space $[0,1]^d$ into openly-disjoint hypercube cells with edge length $l_n := n^{-\frac{1}{d+1}}$. Given $\mathbf{z} \in \mathbb{Z}^d$, we define the cell with index $\mathbf{z}$ as the L^∞ ball of radius l_n centered at $l_n\,\mathbf{z}$, i.e., $C_n(\mathbf{z}) := \left\{x' \in X : \|x' - l_n\mathbf{z}\|_\infty \leq l_n\right\}$, where $l_n\,\mathbf{z}$ is the scalar-vector multiplication of $\mathbf{z}$ by l_n, and $\|\cdot\|_\infty$ is the infinity norm. Let $\mathbf{Z}_n \subset \mathbb{Z}^d$ denote the smallest set for which $\mathbf{C}_n := \{C_n(\mathbf{z}) : \mathbf{z} \in \mathbf{Z}_n\}$ covers the configuration space, i.e., $X \subseteq \bigcup_{\mathbf{z}\in\mathbf{Z}_n} C_n(\mathbf{z})$. Clearly, $\mathbf{Z}_n$ is a finite set, since X is compact.

Let γ_u denote the maximum distance between two points in the unit hypercube using the distance metric $\|\cdot\|$ employed in the `SetDistance` procedure. For instance, if $\|\cdot\|$ is the usual Euclidean metric (L_2 norm), then $\gamma_u = \sqrt{d}$; if $\|\cdot\|$ is the L_∞ norm, then $\gamma_u = 1$.

We group the cells in $\mathbf{C}_n$ into two disjoint subsets, namely the *boundary cells* $\mathbf{B}_n$ and the *interior cells* $\mathbf{I}_n$. Let $\mathbf{B}_n'$ denote the set of all $C_n(\mathbf{z}) \subset \mathbf{C}_n$ that include a part of the obstacle set boundary, i.e., $C_n(\mathbf{z}) \cap \partial X_{\text{obs}} \neq \emptyset$. Then $\mathbf{B}_n$ contains exactly those cells that are within a cell-distance of at most $2\left(\lceil(1/\alpha)\,\gamma_u\rceil + 1\right)$ to some cell in $\mathbf{B}_n'$,

$$\mathbf{B}_n := \left\{C_n(\mathbf{z}) : \|\mathbf{z} - \mathbf{z}'\|_\infty \leq 2\left(\lceil(1/\alpha)\,\gamma_u\rceil + 1\right) \text{ for some } \mathbf{z}' \text{ with } C_n(\mathbf{z}') \in \mathbf{B}_n'\right\},$$

where α is the constant in the constant-factor lower bound in the computation of the `SetDistance` procedure. Finally, $\mathbf{I}_n$ is defined as exactly the set of those cells that are not boundary cells, i.e., $\mathbf{I}_n := \mathbf{C}_n \backslash \mathbf{B}_n$. The reason behind our choice of the number $2(\lceil(1/\alpha)\gamma_u\rceil + 1)$ will become clear shortly.

Let $\lambda(\cdot)$ denote the Lebesgue measure in $\mathbb{R}^d$. The total number of cells, denoted by K_n, can be bounded for all n as follows [1]:

$$K_n \leq \frac{\lambda(X)}{\lambda(C_n(\mathbf{z}))} = \frac{\lambda(X)}{\left(n^{-\frac{1}{d+1}}\right)^d} = \lambda(X)\, n^{\frac{d}{d+1}}. \tag{2.1}$$

Notice the number of cells, K_n, is an increasing and sub-linear function of n.

Let B_n denote the number of boundary cells.

Lemma 1. *There exists a constant $c_1 > 0$ such that $B_n \leq c_1\,(K_n)^{1-1/d}$ for all $n \in \mathbb{N}$.*

Proof. This result follows from the fact that the obstacle boundary can be covered by $N^{(d-1)/d} = N^{1-1/d}$ cells, where N is the number of equal-size cells that cover the configuration space. □

Thus the fraction of cells that are boundary cells can be bounded by

$$\frac{c_1\, B_n}{K_n} = \frac{c_1\,(K_n)^{1-1/d}}{K_n} = c_1\,(K_n)^{-1/d} \leq c_1\,(\lambda(X))^{-1/d}\, n^{-\frac{1}{d+1}} = c_2\, n^{-\frac{1}{d+1}},$$

where c_2 is a constant.

We now bound the number of calls to the collision-distance procedure by examining the number of such calls separately for samples that fall into interior cells or boundary cells, respectively. Recall that S_{free} and S_{obs} are defined as the vertices that are explicitly checked and found to be in X_{free} and X_{obs}, respectively. Define $C := S_{\text{free}} \cup S_{\text{obs}}$. Our key insight is summarized in the following lemma, which will be used to derive a bound on the number of points in C that fall into interior cells.

Lemma 2. *Given any algorithm in Table 2.1, suppose that algorithm is run with n samples using our collision checking algorithm. Let $C_n(\mathbf{z}) \in \mathbf{I}_n$ be some interior cell. Then, there exists at most one vertex from C in $C_n(\mathbf{z})$, i.e.,*

$$|C_n(\mathbf{z}) \cap C| \leq 1 \text{ for all } C_n(\mathbf{z}) \in \mathbf{I}_n.$$

[1] Strictly speaking, this bound should read: $K_n \leq \lceil \lambda(X)\, n^{d/(d+1)} \rceil$, where $\lceil \cdot \rceil$ is the standard ceiling function (i.e., $\lceil a \rceil$ returns the smallest integer greater than a). We will omit these technical details from now on to keep the notation simple.

Before proving Lemma 2, we introduce some additional notation and establish two intermediate results. Let $\mathbf{N}_n(\mathbf{z}) \subseteq \mathbf{C}_n$ denote the set of all cells that have a cell distance of at most $\lceil \gamma_u \rceil + 1$ to $C_n(\mathbf{z})$, i.e., $\mathbf{N}_n(\mathbf{z}) := \{C_n(\mathbf{z}') : \|\mathbf{z}' - \mathbf{z}\|_\infty \leq \lceil \gamma_u \rceil + 1\}$. $\mathbf{N}_n(\mathbf{z})$ includes the cell $C_n(\mathbf{z})$ together with all its neighbors with cell distance less than $\lceil \gamma_u \rceil + 1$. The cells in $\mathbf{N}_n(\mathbf{z})$ are such that they exclude points that are sufficiently far away from points in $C_n(\mathbf{z})$ and they include points that are sufficiently far away from the boundary of the obstacle set. The last two properties are made precise in the following two lemmas.

Lemma 3. *Any point that lies in a cell outside of* $\mathbf{N}_n(\mathbf{z})$ *has distance at least* $\gamma_u \, l_n$ *to any point that lies in* $C_n(\mathbf{z})$*, i.e.,* $\|x - x'\| \geq \gamma_u \, l_n$ *for all* $x \in C_n(\mathbf{z})$ *and all* $x' \in C_n(\mathbf{z}')$ *with* $C_n(\mathbf{z}') \notin \mathbf{N}_n(\mathbf{z})$.

Proof. The claim follows immediately by the construction of $\mathbf{N}_n(\mathbf{z})$. □

Lemma 4. *Any point in a cell from* $\mathbf{N}_n(\mathbf{z})$ *has distance at least* $\big(\lceil (1/\alpha)\, \gamma_u \rceil + 1\big) l_n$ *to any point that lies on the obstacle set boundary, i.e.,* $\|x - x'\| \geq \big(\lceil (1/\alpha)\, \gamma_u \rceil + 1\big) l_n$ *for all* $x \in \partial X_{\text{obs}}$ *and all* $x' \in C_n(\mathbf{z}')$ *with* $C_n(\mathbf{z}') \in \mathbf{N}_n(\mathbf{z})$.

Proof. The interior cell $C_n(\mathbf{z})$ has a cell distance of at least $2(\lceil (1/\alpha)\, \gamma_u \rceil + 1)$ to any cell that intersects with the boundary of the obstacle set. Any cell in $\mathbf{N}_n(\mathbf{z})$ has a cell distance of at most $\lceil \gamma_u \rceil + 1 \leq \lceil (1/\alpha)\, \gamma_u \rceil + 1$ to $C_n(\mathbf{z})$. Thus, by the triangle inequality (for the cell distance function), any cell in $\mathbf{N}_n(\mathbf{z})$ has a cell distance of at least $\lceil (1/\alpha)\, \gamma_u \rceil + 1$ to any cell that intersects the obstacle boundary. □

Now we are ready to prove Lemma 2.

Proof of Lemma 2. The proof is by contradiction. Suppose there exists two points $x_1, x_2 \in C$ that fall into $C_n(\mathbf{z})$. Suppose x_1 is added into C before x_2. Let x_{nearest} denote the nearest point when Algorithm 2.1 was called with x_2.

First, we claim that x_{nearest} lies in some cell in $\mathbf{N}_n(\mathbf{z})$. Clearly, x_{nearest} is either x_1 or it is some other point that is no farther than x_1 to x_2. Note that x_1 and x_2 has distance at most $\gamma_u \, l_n$. By Lemma 3, any point that lies in a cell outside of $\mathbf{N}_n(\mathbf{z})$ has distance at least $\gamma_u \, l_n$ to x_2. Thus, x_{nearest} must lie in some cell in $\mathbf{N}_n(\mathbf{z})$. However, $\texttt{Dist}(x_{\text{nearest}}) \geq (1/\alpha)\, \|x_2 - x_{\text{nearest}}\|$ by Lemma 4; In that case, x_2 should have never

been added to C, even when the `SetDistance` procedure returns α-factor of the actual distance to the obstacle set boundary. Hence, we reach a contradiction. □

The following lemma provides an upper bound on the *expected* number of samples that fall into boundary cells, thus an upper bound on the expected number of points in C that fall into a boundary cell.

Lemma 5. *Let X_n denote a set of n samples drawn independently and uniformly from X. Let S_n denote the number of samples that fall into boundary cells, i.e.,*

$$S_n := \Big|\{x \in X_n : x \in C_n(\mathbf{z}) \text{ with } C_n(\mathbf{z}) \in \mathbf{B}_n\}\Big|.$$

Then, there exists some constant c_3, independent of n, such that $\mathbb{E}[S_n] \leq c_3\, n^{\frac{d}{d+1}}$.

Proof. Let E_i denote the event that the i-th sample falls into a boundary cell. Let Y_i denote the indicator random variable corresponding to this event. From Lemma 1 and the discussion following it, the fraction of cells that are of boundary type is $c_2 n^{-\frac{1}{d+1}}$. Thus, $\mathbb{E}[Y_i] = \mathbb{P}(E_i) = c_2\, i^{-\frac{1}{d+1}}$ and $\mathbb{E}[S_n] = \mathbb{E}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \mathbb{E}[Y_i] = \sum_{i=1}^{n} c_2\, i^{-\frac{1}{d+1}} \leq c_2 \int_1^n x^{-\frac{1}{d+1}} dx$, where $c_2 \int_1^n x^{-\frac{1}{d+1}} dx = \frac{c_2\,(d+1)}{d}\left(n^{\frac{d}{d+1}} - 1\right)$. Thus, $\mathbb{E}[S_n] \leq c_3\, n^{\frac{d}{d+1}}$, where c_3 is a constant that is independent of n. □

The following lemma gives an upper bound on the number of points in C.

Lemma 6. *There exists a constant c_4, independent of n, such that*

$$\mathbb{E}\left[\text{card}\,(C)\right] \leq c_4\, n^{\frac{d}{d+1}}.$$

Proof. On one hand, the number of points in C that fall into an interior cell is at most the total number of interior cells by Lemma 2, and thus less than the total number of cells K_n, which satisfies $K_n \leq \lambda(X)\, n^{\frac{d}{d+1}}$ (see Equation (2.1)). On the other hand, the expected number of points in C that fall into a boundary is no more than the expected number of samples that fall into boundary cells, which is bounded by $c_3\, n^{\frac{d}{d+1}}$, where c is a constant independent of n (see Lemma 5). Thus, we conclude that $\mathbb{E}\left[\text{card}\,(C)\right] \leq c_4\, n^{\frac{d}{d+1}}$ for some constant c_4. □

Finally, we are ready to prove Theorem 1.

proof of Theorem 1. First, we show that p_{cc} is a non-increasing function of n using a standard coupling argument. We couple the events $\{I(n) = 1\}$ and $\{I(n+1) = 1\}$ with the following process. Consider the run of the algorithm with $n+1$ samples. Let A_n and A_{n+1} denote the events that the nth and the $(n+1)$st samples are explicitly checked, respectively. Clearly, $\mathbb{P}(A_{n+1}) \leq \mathbb{P}(A_n)$ in this coupled process, since the first $(n-1)$st samples are identical. Moreover, $\mathbb{P}(A_n) = \mathbb{P}(\{I(n) = 1\}) = p_{\text{cc}}(n)$ and $\mathbb{P}(A_{n+1}) = \mathbb{P}(\{I(n+1) = 1\}) = p_{\text{cc}}(n+1)$. Thus, $p_{\text{cc}}(n+1) \leq p_{\text{cc}}(n)$ for all $n \in \mathbb{N}$. Note that this implies that $\lim_{n\to\infty} p_{\text{cc}}(n)$ exists.

Next, we show that $\lim_{n\to\infty}(1/n)\sum_{k=1}^{n} p_{\text{cc}}(k) = 0$. Clearly, $\sum_{k=1}^{n} I_{\text{point}}(k) = |C|$. Hence,

$$\frac{\sum_{k=1}^{n}\mathbb{E}[I_{\text{point}}(k)]}{n} = \frac{\mathbb{E}[\sum_{k=1}^{n} I_{\text{point}}(k)]}{n} \leq \frac{c_4\, n^{\frac{d}{d+1}}}{n} = c_4\, n^{-\frac{1}{d+1}},$$

where the inequality follows from Lemma 6. Similarly,

$$\limsup_{k\to\infty}(\mathbb{E}[I_{\text{path}}(k)] - \mathbb{E}[I_{\text{point}}(k)]) = 0,$$

since all paths fit into an Euclidean ball with radius $o(n^{-1/(d+1)})$, which is asymptotically smaller than the side length of each cell $l_n = n^{-1/(d+1)}$. Hence,

$$\begin{aligned}\limsup_{n\to\infty}\frac{\sum_{k=1}^{n} p_{\text{cc}}(k)}{n} &= \limsup_{n\to\infty}\left(\frac{\sum_{k=1}^{n}\mathbb{E}[I_{\text{point}}(k)] + \mathbb{E}[I_{\text{path}}(k)]}{n}\right)\\ &= \limsup_{n\to\infty}\frac{\mathbb{E}[\sum_{k=1}^{n} I(k)]}{n} \leq \limsup_{n\to\infty}\frac{c_4\, n^{\frac{d}{d+1}}}{n} = \limsup_{n\to\infty} c_4\, n^{-\frac{1}{d+1}} = 0,\end{aligned}$$

Finally, $p_{\text{cc}}(n+1) \leq p_{\text{cc}}(n)$ for all $n \in \mathbb{N}$ and

$$\lim_{n\to\infty}(1/n)\sum_{k=1}^{n} p_{\text{cc}}(k) = 0$$

together imply that $\lim_{n\to\infty} p_{\text{cc}}(n) = 0$. □

2.4 Experiments

To validate and demonstrate the utility of our method, we compared the performance of implementations of the RRT and RRT* algorithms, both with and without our proposed modification for collision checking. Our implementations are single threaded, utilize a *kd*-tree [9] for point-proximity (i.e., nearest neighbor and k- nearest neighbor) queries, and a segment-Voronoi hierarchy [6] for set distance queries. The experiments were run on a 1.73 GHz Intel Core i7 computer with 4GB of RAM, running Linux. We performed experiments on an environment consisting of a unit-square workspace with 150 randomly placed convex polygonal obstacles (see Figure 2-3). The goal region is a square with side length 0.1 units at the top right corner; the starting point is at the bottom left corner.

Figure 2-3 (Left) shows both the tree of trajectories generated by RRT*, and the set of collision-free balls that are stored in the augmented graph. After 2,000 samples, the collision-free balls have filled a significant fraction of the free space, leaving only a small amount of area uncovered near the obstacle boundaries. As proved in Section 2.3, any future sample is likely to land in a collision-free ball common to both it and its nearest neighbor, making future collision checks much less likely.

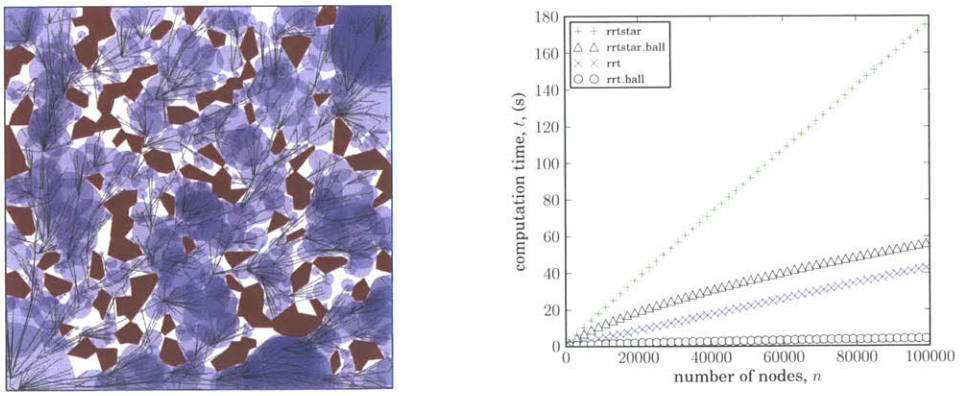

Figure 2-3: (Left) Search tree, black, and balls where explicit collision checks are unnecessary, purple, for the modified RRT* algorithm with $n = 2,000$. (Right) Runtimes for the four algorithms tested, averaged over 30 runs of each algorithm, and using graph size buckets of 1000 nodes.

To generate timing profiles, both RRT and RRT* were run on the obstacle configuration in Figure 2-3 with and without the use of the new

collision algorithm. In each run, both variants of both algorithms are run with the same initial random seed for the point sampling process so that the points sampled in each algorithm are the same (i.e., the sample sequence ω is the same for both algorithms).

Figure 2-3 (Right) illustrates the wall-clock measured runtime required to reach various tree sizes for each of these four implementations. Runtimes are averaged over 30 runs and in graph-size buckets of 1000 nodes. As expected from the amortized complexity bounds, longer runs enable the proposed approach to achieve greater savings. For example, the runtime of RRT* is reduced by 40% at 10,000 vertices, and by 70% at 100,000 vertices, vs. the baseline implementation. The runtime of RRT is reduced by 70% at 10,000 vertices, and by 90% at 100,000 vertices.

The increased efficiency stemming from our proposed approach also results in an effective increase in the rate at which RRT* converges to the globally optimal solution. these implementations. Figure 2-4 (Left) illustrates the cost of the best path found in the baseline RRT* and the modified RRT*. In general and on average, the modified RRT* finds paths of lower cost significantly faster than the baseline RRT*.

Figure 2-4 (Right) illustrates the average number of explicit checks required for points which are not in collision vs. different graph sizes. As the number of samples added to the database grows, the probability of performing an explicit check decreases. At a graph size of 100,000 nodes, on average only one out of every 100 samples required an explicit check when the implementations reached that point.

Figure 2-5 illustrates the computation of a single iteration of the four algorithms at various different graph sizes and for two different obstacle configurations. Increasing the number of obstacles increases the average iteration time while the graphs remain small; as the graph grows, the iteration times for higher obstacle count case approaches that of the lower obstacle count case.

A Voronoi diagram obstacle index makes exact collision distance computation no more expensive than collision checking, however generating a Voronoi diagram in higher dimensions is prohibitively complicated. To address this, we compare the performance of the RRT with and without our proposed modification in a second implementation utilizing a *kd*-tree for point-proximity searches, and an axis-aligned

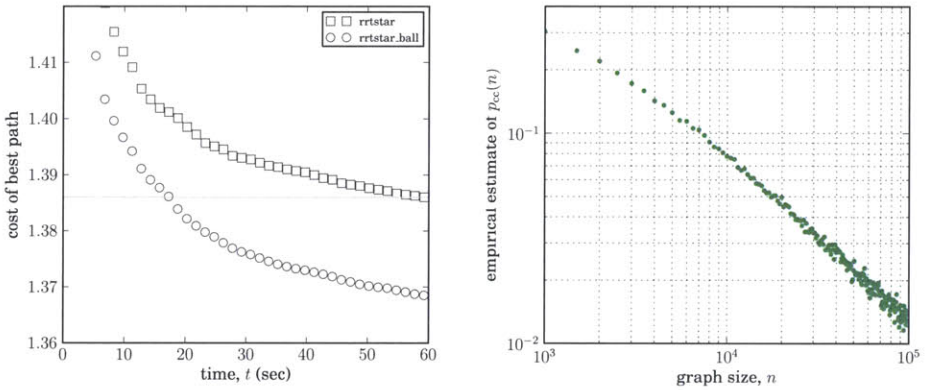

Figure 2-4: (Left) RRT* best-path cost vs. time, with and without the proposed method, averaged over 30 runs and in buckets of 100 time samples. The proposed method yields significantly faster convergence. (Right) The experimental probability of performing an explicit collision query vs. graph size. Only 1% of new nodes require an explicit check when graph size is 100,000 nodes.

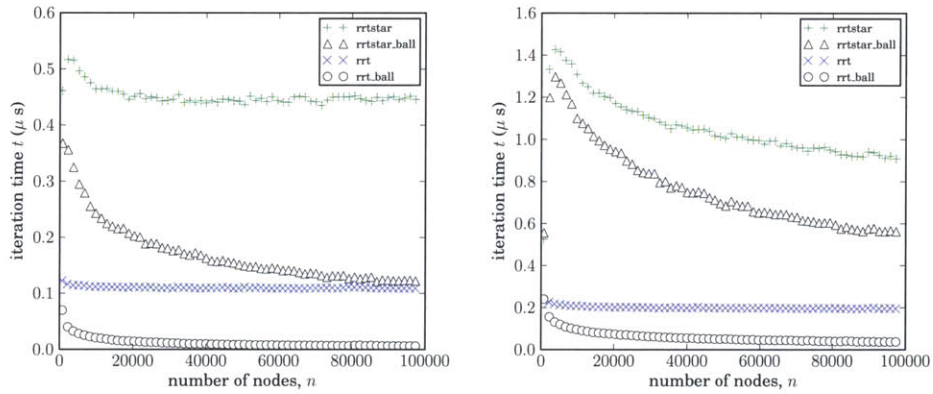

Figure 2-5: The average computation time of a single iteration vs. number of nodes for each algorithm (plot style), over 30 runs, in a configuration with 500 and 1000 obstacles (Left and Right). Environments with more obstacles initially have longer iteration times, but approach those with less obstacles as the number of graph nodes increases.

bounding box tree with Algorithm 2.3 for set distance queries and collision checking. Obstacles are generated as randomly placed simplices.

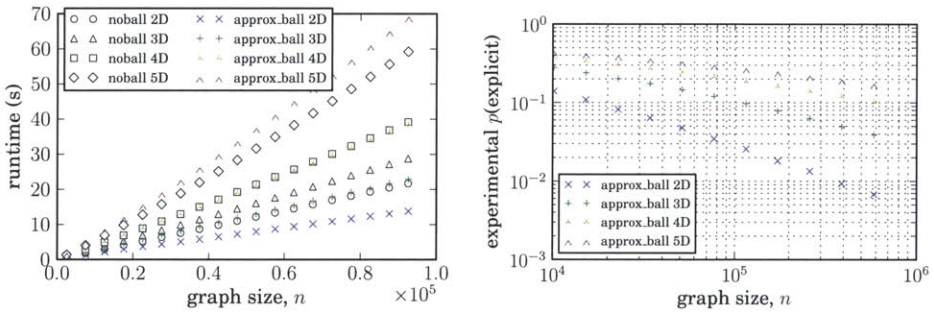

Figure 2-6: Mean wall-clock runtime (Left) and experimental p_{explicit} (Right), over 20 runs, for RRT in a unit workspace $X = [0, 1]^d$ for several d.

Figure 2-6 illustrates the measured runtime of this implementation, as well as the observed frequency of performing an explicit collision check for a unit workspace in 2 to 5 dimensions. For this choice of index, the modified algorithm using an approximate collision distance reaches a given graph size faster in 2 and 3 dimensions, at the roughly the same speed in 4 dimensions, and slower in 5 dimensions. The number of explicit collision checks is also shown. There is a significant degradation of the performance in higher dimensions. In general, the volume coverage of a d-ball becomes less significant at higher dimensions. However, we note that uniform placement of obstacles is a somewhat degenerate scenario in higher dimensions. For many problems of interest, it is likely that obstacles will lie in a lower dimensional subspace, in which case hyper-cylindrical, rather than spherical, certificates would be more prudent.

2.5 Conclusions

This chapter introduces a novel approach to collision checking in sampling-based algorithms. This approach allows us to demonstrate, both theoretically and experimentally, that collision-checking is not necessarily a computational bottleneck for sampling-based motion planning algorithms. Rather, the complexity is driven by nearest-neighbor searches within the graph constructed by the algorithms. Experiments indicate significant runtime improvement in practical implementations.

The proposed approach is very general but there are some important implementation details to be aware of when using it. First, we indicate that points which are sampled in collision are kept in order to characterize the obstacle set (i.e., in addition to those that are not in collision). While this allows that the expected number of explicit collision checks goes to zero, the number of points required to truly characterize the obstacle set can be quite large, especially in high dimension.In practice, strategies for biasing samples away from the obstacle set are likely to be more effective then keeping points which are sampled in collision. If in-collision samples are not kept and no biasing is used, the expected number of explicit checks will approach the proportion of the workspace volume which is in collision.However, even in such a case, the strategy described in this chapter is effective at marginalizing the extra collision checks in the asymptotically optimal variants of sampling-based motion planning algorithms. As an example, the expected runtime ratio between RRT* and RRT will be a constant which does not depend on the obstacle configuration, even if no in-collision samples are kept.

In addition, we show that calculating a sufficient approximation of the collision distance of a particular point and obstacle often does not require more computation than the worst case of performing a collision query. While this is true for a single obstacle, it is important to note that collision checking is often done using a spatial index and the choice of index may affect how the efficiency of a collision distance query compares to a simple collision query.

Also, In practice our method may benefit multi-query algorithms (e.g. PRM) and asymptotically optimal algorithms (e.g. PRM*, RRT*) more than single-query feasible planning algorithms (e.g. RRT). The latter return after finding a single path, and thus experience less ball coverage of the free space and proportionately more explicit checks, in a sparse environment. The multi-query and asymptotically optimal algorithms require more collision checks per new node, and so offer more opportunities for savings.

Lastly, some of the steps leverage the fact that in path planning problems, the straight line connecting two points is a feasible trajectory for the system. This is in general not the case for robotic systems subject to, e.g., differential constraints. Extensions to such systems is a topic of current investigation.

Chapter 3

Workspace Certificates

In the previous chapter we illustrated the utility of caching discrete collision checks to accelerate future collision queries in sampling-based algorithms. The proposed method relies on the ability to efficiently compute the collision distance in configuration space. It is often the case, however, that collision information for the system and obstacles are represented in a physical workspace, and cannot be efficiently mapped to configuration space. Furthermore, the nearest obstacle in the workspace is not necessarily the nearest obstacle in configuration space.

In this chapter we demonstrate the tools necessary to utilize the collision cache with workspace certificates for rigid body systems and articulated models. In particular Canny's method of non-linear interpolation between orientations [18] provides a method for local planning of articulated robots which is as straight forward to implement as straight line segments for point robots, and admits algebraic conditions for continuous collision checking which can be evaluated to arbitrary precision for points of contact. For an articulated model the polynomial expression for the motion of a particular joint of the robot grows with the length of the armature, but we show that the theory of Sturm sequences allows for efficient and robust continuous collision checking of such paths without solving for the exact roots of these polynomials.

3.1 Preliminaries

3.1.1 Collision geometries

A *polyhedron* P in $\mathbb{R}^n$ is the set of points $\mathbf{x} \in \mathbb{R}^n$ satisfying

$$\begin{aligned} \mathbf{n}_i \cdot \mathbf{x} &< d_i & i &\in \mathcal{I} \\ \mathbf{n}_j \cdot \mathbf{x} &\leq d_j & j &\in \mathcal{J} \end{aligned}$$

for some set of inequality constraints. We refer to this definition as the *half-space representation* of a polyhedron (a polyhedron is the set of $x \in \mathbb{R}^n$ formed by the intersection of a finite number half-spaces).

A polyhedron is *bounded* if it is a bounded subset of $\mathbb{R}^n$, or, equivalently, if the number of linearly independent constraint equations strictly greater than n. A *polytope* is the convex hull of a finite number of points. A polytope is a bounded polyhedron. A polytope is *open* if it is an open subset of $\mathbb{R}^n$, or, equivalently, if all inequalities are strict. A polytope is *closed* if it is a closed subset of $\mathbb{R}^n$, or, equivalently, if none of the inequalities are strict. We will assume that all collision geometries (both obstacles and workspace volumes of the system) are polytopes in $\mathbb{R}^3$, or in $\mathbb{R}^2$, and all certificate volumes are polyhedra.

Let $P \subset \mathbb{R}^n$, $n = 2, 3$ be a polyhedron, the ith *face* $f_i \subset P$ is the set of points $\mathbf{x}$ satisfying

$$\mathbf{n}_i \cdot \mathbf{x} = d_i, \tag{3.1}$$

$$\mathbf{n}_j \cdot \mathbf{x}, \leq d_j \qquad j \neq i. \tag{3.2}$$

We are often concerned only with the *oriented plane coincident to face* i, which is the set of points satisfying only equation 3.1. When this context is understood, we may also denote this plane as f_i. By the convention that all inequalities are "less than", faces are *oriented*, such that the vector $\mathbf{n}_i$ is a a vector normal to f_i and points outside the polyhedron.

Let $P \subset \mathbb{R}^3$ be a polyhedron. An *edge* $e_{i,j} \subset P$ is the set of points satisfying exactly two of the constraint equations with equality, and all others with inequality. This set forms a line segment in $\mathbb{R}^3$. When we refer to an edge of P this line segment has an arbitrary orientation. We

may describe the edge as

$$\{\mathbf{x} \,|\, \mathbf{x} = \mathbf{x}_0 + s\mathbf{e}, s \in [0,1]\},$$

where $\mathbf{x}_0$ is one endpoint of the line segment and $\mathbf{e} = \mathbf{x}_1 - \mathbf{x}_0$ with $\mathbf{x}_1$ the other endpoint. In this case edges may be *oriented* with respect to f_i such that the vectors $\mathbf{e}_j$ point counter-clockwise around the face. In other words the vector $\mathbf{n}_i \times \mathbf{e}_j$ points to the interior of f_i for all j edges of face i. We are often concerned only with the *line coincident to edge i* which is the set of all points which satisfy the two equality constraints of the edge. When this context is understood, we may also denote this line as e_j.

Let $P \subset \mathbb{R}^n$ be a polyhedron. A *vertex* $\mathbf{v}_i \in P$ is a point which satisfies n constraints equations with equality, and all others with inequality. We note that an *open* polytope does not contain its vertices, edges, and faces.

There are several different methods for efficiently storing the representation of a polyhedron on a digital computer. For the remainder of this paper we do not necessarily assume any particular representation, but that the following procedures are efficient.

procedure	**description**
$\mathbf{v}_i \leftarrow$ `Vertex` (i, P),	retrieves the ith vertex of P
$f_i \leftarrow$ `Face` (i, P)	retrieves the ith face of P
$(\mathbf{x}_0, \mathbf{e}) \leftarrow$ `Edge` (j, f_i)	retrieves the jth edge of face f_i
$(\mathbf{n}, d) \leftarrow$ `Constraint` (j, f_i)	retrieves the jth constraint of face f_i

Table 3.1: Primitive procedures for polytope data structures

We also assume it is efficient to iterate over these primitives, but make no assumptions about the ordering of vertices, faces, or edges with respect to their indices.

3.1.2 Articulated model

An *armature* is a tree of reference frames each one referred to as a *link* of the armature. We assign a unique index to each link in the armature and denote the reference frame of the ith link as F_i. Each link has a unique parent. We refer to the *root* of the armature as the single link whose parent is the global reference frame.

Each reference frame in the armature is related to its parent by a rotation R_i and a translation $\mathbf{T}_i$. Given a vector $\mathbf{x}$ represented in F_i, the same point represented in its parent frame F_j is given by:

$$\mathbf{x}^j = \mathbf{T}_i + R_i\mathbf{x}.$$

A *collision volume* is a closed set of points in the workspace which are occupied by the model. Collision volumes are parameterized with respect to the reference frame of a link of the armature (i.e., they "belong" to a particular link).

We will assume that all collision volumes are closed polytopes. While it is common to model the geometry of a robot as a non-convex triangular mesh there are efficient automated algorithms for generating approximate convex decompositions [60, 64] from such meshes. By taking the convex hulls of this approximate decomposition, we may efficiently and automatically generate a convex cover of the approximation of the original mesh. As a closed polytope, a collision volume may be represented as

$$C_i = \{\mathbf{x} \,|\, N_i\mathbf{x} \leq \mathbf{d}_i\},$$

where the vector inequality $\mathbf{x} \leq \mathbf{y}$ indicates that $x_i \leq y_i$ for each of the ith elements of the vectors $\mathbf{x}$ and $\mathbf{y}$, and N_i is a matrix which is the row concatenation of face normals, and $\mathbf{d}_i$ is a matrix which is the row concatenation of face offsets. We use the notation $C_i = (N_i, \mathbf{d}_i)$ to denote the specification of a collision volume. Note that N_i must have at least $n + 1$ linearly independent rows.

If a link i is parented to link j then we say that links i and j are *joined*. Link i may be free to translate and/or rotate with respect to its parent, potentially with constraints on the motion. We refer to the specification of these constraints on motion between a link and its parent as a *joint* of the armature.

As an example, in robotic manipulators, it is often the case that a link may rotate about a fixed axis with respect to its parent. We refer to this as a *one degree of freedom (1-DOF)) rotational joint* (a.k.a a hinge joint, or elbow joint). A link may also rotate about an arbitrary axis with respect to its parent. We refer to this as a *three degree of freedom (3-DOF) rotational joint* (a.k.a conical joint, or shoulder joint).

We refer to the combined specification of an armature, its collision

volumes, and joints as an *articulated model*.

The *configuration* of an articulated model is a specification of the translation $\mathbf{T}_i$ and rotation R_i for each link i of the armature. The set of all possible configurations for an articulated model is called the *configuration space*, denoted X.

An obstacle is an open set of points in the workspace. In order for a motion plan to be *collision free* it must be true that for all collision volumes C_i and all obstacles O_j, $C_i \cap O_j = \emptyset$.

We will assume that all collision volumes are open, not necessarily bounded polyhedra, and thus can be represented as

$$O_i = \{\mathbf{x} \,|\, N_i\mathbf{x} < \mathbf{d}_i\}\,.$$

We use the notation $O_i = (N_i, \mathbf{d}_i)$ to denote the specification of an obstacle. Note that N_i and $\mathbf{d}_i$ may have any number of rows.

Given a point $\mathbf{x}$ as a vector in frame F_i, we may find the vector in the global frame which represents that point by successively applying the inverse rotation and translation along the ancestry of link i. We may represent this compactly by the use of *homogeneous coordinates* [56]. In $\mathbb{R}^n$, $n \in \{2, 3\}$, we may represent the combined rotation of R_i and translation of $\mathbf{T}_i$ by a $(n+1)\times(n+1)$ matrix of a certain form. Because this rotation and translation define a frame with respect to its parent, we use F_i to denote both the ith frame, and the homogeneous matrix used to transform coordinates in frame i to coordinates in the parent frame. We generate a homogeneous coordinate vector $\underline{\mathbf{x}} \in \mathbb{R}^{n+1}$ from a vector $\mathbf{x} \in \mathbb{R}^n$ by row concatenation with unity:

$$F_i\,\underline{\mathbf{x}} = \begin{bmatrix} R_i & \mathbf{T}_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{T}_i + R_i\mathbf{x} \\ 1 \end{bmatrix}. \tag{3.3}$$

In this representation, the equation for $\mathbf{x}^g$ represented in the global frame is

$$\underline{\mathbf{x}}^g = \left(\prod_{j\in J} F_j\right)\underline{\mathbf{x}}, \tag{3.4}$$

where J is the sequence of link indices defining the ancestry of link i: $J = \{0, \ldots, i\}$, with 0 the root link and i the link of the frame in which $\mathbf{x}$ is represented.

3.1.3 Exponential map from $so(3)$ to $SO(3)$

There are many different ways of parameterizing the group of rotations in R^3. We describe here the axis, angle parameterization $so(3)$ and its relation to rotation matrices $SO(3)$.

Given an axis $\mathbf{v}$, $|\mathbf{v}| = 1$, and an angle $\theta \in \mathbb{R}$, we may describe the rotation about $\mathbf{v}$ by the angle θ as a single vector $\boldsymbol{\omega} \in \mathbb{R}^3$, $\boldsymbol{\omega} = \mathbf{v}\theta$. As a vector describing a rotation, we say that $\boldsymbol{\omega} \in so(3)$. We then define the exponential map from $so(3)$ to $SO(3)$.

$$\begin{aligned} \theta &= \|\boldsymbol{\omega}\|, \\ \mathbf{v} &= \frac{1}{\theta}\boldsymbol{\omega}, \\ \exp(\omega) &= I\cos\theta + \widehat{\mathbf{v}}\sin\theta + (1-\cos\theta)\mathbf{v}\mathbf{v}^\top, \end{aligned} \tag{3.5}$$

where $\widehat{\mathbf{x}}$ is a skew symmetric "cross-product" matrix:

$$\widehat{\mathbf{x}} = \begin{bmatrix} 0 & -\mathbf{x}_2 & \mathbf{x}_1 \\ \mathbf{x}_2 & 0 & -\mathbf{x}_0 \\ -\mathbf{x}_1 & \mathbf{x}_0 & 0 \end{bmatrix}.$$

Equation 3.5 is also known as *Rodrigues' formula* for rotations. The inverse (log) map from $SO(3)$ to $so(3)$ is given by

$$\begin{aligned} \theta &= \cos^{-1}\left(\frac{\text{trace}(R)}{2}\right), \\ \mathbf{v} &= \frac{1}{2\sin\theta}\begin{bmatrix} R(2,1) - R(1,2) \\ R(0,2) - R(2,0) \\ R(1,0) - R(0,1) \end{bmatrix}, \\ \log(R) &= \mathbf{v}\theta. \end{aligned} \tag{3.6}$$

Note that, unlike the exponential map, the log map is not defined everywhere. In particular, consider the case that $\theta = \pi$. In this case we may revisit Rodrigues' formula substituting π for θ and derive

$$\mathbf{v}\mathbf{v}^\top = \frac{1}{2}(R + I).$$

The diagonal terms of $1/2\,(R + I)$ gives the squared magnitude of the elements of $\mathbf{v}$ and the off-diagonal elements yield the signs up to a sign ambiguity. Thus there are exactly two possibilities for $(\theta, \mathbf{v})$, given R.

We will represent orientations in $\mathbb{R}^3$ as rotation vectors when used as configuration variables, and as rotation matrices when transforming between frames.

3.1.4 Sturm sequences

Given a polynomial $p(x)$, a *Sturm sequence of* $p(x)$ is a finite sequence of polynomials $p_0(x), p_1(x), \ldots p_n(x)$ of decreasing degree satisfying the following [38]:

1. $p_0 = p$ has no repeated roots
2. if $p_0(x) = 0$ then $\operatorname{sgn}(p_1(x)) = \operatorname{sgn}(p_0'(x))$
3. if $p_i(x) = 0, 0 < i < n$ then $\operatorname{sgn}(p_{i-1}(x)) = -\operatorname{sgn}(p_{i+1}(x))$
4. $\operatorname{sgn}(p_n)$ is constant

where $\operatorname{sgn}(x)$ is the signum function defined as

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0. \end{cases}$$

Theorem 2 (Sturm's Theorem)**.** *Let $p_0, p_1, \ldots, p_n$ be a Sturm sequence of the square-free polynomial $p(x)$, and let $\sigma(x) : \mathbb{R} \to \mathbb{N}$ denote the number of sign changes in the sequence*

$$p_0(x),\ p_1(x),\ \ldots,\ p_n(x),$$

where a sign of 0 is ignored. Then, for any two real numbers $u, v \in \mathbb{R}$, $u < v$, the number of distinct real roots of p in the half-open interval $(u, v]$ is $|\sigma(u) - \sigma(v)|$.

If p is not square-free, then Sturm's theorem holds for u, v so long as neither u nor v is a multiple root of p. A Sturm sequence of the

polynomial p may be constructed as follows:

$$\begin{aligned} p_0(x) &= p(x), \\ p_1(x) &= p'(x), \\ p_2(x) &= p_1(x)q_0(x) - p_0(x), \\ &\cdots \\ p_n(x) &= p_{n-1}(x)q_{n-2}(x) - p_{n-2}(x), \end{aligned}$$

where $q_i(x)$ is the quotient of $p_{i-1}(x)/p_{i-2}(x)$. In other words $p_i(x)$ is the remainder after polynomial long division of $p_{i-1}(x)$ by $p_{i-2}(x)$.

3.2 Local planning

3.2.1 Canny interpolation

For sampling-based motion planning, a requisite component is a *local planning method.* The local planner takes two configurations and generates a path $\sigma : [0,1] \to X$. For efficient sampling-based planning, the generated plan should be computed efficiently, admit efficient collision queries, and satisfy a notion of optimality: for a distance metric $d : X \times X \to \mathbb{R}$

$$\lim_{d(\mathbf{x}_0,\mathbf{x}_1)\to 0} {}^{J(\sigma)}/_{J^*} = 1,$$

where $J^* = \min_{\sigma\in\Sigma} J(\sigma)$ and Σ denotes the set of all possible paths between configurations $\mathbf{x}_0$ and $\mathbf{x}_1$. A common method of local planning is to use linear interpolation between configurations. For a particular configuration variable (say θ_i for a 1-DOF joint) we choose $\theta_i(s) = \theta_i^0 + s(\theta_i^1 - \theta_i^0)$. This satisfies the requirements of being computed efficiently and satisfies a notion of optimality for many distance metrics, but is difficult to compute points of contact. The method of interpolating orientations developed by Canny [18] using the stereographic projection of quaternions is asymptotically equivalent to linear interpolation as the magnitude of the angle goes to zero, but admits an efficient method of collision checking. We derive equations for representing Canny's method of interpolation using rotation matrices and homogeneous transforms in order to derive algebraic conditions of collision for linked frames.

Consider a vector $\mathbf{x}$ represented in the frame F_i of link i. At the

initial configuration that reference frame is translated and rotated with respect to its parent by by $\mathbf{T}_0$ and R_0, and at the target configuration that reference frame is translated and rotated with respect to its parent by $\mathbf{T}_1$, R_0.

The path σ between the two configurations is generated on a link by link basis as follows: The translation $\mathbf{T}_i(s)$ is found by a linear interpolation:

$$\mathbf{T}(s) = \mathbf{T}_0 + s\,(\mathbf{T}_1 - \mathbf{T}_0)\,, \qquad 0 \leq s \leq 1. \tag{3.7}$$

We generate intermediate rotations by linearly interpolating the tangent of the angle. We first find the *direct* relative rotation $\Delta R_{0,1}$ from R_0 to R_1

$$\Delta R_{0,1} = R_1 R_0^\top.$$

Then we determine the axis-angle representation of that rotation by Equation 3.6,

$$\begin{aligned} \boldsymbol{\omega} &= \log(\Delta R_{0,1}), \\ \Delta\theta &= \|\boldsymbol{\omega}\|, \\ \mathbf{v} &= \frac{1}{\Delta\theta}\boldsymbol{\omega}. \end{aligned}$$

Note that the magnitude of the direct rotation is at most π. Further note that if the magnitude of the rotation is exactly $\Delta\theta = \pi$, then the log map is undefined. As noted earlier, there are in fact two axis-angle rotations which satisfy Rodrigues' formula in this case. For the purpose of local planning, we may simply consider both rotations.

We will generate a matrix polynomial in s which yields a rotation about $\mathbf{v}$ by an angle $\theta(s)$ such that $\tan\theta(s) = s\tan(\Delta\theta)$. We do this by first defining an intermediate frame F_i' in which the x axis is coincident with $\mathbf{v}$. The translation for F_i' is zero, and its rotation is given by:

$$\begin{aligned} \mathbf{v}' &= \frac{\mathbf{e}_0 \times \mathbf{v}}{\|\mathbf{e}_0 \times \mathbf{v}\|}, \\ \theta' &= \sin^{-1}\|\mathbf{e}_0 \times \mathbf{v}\|, \\ R' &= \exp(\theta'\mathbf{v}'). \end{aligned}$$

Within this intermediate frame, rotation about the vector $\boldsymbol{\omega}$ is expressed as a rotation about the x axis. At an intermediate angle θ, $0 \leq \theta \leq \Delta\theta$, we may compute the orientation along minimum rotation

path by the following equation:

$$R(\theta) = R_x(\theta) R' R_0,$$

where R_x is the basic rotation matrix about the x axis of magnitude θ, given by

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}.$$

Because a rotation by θ is equivalent to two rotations by $\theta/2$ we have

$$\begin{aligned} R_x(\theta) &= R_x\left(\frac{\theta}{2}\right)^2 \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta/2 & -\sin\theta/2 \\ 0 & \sin\theta/2 & \cos\theta/2 \end{bmatrix}^2 \\ &= \cos(\theta/2)^2 \begin{bmatrix} \sec(\theta/2)^2 & 0 & 0 \\ 0 & 1-\tan(\theta/2)^2 & -2\tan(\theta/2) \\ 0 & 2\tan(\theta/2) & 1-\tan(\theta/2)^2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \frac{1}{1+\tan(\theta/2)^2} \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\tan(\theta/2)^2 & -2\tan(\theta/2) \\ 0 & 2\tan(\theta/2) & -\tan(\theta/2)^2 \end{bmatrix}. \end{aligned}$$

Or, more compactly:

$$\left(1+\tan(\theta/2)^2\right) R_x(\theta) = \begin{bmatrix} 1+\tan(\theta/2)^2 & 0 & 0 \\ 0 & -\tan(\theta/2)^2 & -2\tan(\theta/2) \\ 0 & 2\tan(\theta/2) & -\tan(\theta/2)^2 \end{bmatrix}.$$

We may ensure that any linear equation involving $R_x(s)$ is a polynomial in s by picking $\theta(s)$ such that $\tan\theta/2$ is linear in s. Thus define

$$\gamma = \tan\Delta\theta/2, \tag{3.8}$$

$$\theta(s) = 2 * \tan^{-1}(s\gamma). \tag{3.9}$$

Then we have that

$$\begin{aligned}\tan\left({}^{\theta(s)}/_{2}\right) &= \tan\left(\left({}^{1}/_{2}\right)2\tan^{-1}(s\gamma)\right), \\ &= \tan\left(\tan^{-1}(s\gamma)\right), \\ &= s\gamma.\end{aligned}$$

Note that this choice means that orientations are interpolated linearly along a central projection of the unit sphere in quaternion space.

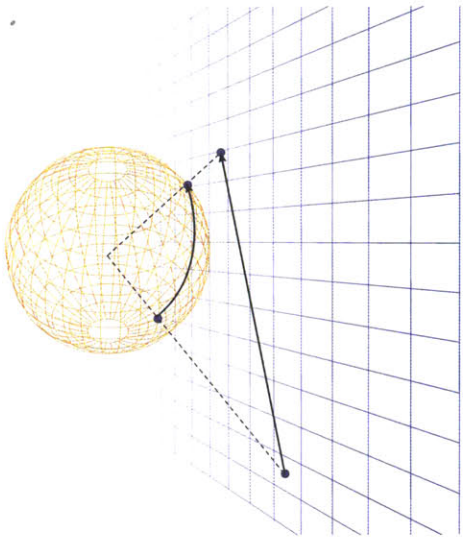

Figure 3-1: Centroidal projection of a path between two points in S^2. The Canny interpolation between orientations is a similar projection in quaternion space (S^3).

Because R' and R_0 are defined by the pair of configurations in the interpolation and constant on σ, we define $\Delta R_{i,\sigma} = R'R_0$, and we use $\gamma_{i,\sigma}$ to denote specifically the value of γ for link i on σ so that $R_x(s)$ and $R(s)$ are

$$\left(1+\gamma_{i,\sigma}^2 s^2\right) R_x(\theta) = \begin{bmatrix} 1+\gamma_{i,\sigma}^2 s^2 & 0 & 0 \\ 0 & -\gamma_{i,\sigma}^2 s^2 & -2\gamma_{i,\sigma} s \\ 0 & 2\gamma_{i,\sigma} s & -\gamma_{i,\sigma}^2 s^2 \end{bmatrix}, \tag{3.10}$$

$$\left(1+\gamma_{i,\sigma}^2 s^2\right) R_{i,\sigma}(s) = \left(1+\gamma_{i,\sigma}^2 s^2\right) R_x(s)\Delta R_{i,\sigma}. \tag{3.11}$$

The homogeneous transformation for $F_{i,\sigma}$ along σ is then given by

$$F_{i,\sigma}(s) = \begin{bmatrix} R_{i,\sigma}(s) & \mathbf{T}_{i,\sigma}(s) \\ \mathbf{0} & 1 \end{bmatrix}$$

$$\left(1+\gamma_{i,\sigma}^2 s^2\right) F_i(s) = \begin{bmatrix} \left(1+\gamma_{i,\sigma}^2 s^2\right) R_i(s) & \left(1+\gamma_{i,\sigma}^2 s^2\right) \mathbf{T}_i(s) \\ \mathbf{0} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} 1+\gamma_{i,\sigma}^2 s^2 & 0 & 0 \\ 0 & -\gamma_{i,\sigma}^2 s^2 & -2\gamma_{i,\sigma} s \\ 0 & 2\gamma_{i,\sigma} s & -\gamma_{i,\sigma}^2 s^2 \end{bmatrix} \Delta R_{i,\sigma} & \left(1+\gamma_{i,\sigma}^2 s^2\right)\left(\mathbf{T}_{i,0} + s\Delta\mathbf{T}_{i,\sigma}\right) \\ \mathbf{0} & \left(1+\gamma_{i,\sigma}^2 s^2\right) \end{bmatrix}$$

Note that each element of this matrix is a polynomial of degree at most 3. Thus, after appropriate pre-multiplication, the matrix polynomial describing the homogeneous transformation from the global frame to F_i is a polynomial of degree $3k$ where k is the number of links between the global frame and F_i.

We will see that this yields interference predicates which are univariate polynomials for the orientation of frame F_i along the path σ.

3.2.2 Constrained motions

Generally speaking, an articulated robot is composed of links which are joined in such a way as to constrain the relative motion of their reference frames. In particular, the joint between two links is most often actuated by either a linear or rotary actuator. A linear actuator allows only translational motion in a fixed direction, and a rotary actuator allows only rotation about a fixed axis. These types of constrained motions simplify the polynomial equation for F_i. If F_i is linked to F_j by a linear actuator with direction $\mathbf{v}$ in F_j, then

$$\left(1+\gamma_{i,\sigma}^2 s^2\right) F_{i,\sigma} = \begin{bmatrix} R & \left(1+\gamma_{i,\sigma}^2 s^2\right)\left(x_0 + s\Delta x\right)\mathbf{v} \\ \mathbf{0} & \left(1+\gamma_{i,\sigma}^2 s^2\right) \end{bmatrix},$$

where x_0 is the translation at the initial configuration, and $x_1 = x_0 + \Delta x$ is the translation at the final configuration. Likewise if F_i is linked to

F_j by a rotary actuator which rotates about $\mathbf{v}$ in F_j then

$$\left(1+\gamma_{i,\sigma}^2 s^2\right) F_{i,\sigma} = \begin{bmatrix} \begin{bmatrix} 1+\gamma_{i,\sigma}^2 s^2 & 0 & 0 \\ 0 & -\gamma_{i,\sigma}^2 s^2 & -2\gamma_{i,\sigma} s \\ 0 & 2\gamma_{i,\sigma} s & -\gamma_{i,\sigma}^2 s^2 \end{bmatrix} \Delta R_i & \mathbf{0} \\ \mathbf{0} & \left(1+\gamma_{i,\sigma}^2 s^2\right) \end{bmatrix}$$

where ΔR_i is fixed and does not change for different paths. We note that for rotary actuators with a range greater than π, our choice of $\Delta\theta$ as the direct rotation with magnitude always $< \pi$ may be in a direction which is not feasible for the actuator. However for two configurations we may simply test whether or is not the case, and if it $\Delta\theta$ as described above is in a direction which is infeasible for the actuator we simply choose $\Delta\theta' = \Delta\theta - 2\pi$, though we must then split the path into two intermediate configurations because $\Delta\theta'$ has magnitude greater than π.

3.3 Collision checking preliminaries

For an articulated model evolving along a path σ between two configurations generated by a Canny interpolation, we duplicate here the results of [18]. We derive the following formulations using homogeneous transformations as this leads more compact representations of motions for an armature than quaternion algebra. The development of these interference tests with quaternions is given in [18], and we note that in implementation on a digital computer with finite precision arithmetic representing rotations as quaternions generally suffers from fewer numerical precision issues.

3.3.1 Type A contact

Given a point $\mathbf{x}$ in link i of an articulated model undergoing Canny interpolated motion between two configurations parameterized by $s \in [0,1]$, and an oriented plane $(\mathbf{n}, d)$, the values of s where $\mathbf{x}$ is on the interior halfspace of the plane must satisfy

$$\underline{\mathbf{n}} \cdot \left(\prod_{j \in J} F_{j,\sigma}(s) \right) \underline{\mathbf{x}} - d \leq 0.$$

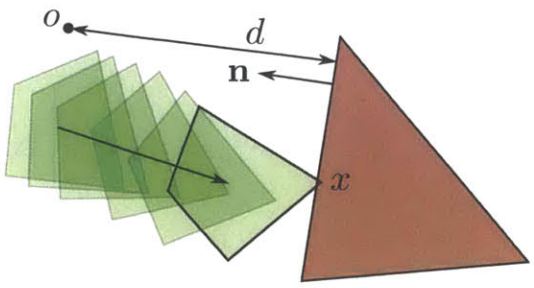

Figure 3-2: Type A contact occurs when a vertex of the moving object pierces a face of the obstacle.

Noting that by construction $\left(1+\gamma_j^2 s^2\right) > 0$ for all j, and multiplying by this factor on both sides we get

$$\underline{\mathbf{n}} \cdot \left(\prod_{j \in J} \left(1+\gamma_j^2 s^2\right) F_{j,\sigma}(s) \right) \underline{\mathbf{x}} - d \prod_{j \in J} \left(1+\gamma_j^2 s^2\right) \leq 0.$$

We define the left hand side of the inequality to be a *type A constraint*:

$$\mathcal{A}_{\mathbf{n},d,\mathbf{x},\sigma}(s) = \underline{\mathbf{n}} \cdot \left(\prod_{j \in J} \left(1+\gamma_j^2 s^2\right) F_{j,\sigma}(s) \right) \underline{\mathbf{x}} - d \prod_{j \in J} \left(1+\gamma_j^2 s^2\right). \quad (3.12)$$

$\mathcal{A}$ is a polynomial of degree $3 \cdot |J|$. The number of zero crossings of this polynomial on the interval $s \in [0,1]$ is the number of times that $\mathbf{x}$ crosses the plane on the path between configurations. By the convention that $\mathbf{n}$ is normal to the plane in the direction that is *outside* a polyhedron, Equation 3.12 will be positive when $\mathbf{x}$ is outside of the polyhedron, negative inside the polyhedron, and zero on the surface of the polyhedron.

3.3.2 Type B contact

Given an oriented plane $(\mathbf{n}, d)$ on link i, of an articulated model undergoing Canny interpolated motion between two configurations parameterized by $s \in [0,1]$, then the same oriented plane in the global frame is given by $(\mathbf{n}^g, d^g)$. We may compactly represent the face $\mathbf{n}, d$ by the *homogeneous normal* $\mathbf{N}$ which is the row concatenation of $\mathbf{n}$ with d. Then the same plane defined in the global coordinate frame is given by

$$\begin{bmatrix} \mathbf{n}^g \\ d^g \end{bmatrix} = \left(\prod_{j \in J} F_j \right) \begin{bmatrix} \mathbf{n} \\ d \end{bmatrix}.$$

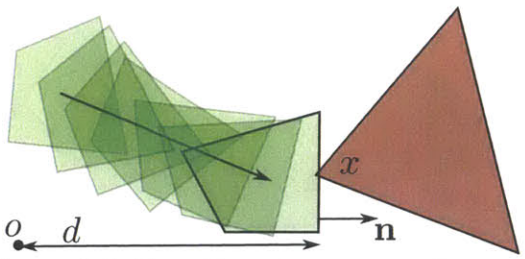

Figure 3-3: Type B contact occurs when a face of the moving object is pierced by a vertex of the obstacle.

We may also use the homogeneous representation to define the constraint for when a static point $\mathbf{x}$ in the workspace lies on the interior halfspace of a moving plane:

$$\begin{bmatrix} \mathbf{x} \\ -1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{n}^g \\ d^g \end{bmatrix} \leq 0$$

$$\begin{bmatrix} \mathbf{x} \\ -1 \end{bmatrix}^\top \left(\prod_{j \in J} F_j \right) \begin{bmatrix} \mathbf{n} \\ d \end{bmatrix} \leq 0.$$

We multiply both sides by the product of the rotation denominators and get

$$\begin{bmatrix} \mathbf{x} \\ -1 \end{bmatrix}^\top \left(\prod_{j \in J} (1 + \gamma_{j,\sigma}^2 s^2) F_j(s) \right) \begin{bmatrix} \mathbf{n} \\ d \end{bmatrix} \leq 0.$$

We define the left hand side of this inequality to be a *type B constraint*:

$$\mathcal{B}_{\mathbf{x},\mathbf{n},d,\sigma}(s) = \begin{bmatrix} \mathbf{x} \\ -1 \end{bmatrix}^\top \left(\prod_{j \in J} (1 + \gamma_{j,\sigma}^2 s^2) F_j(s) \right) \begin{bmatrix} \mathbf{n} \\ d \end{bmatrix}. \tag{3.13}$$

$\mathcal{B}$ is a polynomial of degree $3 \cdot |J|$. The number of zero crossings of this polynomial on the interval $s \in [0, 1]$ is the number of times that the plane crosses $\mathbf{x}$ on the path between configurations. And, as with type A constraints, by the convention that $\mathbf{n}$ is normal to the plane in the direction that is *outside* a polyhedron, Equation 3.13 will be positive when $\mathbf{x}$ is outside of the polyhedron, negative inside the polyhedron, and zero on the surface of the polyhedron.

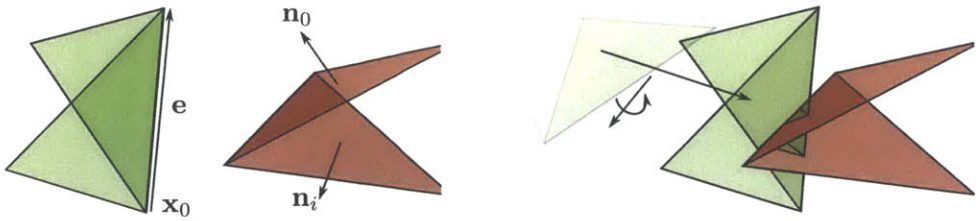

Figure 3-4: Type C contact occurs when an edge of the moving object crosses an edge of the obstacle, piercing one of its faces.

3.3.3 Type C contacts

Type C constraints are only meaningful in $\mathbb{R}^3$, as all interferences in $\mathbb{R}^2$ may be detected by type A and B constraints. An edge is the line segment formed at the intersection of two faces of a polyhedron. Let $(\mathbf{x}_0, \mathbf{x}_1)$ be two endpoints of an edge of a polyhedron represented in F_i. We may describe the line coincident to this edge as the set of points

$$\mathbf{x} = \mathbf{x}_0 + z\mathbf{e}$$

for $\mathbf{e} = \mathbf{x}_1 - \mathbf{x}_0$ and $z \in \mathbb{R}$. Clearly $\mathbf{x}$ is only on the edge if $z \in [0, 1]$. Let us now consider all edges of an obstacle which border a particular face $f_0 = (\mathbf{n}_0, d_0)$. Unless $\mathbf{e}^g$ is orthogonal to $\mathbf{n}_0$, the point on the line coincident to the moving edge intersects the plane coincident to f_0 at

$$\begin{aligned} 0 &= \mathbf{n}_0^\top \left(\mathbf{x}_0^g + z\mathbf{e}^g\right) - d_0, \\ z &= \frac{d_0 - \mathbf{n}_0^\top \mathbf{x}_0^g}{\mathbf{n}_0^\top \mathbf{e}^g}. \end{aligned}$$

This point of intersection is interior to f_0 if the constraint for all adjacent faces is satisfied. Let $f_i = (\mathbf{n}_i, d_i)$ be an adjacent face to f_0. Then

$$\mathbf{n}_i^\top \left(\mathbf{x}_0^g + \frac{d_0 - \mathbf{n}_0^\top \mathbf{x}_0^g}{\mathbf{n}_0^\top \mathbf{e}^g}\mathbf{e}^g\right) - d_i \leq 0,$$

$$\left(\mathbf{n}_0^\top \mathbf{e}^g\right)\left(\mathbf{n}_i^\top \mathbf{x}_0^g - d_i\right) - \left(\mathbf{n}_0^\top \mathbf{x}_0^g - d_0\right)\left(\mathbf{n}_i^\top \mathbf{e}^g\right) \begin{cases} \leq 0 & \left(\mathbf{n}_0^\top \mathbf{e}^g\right) \geq 0, \\ \geq 0 & \left(\mathbf{n}_0^\top \mathbf{e}^g\right) < 0. \end{cases}$$

We define the left hand side of this a *type C constraint.* Expanding $\mathbf{x}_0^g$ and $\mathbf{e}^g$ yields

$$\begin{aligned}\mathcal{C}_{e,f_0,f_i,\sigma}(s) = &\left(\mathbf{n}_0^\top \left(\prod_{j\in J}(1+\gamma_{j,\sigma}^2 s^2)R_j(s)\right)\mathbf{e}\right)\\ &\cdot\left(\underline{\mathbf{n}}_i^\top \left(\prod_{j\in J}(1+\gamma_{j,\sigma}^2 s^2)F_j(s)\right)\underline{\mathbf{x}}_0 - d_i\right)\\ &-\left(\mathbf{n}_i^\top \left(\prod_{j\in J}(1+\gamma_{j,\sigma}^2 s^2)R_j(s)\right)\mathbf{e}\right)\\ &\cdot\left(\underline{\mathbf{n}}_0^\top \left(\prod_{j\in J}(1+\gamma_{j,\sigma}^2 s^2)F_j(s)\right)\underline{\mathbf{x}}_0 - d_0\right).\end{aligned} \tag{3.14}$$

Type C constraints require a bit more care in their use. In particular, the sign of the inequality depends on the sign of $\left(\mathbf{n}_0^\top \mathbf{e}^g\right)$. Thus, the point at the intersection of the line coincident to the moving edge e and the plane coincident to the face f_0 lies on the interior of the f_0 if and only if, for all faces f_i adjacent to f_0

$$\operatorname{sgn}\left(\mathcal{C}_{e,f_0,f_i,\sigma}(s)\right) = -\operatorname{sgn}\left(\mathbf{n}_0^\top \left(\prod_{j\in J}(1+\gamma_{j,\sigma}^2 s^2)R_j(s)\right)\mathbf{e}\right).$$

We denote the right hand side as

$$\mathcal{C'}_{e,f_0,f_i,\sigma}(s) = \mathbf{n}_0^\top \left(\prod_{j\in J}(1+\gamma_{j,\sigma}^2 s^2)R_j(s)\right)\mathbf{e},$$

such that the type C constraint is given by

$$\operatorname{sgn}\left(\mathcal{C}_{e,f_0,f_i,\sigma}(s)\right) = -\operatorname{sgn}\left(\mathcal{C'}_{e,f_0,f_i,\sigma}(s)\right). \tag{3.15}$$

Note that $\mathcal{C}$ is only zero at a vertex or when the moving edge is normal to both faces of the static edge.

3.4 Interference checking

In continuous collision checking, it is common practice to divide the problem into multiple phases (see e.g. [34, 84, 107, 73]). Each phase in the so-called *collision checking pipeline* is designed to reduce the workload of subsequent phases. At the lowest level (the terminal phase in a pipeline, often called the *narrow phase*) is the interference checker.

Given a collision volume C on link i of an articulated model evolving

on a path σ, and an obstacle O, the *narrow phase algorithm* performs fine-grain continuous collision checking and determines if $C^g(s) \cap O \neq \emptyset$ for some s along the subinterval of interpolation $s \in [s_0, s_1]$. In this section we review the procedure of interference checking which provides the tools we will use in workspace certificate checking.

3.4.1 The narrow phase algorithm

Given a collision volume C in link i on an articulated model evolving according to a Canny interpolated path σ through configuration space, we wish to determine if C collides with a static obstacle O. We say that C collides with O if there exists $s \in [s_0, s_1]$ such that the intersection $C^g(s) \cap O \neq \emptyset$. $C^g(s)$ denotes the geometry of the collision volume represented in the global frame at interpolation parameters $s \in [s_0, s_1]$. We assume that at s_0 the collision volume does not intersect the obstacle, i.e., $C^g(s_0) \cap O = \emptyset$.

A collision may occur in one of three ways.

1. A vertex of C lies inside O,
2. A vertex of O lies strictly inside C,
3. An edge of C pierces a face of O,
4. An edge of O pierces a face of C.

A type 1 collision occurs for a vertex $\mathbf{v}$ of C if there is a value of s such that the type A constraint (Equation 3.12) is satisfied for every face of O. A type 2 collision occurs for a vertex $\mathbf{v}$ of O if there is a value of s such that the type B constraint (Equation 3.13) is satisfied for every face of C.

Types 3 and 4 collision is slightly more complex, however there is a symmetry that may be exploited. If an edge of O pierces a face of C, then either one endpoint of that edge lies inside C (a type 2 collision), or else an edge of C also pierces a face of O. A type 4 collision will be detected if all collisions of type 2 and 3 are detected.

A type 3 collision may be detected by evaluation of type A and type C constraints. For a moving edge e and a static face f_0, if the type C constraint for all faces adjacent f_0 are satisfied, then there is a point on the line coincident to the edge which is interior to f_0. If the type B constraint for one endpoint of the moving edge and f_0 is satisfied, and the type B constraint for the other endpoint is not satisfied, then this

point on the line coincident to the edge and interior to f_0 also lies on the edge, and thus $e \cap f_0 \neq \emptyset$ and the moving edge pierces the static face. We determine whether contact occurs by evaluating all possible cases of type 1, 2, and 3 contacts.

Each of these contacts is determined by conditions on sign of several polynomials. The theory of Sturm sequences allows us a convenient test which may indicate if, for a particular segment of the interpolation parameter, there is guaranteed to be a point of contact, or there is guaranteed to be no point of contact. The test may be inconclusive, which yields a natural bisection search algorithm for the narrow phase. Let $\texttt{Predicate}_t$ be the procedure for the test, with the output being one of `contact`, `no-contact`, `inconclusive`. Thus for an interval $[s_0, s_1]$ the narrow phase algorithm proceeds as in Algorithm 3.1.

```
1  if d > d_max then
2  |  return inconclusive
3  else
4  |  r ← Predicate_t([s_0, s_1])
5  |  if r = inconclusive then
6  |  |  r_0 ← EvaluateInterval_t([(s_0+s_1/2), s_1], d + 1)
7  |  |  r_1 ← EvaluateInterval_t([s_0, (s_0+s_1/2)], d + 1)
8  |  |  if r_0 = contact || r_1 = contact then
9  |  |  |  return contact
10 |  |  else if r_0 = inconclusive || r_1 = inconclusive
   |  |  then
11 |  |  |  return inconclusive
12 |  |  else
13 |  |  |  return no-contact
14 |  else
15 |  |  return r
```

Algorithm 3.1: `EvaluateInterval` ($\texttt{Predicate}_t, [s_0, s_1], d = 0$)

Note the use of a maximum recursion depth $d_{\max}$. If any test is inconclusive at a depth of d, then there may or may not be a collision on some subinterval of size $\Delta s/2^d$. The maximum range of s is 1 so $\Delta s = s_1 - s_0 < 1$. Thus, for an appropriate choice of $d_{\max}$ in the context of a sampling-based planner, we are effectively choosing a threshold for very small motions at which, if the test is inconclusive, we may simply choose to consider the path in collision, and not add the path to our graph. If the test is inconclusive, it means that there are multiple zero crossings of some set of constraints for a small range of s. Even if the the path is in fact collision-free, this implies that a small change of the path

could make it collide, and thus it is "close" to the obstacle. For asymptotically optimal planning algorithms, as the number of samples grows, the expected distance between any two configurations will shrink, so this threshold will also shrink. We may choose $d_{\max} = \infty$ if we wish to detect all collision free paths with certainty, but in implementation it provides an opportunity for increasing the computational efficiency.

3.4.2 Types 1 and 2 predicates

The predicate for type 1 contact between a vertex $\mathbf{v} \in C$ and O, given in Algorithm 3.2, returns `contact` if there is an $s \in [s_0, s_1]$ where the vertex is entirely contained by O, or `no-contact` if there is face such that, for all $s \in [s_0, s_1]$, the constraint is not satisfied. These two facts may be determined by the sign of the constraint equation at the start of the interval, and the number of zero crossings during the interval.

Thus for a vertex $\mathbf{v} \in C$ we generate a Sturm sequence of the constraint equation $\mathcal{A}_{\mathbf{v},\mathbf{n}_j,d_j,\sigma}(s)$ for each face $f_j \in \texttt{Faces}(O)$. Let S denote a Sturm sequence and$\mathcal{S}$ be the set of these Sturm sequences. The type 1 predicate then takes a set of Sturm sequences $\mathcal{S}$ and an interval $[s_0, s_1]$ and performs the following test to potentially determine if the path encounters a contact or not.

First, if there is any constraint j which is satisfied at the start of the interval, $p_{0,j}(s_0) = \mathcal{A}_{\mathbf{v},\mathbf{n}_j,d_j,\sigma}(s) \leq 0$, and has no zero crossings $\sigma_j(s_1) - \sigma_j s_0$ then that constraint is satisfied for the entire interval, meaning that the vertex lies on the interior halfspace of that face for the entire subpath of σ, so we may remove it from further consideration.

If all the constraints are satisfied at either the start or end of the interval, i.e., $\forall j, p_{0,j}(s_0) = \mathcal{A}_{\mathbf{v},\mathbf{n}_j,d_j,\sigma}(s_0) \leq 0$ or $\forall j, p_{0,j}(s_1) = \mathcal{A}_{\mathbf{v},\mathbf{n}_j,d_j,\sigma}(s_1) \leq 0$ then the vertex is contact with the obstacle at s_0 or s_1 respectively, and the predicate returns `contact`.

If there is only one constraint under consideration, i.e., all other constraints have been removed from consideration because they are satisfied on the entire interval, and it has at least one zero crossing, i.e., $\sigma(s_1) - \sigma(s_0) > 0$, then the vertex enters the obstacle at the zero crossing, and the predicate returns `contact`.

If there is any constraint which is not satisfied at the start of the interval and has no zero crossings on the interval, then that constraint is left unsatisfied for the entire interval, and there can be no contact on

the interval, so the predicate returns `no-contact`.

If none of these conditions are satisfied then there are multiple zero crossings of multiple constraints which may or may not be interleaved in such a way as to make contact, so the predicate is indeterminate and returns `inconclusive`, indicating that the interval is to be subdivided and retested.

```
1  leftContact ← true
2  rightContact ← true
3  foreach P ∈ S do
4      if p_0(s_0) ≤ 0 then
5          if σ_P(s_1) − σ_P(s_0) = 0 then
6              S ← S \ P
7      else
8          leftContact ← false
9          if σ_P(s_1) − σ_P(s_0) = 0 then
10             return no-contact
11     if p_0(s_1) > 0 then
12         rightContact ← false
13 if leftContact || rightContact then
14     return contact
15 if card(S) = 1 then
16     if σ_P(s_1) − σ_P(s_0) > 0 then
17         return contact
18 return inconclusive
```

Algorithm 3.2: $\texttt{Predicate}_{1,2}(\mathcal{S}, [s_0, s_1])$

The predicate for type 2 contact between a vertex $\mathbf{v} \in O$ and the faces of C is the same as type 1 contact (Algorithm 3.2) but the Sturm sequences are generated from the type B constraints, $\mathcal{B}(s)$.

3.4.3 Type 3 predicate

The predicate for type 3 contact requires a bit more effort. The type C constraints $\mathcal{C}$ and $\mathcal{C}'$ indicate when the intersection point of the line coincident to the moving edge and the face in question is interior to the face. The type A constraint $\mathcal{A}$ indicates when the endpoints are on the inside and outside of the face. Thus for a given edge e and face f_0 we enumerate the adjacent faces f_j, and generate the Sturm sequences $P_j = \texttt{SturmSeq}(\mathcal{C})$, $P'_j = \texttt{SturmSeq}(\mathcal{C}')$. Then we generate the Sturm sequences $P_0 = \texttt{SturmSeq}(\mathcal{A}_0)$ and $P_1 = \texttt{SturmSeq}(\mathcal{A}_1)$ for the two endpoints of the moving edge. Let $\mathcal{S}$ be the set of Sturm sequence pairs

(P_j, P'_j) and the pair (P_0, P_1). The type 3 predicate then takes as input $\mathcal{S}$ and an interval $[s_0, s_1]$ and performs the following tests to potentially determine if the path encounters a contact or not.

As with type 1 and two 2 predicates, if there is any constraint pair which is satisfied for the entire interval, $\operatorname{sgn}(p_{j,0}(s_0)) = -\operatorname{sgn}\left(p'_{j,0}(s_0)\right)$, $\sigma_{j,0}(s_1) - \sigma_{j,0}(s_0) = 0$, and $\sigma'_{j,0}(s_1) - \sigma'_{j,0}(s_0) = 0$ then we may remove it from further consideration.

If there is only one constraint pair under consideration, i.e., all other constraints have been removed from consideration because they are satisfied on the entire interval, and this remaining constraint pair has at least one zero crossing, i.e., $\sigma_{j,0}(s_1) - \sigma_{j,0}(s_0) > 0$, or $\sigma'_{j,0}(s_1) - \sigma'_{j,0}(s_0) > 0$, then the edge pierces the face at the zero crossing, and the predicate returns `contact`, so long as the zero crossings for both constraints in the pair are not at the same value of s. For the type C constraint pair, both constraints in the pair may have a zero crossing at the same value of s only if the edge crosses at an orientation so that it is parallel to the face. In this case the edge must also pierce some other face at an orientation which is not parallel to that face, so we may safely disregard this case. For the endpoint constraint pair, both constraints in the pair may have a zero crossing at the same value of s only if both endpoints cross the same face at s. This is the same as the previous case and either the edge pierces some other face at s, or one of the endpoints enters the obstacle at s (a type 1 contact) and so we may safely disregard this case as well.

If there is any constraint pair constraint which is not satisfied for the entire interval $\operatorname{sgn}(p_{j,0}(s_0)) = \operatorname{sgn}\left(p'_{j,0}(s_0)\right)$, $\sigma_{j,0}(s_1) - \sigma_{j,0}(s_0) = 0$, $\sigma'_{j,0}(s_1) - \sigma'_{j,0}(s_0) = 0$, then there can be no contact and the predicate returns `no-contact`.

If none of these conditions are satisfied then there are multiple zero crossings of multiple constraints which may or may not be interleaved in such a way as to make contact, so the predicate is indeterminate and returns `inconclusive`, indicating that the interval is to be subdivided and retested.

3.4.4 Implementation notes

There are a couple of opportunities for information reuse in these predicates. Because the constraint equations do not change in calls to the

```
1  leftContact ← true
2  rightContact ← true
3  foreach (P, P') ∈ S do
4      if sgn(p_0(s_0)) = −sgn(p'_0(s_0)) then
5          if σ_P(s_1) − σ_P(s_0) = 0 & σ'_P(s_1) − σ'_P(s_0) = 0
           then
6              S ← S \ P
7      else
8          leftContact ← false
9          if σ_P(s_1) − σ_P(s_0) = 0 & σ'_P(s_1) − σ'_P(s_0) = 0
           then
10             return no-contact
11     if sgn(p_0(s_0)) ≠ −sgn(p'_0(s_0)) then
12         rightContact ← false
13 if leftContact || rightContact then
14     return contact
15 if card(S) = 1 then
16     if σ_P(s_1) − σ_P(s_0) > 0 || σ'_P(s_1) − σ'_P(s_0) > 0 then
17         return contact
18 return inconclusive
```

Algorithm 3.3: $\texttt{Predicate}_3\,(\mathcal{S}, [s_0, s_1])$

predicate, but only the interval over which they are evaluated, the Sturm sequences may be generated at once at the start of the narrow phase, and then be reused in all iterations of the bisection. In addition, when an interval is subdivided, any constraints that have been removed from consideration, may be ignored in evaluating the subintervals.

In addition, this narrow phase procedure is a brute force expansion of all possible feature intersections. An alternative approach may consider only the nearest pair of features between C and O, as in [61]. In this case only Sturm sequences for the particular contact type need be generated for a particular sub interval, but an additional iteration over the sequence of nearest features is required.

3.5 Efficient collision checking with workspace certificates

Using the workspace collision tools described in this chapter, we may apply the efficient collision checking algorithm from chapter 2 without explicitly mapping collision volumes and obstacles to the configuration

space. When a new collision-free configuration is added to the planning graph, we associate with each collision volume a certificate volume which is a polyhedron in the workspace whose interior fully contains the associated collision volume.

3.5.1 Querying workspace certificates

Given a Canny interpolated path between two configurations $\mathbf{c}_a$ and $\mathbf{c}_b$ we wish to determine if the ith collision volume remains within its certificate volume for all values of the interpolation parameter $s \in [0, 1]$. Assuming that collision volumes are polytopes and certificate volumes are polyhedra in the workspace, we may restrict our focus to type A collision predicates:

Theorem 3. *Let P_0 be a collision volume undergoing a smooth motion described by the path $\sigma : [0,1] \to \mathbb{R}^n \times SO(n)$. Let P_1 be a collision certificate such that at $\sigma(0)$, $P_0 \subset P_1$. If, at some $s \in [0,1]$, $\sigma(s)$ is such that P_0 leaves P_1, $P_0 \cup P_1 \neq P_1$, then there exists some vertex $\mathbf{x}$ of P_0 and some face $\mathbf{n}, d$ of P_1 such that*

$$\underline{\mathbf{n}} \cdot \left(\prod_{j \in J} F_{j,\sigma}(s) \right) \underline{\mathbf{x}} - d > 0.$$

Proof. Assume that the converse is true, and there is no such vertex. Let $P_0 \cup P_1 \neq P_1$ at s, and let $\mathbf{x}$ be some non-vertex point $\mathbf{x} \in P_0$ and outside of P_1, $\mathbf{x} \notin P_1$. By definition of a polytope, $\mathbf{x}$ may be represented by a convex combination of the vertices of P_0. By assumption all vertices of P_0, $\mathbf{v}_j \in P_1$, so by convexity of P_1, $\mathbf{x}$ must all be in P_1, which is a contradiction. □

Theorem 3 says that if the collision volume leaves the certificate volume the first point of contact will occur with a vertex of the collision volume moving fromm the interior half space of a face of the certificate volume to the exterior half space, i.e., a type A predicate taking a zero value. In particular, a collision volume leaves a certificate volume at the minimum value of s such that any vertex of the collision volume has a positive type A value with any face of the bounding volume.

To evaluate path containment within a certificate then, we compute the sturm polynomial for the type A constraint associated with each

vertex-face pair (vertex of the collision volume, face of the certificate volume). We then compute the Sturm value for all such constraint polynomials at the start ($\sigma_i(0)$) and at the end ($\sigma_i(1)$). If $\sigma_i(1) - \sigma_i(0) > 0$ for any vertex-face pair i, then the collision volume exits the certificate volume at some point along that path. Otherwise the entire path lies within the certificate volume, and is known to be collision free.

We note that evaluating certificate containment of a path in this way is significantly more complex than for configuration space certificates which require only computing an Euclidean distance. However, this computation requires only *evaluating* a set of polynomials at two values of their argument, whereas continuous collision checking requires *solving* a set of polynomials (i.e., algorithm 3.2), once for each obstacle identified in the broadphase algorithm. Furthermore, the test is applied only to a single certificate volume, and not to a multitude of volumes as required n continuous collision checking.

3.5.2 Generating certificates from collision distances

Just as with configuration space certificates, workspace certificates can be built using the collision distance from the static collision checks. Exact distances may be computed with, i.e., Lin-Canny [61] or extended GJK [29] in conjunction with a bounding volume hierarchy. Approximate collision distances similar to those used for the previous section may also be computed as a computational optimization using GJK as in appendix A.

Let $\mathbf{c}$ be a configuration, and let $\mathbf{x}_i$ R_i, d_i be the position, orientation, and collision distance of the ith collision volume of the system at this configuration. The Minkowski sum $C_i \oplus \mathcal{B}_{d_i}$, of the ith collision volume with a ball or radius d_i yields a convex certificate whose interior is collision free, but for which path containtment is difficult to evaluate. We can select instead any polytope contained in $C_i \oplus \mathcal{B}_{d_i}$ which also contains C_i.

We may consider various certificate volume families for selecting such polytopes, trading off between complexity of the volume and spatial coverage, just as in how we select the collision volume itself. In this section we will discuss using certificates which are dilated copies of the collision volume.

We may generate a certificate volume quite easily by dilating each

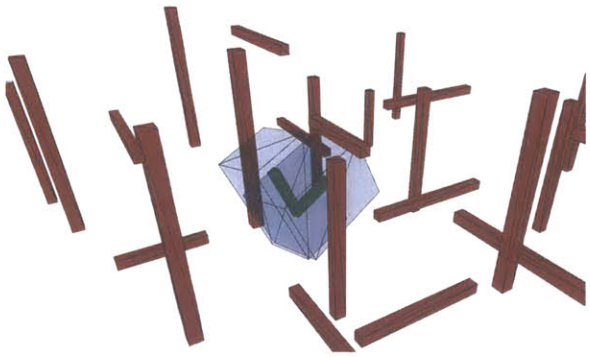

Figure 3-5: Example of a workspace certificates for an L-shaped volume found from conservative dilation of the volume faces by the collision distance. The example volume is in green, the obstacles in red, and the certificate volumes in translucent blue.

of the faces of the collision volume by a distance $D_i \leq d_i$ which is selected such that surface of the certificate volume is at most d_i from the surface of the collision volume. If two faces are adjacent with a very small angle between them (i.e., the dot product of the normals is near negative unity) then D_i may be overly conservative, so it may be beneficial to augment the face set of the collision volume with a redundant face at the meeting edge with normal half way between the the normals of the two faces (the average normal), as illustrated for a 2d volume in figure 3-6. Note that we may also simply assume such options where exploited in the modeling phase.

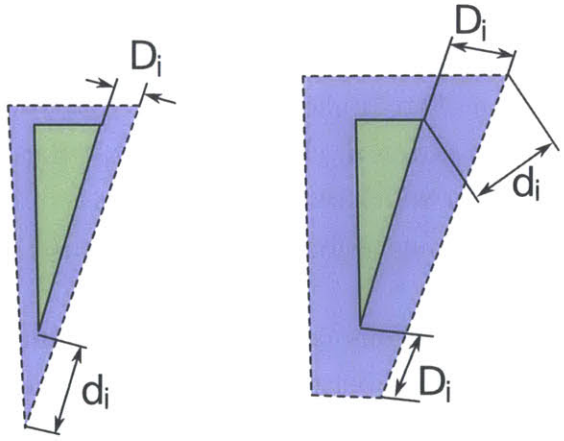

Figure 3-6: The addition of a redundant face allows for larger collision volumes.

3.5.3 Generating certificates by priority pruning

The method of computing certificates in section 3.5.2 has the advantage of requiring only collision distance output from the collision checker, the same as in chapter 2. Certficates generated in this way may, however, be overly conservative in the case that a collision volume is very close to a single obstacle, but very far from any others. In this section we propose a certificate generation algorithm which requires digging slightly deeper into the discrete collision checker. The algorithm is summarized in algorithm 3.4.

```
   input  : X_o the workspace obstacle set
   input  : P_i a workspace collision volume
1  initialize P_c = R^n
2  while X_o ≠ ∅ do
3  |  x ← Nearest(X_o, P)
4  |  if x ∈ P then
5  |  |  return in-collision
6  |  (n, d) ← SupportingHyperplane(P, x)
7  |  P_c ← P_c ∩ {x | n · x ≤ d}
8  |  X_o ← X_o ∩ {x | n · x ≤ d}
9  return P_c
```

Algorithm 3.4: `PruneCertificate`(X_o, P) returns *in-collision* if $P \cap X_o \neq \emptyset$. Otherwise returns a polyhedron P_c which is collision free and fully contains P.

Algorithm 3.4 builds a polyhedral certificate P_c by first finding the nearest obstacle to the obstacle set. The algorithm returns the point $\mathbf{x} \in X_o$ which is nearest to the collision volume. The supporting hyperplane at $\mathbf{x}$ is a hyperplane coincident to $\mathbf{x}$ and oriented so that the normal $\mathbf{n}$ points from $\mathbf{x}$ toward a nearest point in P to $\mathbf{x}$[29]. This hyperplane is added as a face of P_c, and the obstacle set is pruned to remove obstacle points on the outer halfpace of the supporting hyperplane. The algorithm is then repeated on the remaining obstacle set until all obstacles lie outside of P_c.

Note that this certificate generating static collision algorithm performs even more work than a collision-distance algorithm (which would terminate after finding the first nearest obstacle). The time complexity is now output sensitive in the number of faces of P_c, which will depend on the distribution of obstacles around the collision volume. The trade-

off being made, however, is that the collision certificate does not suffer from the same level of conservatism as the collision-distance method.

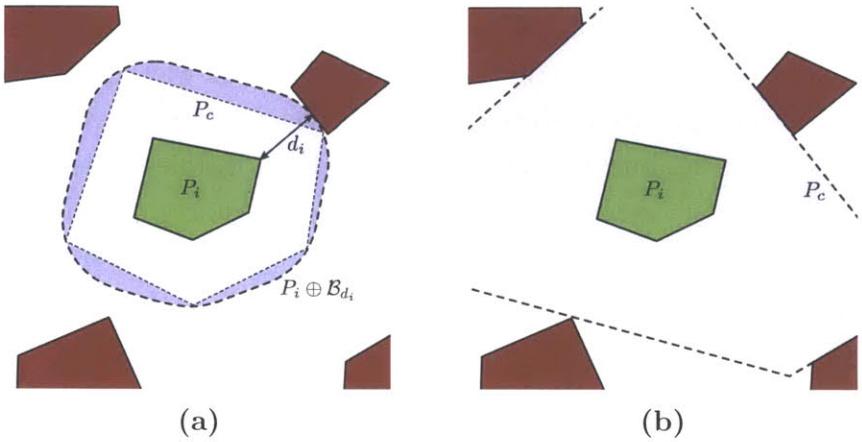

Figure 3-7: Example of 2d certificates generated using (a) dilated collision volume and collision distance and (b) successive pruning by supporting hyperplane of nearest obstacle

3.6 Experimental results

In order to demonstrate the utility of workspace certificates and their ability to accelerate sampling-based planning algorithms we performed experiments on the 3d piano mover problem, as illustrated in figure 3-5. The moving object is an L-shaped volume, decomposed into two convex collision volumes. The planning algorithm is PRM* [45], collision checking is done with FCL [73], and proximity queries are done with brute force on a GPU. The cost of a path segment between two configurations $(\mathbf{x}_a, R_a)$ and $(\mathbf{x}_b, R_b)$ is taken to be $\|\mathbf{x}_b - \mathbf{x}_a\| + w \log\left(R_a^\top R_b\right)$. We use the simple certificate volumes which are dilated collision volumes by the collision distance found during rejection sampling. The certificate checking code computes the $n \times m$ vertex-face constraint polynomials, and evaluates them all simultaneously on a GPU, leading to very low certificate checking overhead. The results presented in this section are averaged over 20 trials with different random seeds for the sampling process.

Figure 3-8 shows the wall-clock total normalized runtime (runtime divided by number of points) with and without the use of a workspace certificate cache, and when the obstacle set is spaced far apart (inter-obstacle distances 3x the length of the moving object). We see that

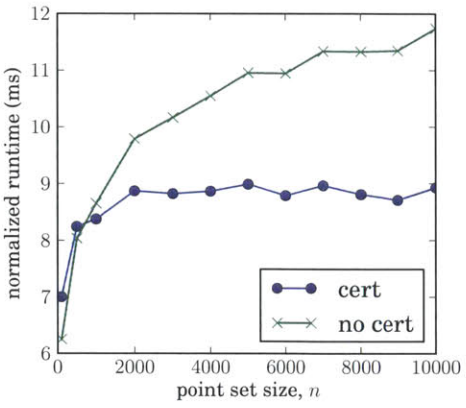

Figure 3-8: Using workspace certificates accelerates continuous collision checking by up to 40%, resulting in a 40% total runtime reduction

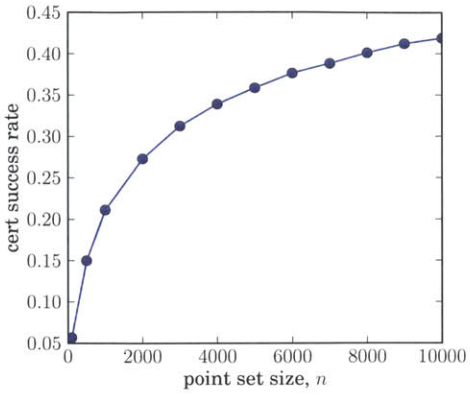

Figure 3-9: The certificate success rate grows with the size of the point set, as the inter-configuration distance is shrinking.

the use of the certificate cache reduces the total runtime between 20% and 40% depending on the number of points. This performance increase comes from reducing the time to perform continuous collision queries for path segments. Building the cache, however, requires an increase in the cost of static collision checking each of the samples during the sampling phase, performing a collision-distance query instead of a collision-only query. This increase is from about 0.05ms to about 0.25ms (5X), however, because the static collision queries are such a small part of the overall runtime, this overhead is amortized by the acceleration of the continuous checking resulting in an overall performance boost for the planning algorithm.

The benefits of the certificate cache exhibit a trend of increasing reward. As the number of samples increases, the inter-sample configuration distance is reduced, and so the likelihood that the path between two samples lies entirely within the certificate increases. Figure 3-9 shows the cache hit-rate for different point set sizes and demonstrates this increase for more samples.

Because of the anytime property of asymptotically optimal sampling-based algorithms, such as PRM*, we can see the end-to-end benefit of using the certificate cache. Depending on the planning time budget, we observe a 5s to 10s improvement in the time to find a solution of a given cost.

The symmetrically dilated collision certificate, while efficient to compute, is a rather conservative choice. When one face of the collision

volume is close to an obstacle at a particular configuration, then all of the faces are dilated by that short distance. When the obstacles are rather tightly packed, this conservatism can greatly reduce the utility of the certificate cache. Figure 3-10 illustrates the that runtime of the cache-enabled implementation is only marginally improved over the baseline implementation when obstacles are tightly packed in this 3d piano mover example, and when using conservatively dilated certificates. We can see in figure 3-11 that the conservatism leads to a very low cache hit-rate in this case.

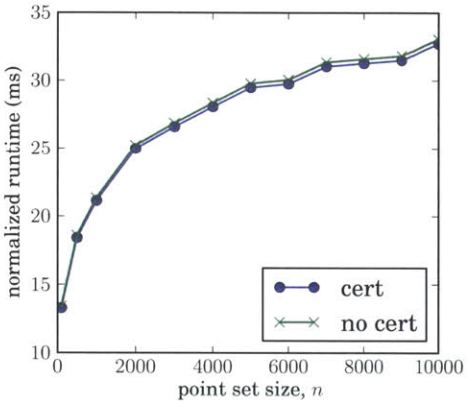

Figure 3-10: When obstacles are close together ($< 1\times$ the length of the moving object) the benefit of the collision cache is greatly reduced.

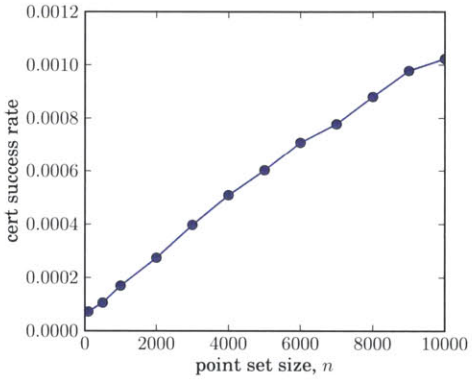

Figure 3-11: The likelihood of the path to neighboring configurations lying within a sample's certificate volume remains small for even relatively large point set sizes.

In this case, the higher coverage afforded by certificates generated from successive pruning (algorithm 3.4) provides a more appropriate collision cache. Figure 3-12 shows the runtime of the cache accelerated algorithm for the closely-packed obstacle case when certificates are generated with this method. We recover the runtime improvement of the widely-spaced experiment. The trade-off between the two certificate generating methods is evident in figure 3-13. The increased cost of generating the certificates is offset by the higher cache hit rate, reaching 30% after 10,000 samples.

3.7 Conclusions

The use of the workspace certificate cache is closely related to several common continuous collision checking algorithms based on conservative

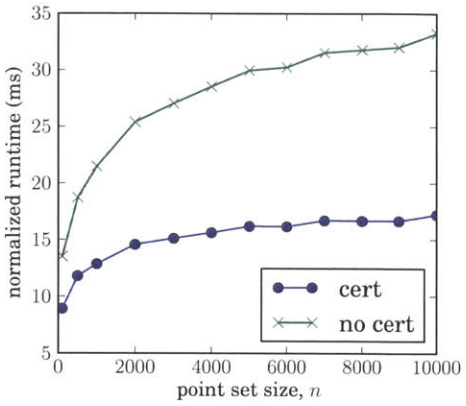

Figure 3-12: Caching of certificates generated by successive pruning yields significantly improved runtime when obstacles are close together with respect to the size of the collision volume.

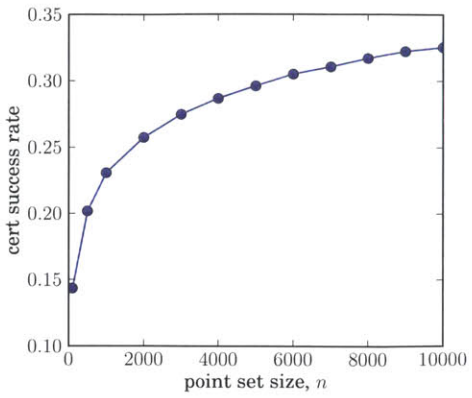

Figure 3-13: Certificates generated by successive pruning provide a cache with a higher cache rate when obstacles are close together.

advancement such as kinetic data structures [50] and adaptive bisection [85, 93]. These algorithms determine if a path is collision free by incrementally building a workspace certificate for some configuration in the path, removing the segment of the path which is certified collision free, and then recursing on the remaining uncertified paths. Because sampling-based planning algorithms often consider multiple connections from a single source configuration, the same initial certificate may be used for all paths from the source configuration. Furthermore, since the the source configuration is statically collision checked during the sampling phase, this initial certificate can be computed up front, prior to any continuous collision checking. The key insight, however, is the fact that increased sampling resolution decreases inter-configuration distances. When the configuration set is large enough, all candidates for connection will be sufficiently close to the source configuration that the entire path segment can be certified by this single workspace certificate.

The experimental results demonstrate the potential for significant runtime improvement using the workspace certificate cache. This is largely due to the fact that single certificate checking yields straightforward parallel implementations on modern hardware, meaning that even if success rate is zero the additional cost of checking the initial certificate is marginal at worst. If the continuous collision checking algorithm is in fact a variant of conservative advancement, the overhead merged into the continuous collision checking procedure. Furthermore, use of

a collision cache exhibits increasing reward, so its use is particularly advantageous in long running implementations.

The experiments also show that there is significant room for improvement in parameterizing the collision volumes. Symmetrically dilated volumes have the advantage of requiring only one additional scalar of storage per configuration, but can be overly conservative, especially in dense obstacle environments. A particular area of future work that could greatly benefit applications of this algorithm would be designing efficient static collision checking proceedures which emit whole volumes rather than simply the collision distance. If such procedures can be designed in a way which does not incur too large an overhead, the effectiveness of the collision cache can be greatly improved.

Chapter 4

Free-configuration Biased Sampling

Sampling-based algorithms avoid the construction of an explicit free-space map often generally relying on rejection sampling for generating new configurations (i.e., $\mathbf{x}_{\text{sample}}$) from the free space [56]. In rejection sampling a point is first drawn from the configuration space[1] and then statically collision checked. If the point is in collision, then it is discarded and a new point is drawn. This continues until a collision-free point is found.

Once a collision-free sample is found, it is immediately submitted to a proximity query in order to discover candidates for graph connection. There are many data structures which are used for these proximity queries, most notably spatial indices like bounding volume hierarchies (e.g., *kd*-trees and quad-trees), and tessellations (e.g., triangulations and Voronoi diagrams) [90].

In the algorithm from chapters 2 and 3 we take advantage of the spatial correlation of collision properties by caching collision free volumes along with sites in a proximity search structure. Building a certificate cache which reduces the collision checking time of new samples and paths which are in fact collision-free requires very little overhead because proximate neighbors are already indexed for finding candidate connections. We showed, however, that achieving the same acceleration of collision checking for samples and paths that are in collision requires storing additional data that would not otherwise have been stored.

[1] Often either from a uniform distribution or deterministically (for example, using a Halton sequence or a space filling curve).

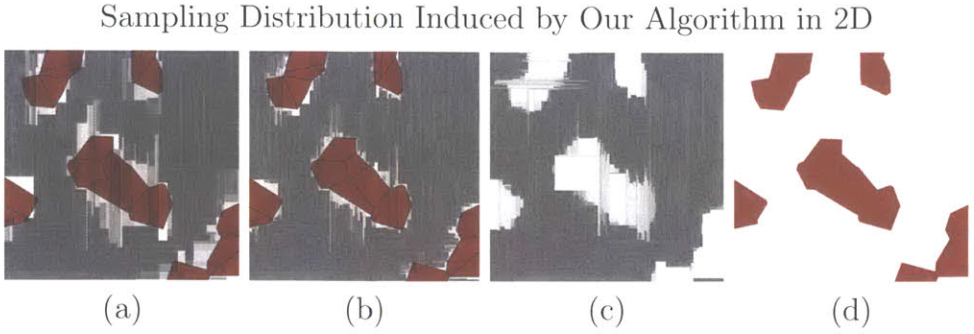

Figure 4-1: The induced sampling distribution of an augmented *kd*-tree after 10^4, 10^5, and 10^6 samples are shown in (a), (b), and (c), respectively. White-black represent low-high sampling probability density. The actual obstacle configuration appears in (d), obstacles are red.

Furthermore, while the cache accelerates the process of determining whether or not a particular point or path is in collision, it does nothing to reduce the frequency with which we must reject a new sample.

Our key insight is that many spatial indices passively encode information about the location of obstacles within the configuration space; by exposing and exploiting this information we can significantly reduce the number of rejected (i.e., *unnecessary*) future samples. In particular, by augmenting the index to record empirical collision information, it is possible to generate samples from a distribution that converges to uniform sampling over the *free space*, while *simultaneously* finding connection candidates for the new sample. In exchange for reducing the number of rejected samples, we accept a constant factor increase in the space/memory complexity of our spatial index, and sacrifice independence of consecutive samples.

4.1 Previous Work

In the context of sampling-based motion planing, a variety of techniques have been proposed to increase the chances of sampling from within narrow corridors [43, 41, 89], goal regions [32], or other particular regions of the configuration space [2, 105, 13]. Surveys of these methods can be found in [28], [62], and [91]. In contrast to our work, none are concerned with sampling uniformly from the free space.

Our idea can be described as an *adaptive sampling algorithm* as defined in [62] (i.e., a sampling algorithm that changes its sampling distribution as it runs, in response to new information that is gathered

and/or other stimuli). A canonical algorithm for adaptive sampling from a univariate distribution that learns the envelope and squeezing functions is presented in [30]. The main differences between [30] and our work is that we consider a multivariate distribution with an arbitrarily large number of variables (i.e., dimensions), and we focus on the problem of uniform sampling from an unknown (but discoverable) subspace of interest.

Related work in adaptive sampling for sample based motion planning follows. The randomized planning algorithm in [42] generatesnew samples such that the chance of sampling a particular point is inversely proportional to the local density of previous samples near that point. The planner in [91] is similar to [42], except that it samples less densely in open regions of the configuration space and more densely in cluttered regions. Tha planner in [78] weights the probability of sampling from a particular region based on the properties of the nearest node as follows: the weight is inversely proportional to a function of the nearby node's A* cost and its number of graph-neighbors, and proportional to a function of its order in the sample sequence. A later modification [79] additionally weights new samples inversely proportional to a function of the local density of previous samples. The main difference between all of these methods and our idea is that they focus on generating samples from sparsely sampled portions of the configuration space—a practice that is actually expected to *decrease* the probability a new point is sampled from the free space[2]. In contrast, we are interested in producing the opposite behavior—as more and more samples are generated, our algorithm has an *increasing* probability of sampling from the free space.

Finally, [72] is arguably related to our idea because it records statistics of total successful vs. unsuccessful samples (although it uses this data to test a stopping criterion that results in probably more than a user defined proportion of the free-space being explored). While we propose recording similar statistics in the spatial index structure, we use this data to guide future sampling instead.

[2]The rejection of obstacle space points upon collision detection results in free space regions becoming increasingly populated with old samples (e.g., relative to obstacle space), and hence the sparsely populated obstacle space is sampled increasingly frequently.

4.2 Algorithm

In this section we illustrate the application of our method by describing the sampling and search algorithm for an augmented *kd*-tree. The algorithm presented in this section may be used as a guide to implementing the method for other hierarchical spatial data structures where nodes at any depth form a tessellation of the indexed space.

Our method relies on storing extra data in each node of a *kd*-tree. A *kd*-tree is a special type of binary search tree that can efficiently determine the nearest neighbor(s) of a query point $\mathbf{x}_q$ within a previously defined set of points $\mathbf{X} \subset \mathbb{R}^d$ [9]. Each node in the *kd*-tree is a `Node` data structure the fields of which are such that the operations summarized in table 4.1 are all constant time.

Each interior node in a *kd*-tree defines an axis-aligned hyper-rectangle $H \subset \mathbb{R}^d$. It forms the root of a subtree where all descendant nodes are a strict subset of H. An interior node is assigned a point $\mathbf{x} \in \mathbf{X}$ and an index $j \in \{1 \ldots d\}$. Its two children are the hyper-rectangles found by splitting H with a hyperplane aligned to the jth axis and coincident to $\mathbf{x}$. Leaf nodes are the same as interior nodes except that they have no point $\mathbf{x}$ or children (yet). Finally, `Measure` (H) returns the measure of the *total space* within H, and `SampleUniform` (Y) returns a point from a uniform distribution over a set Y.

Table 4.1: Constant type procedures for a node N

procedure	return type	description
`point` (N)	vector $\in \mathbb{R}^d$	The point associated with this node
`split` (N)	integer $\in \{1..d\}$	The dimension of the split plane
`count` (N)	scalar $\in \mathbb{R}^+$	The number of samples generated from H
`freecount` (N)	scalar $\in \mathbb{R}^+$	The number of collision free samples generated from H
`weight` (N)	scalar $\in \mathbb{R}^+$	The weight of node N in the induced distribution
`child`$_i$ (N)	node structure	Reference to the ith child of the node (two in the case of a kd-tree), or a *null* reference, $\emptyset$, if this is a leaf node
`hRect` (N)	hyper-rectangle	The hyper-rectangle covered by the node

In our augmented *kd*-tree we associate three additional fields to each node: $T =$ `count` (V), $F =$ `freecount` (V), and $M =$ `weight` (V). Both

```
1  if isLeaf(V) then
2      x ← SampleUniform(hRect(V)) ;
3      count(V) ← count(V) + 1 ;
4      r ← isCollision(x) ;
5      t ← count(V) ;
6      f ← freecount(V) ;
7      if r then
8          point(V) ← x ;
9          freecount(V) ← freecount(V) + 1 ;
10         (child_0(V), child_0(V)) ← split(V, x) ;
11         for i = {0, 1} do
12             V_c ← child_i(V) ;
13             parent(V_c) ← V ;
14             split(V_c) ← (split(V) + 1) mod d ;
15             w ← measure(v)/measure(V) ;
16             count(V_c) ← w · t;
17             freecount(V_c) ← w · f ;
18             weight(V_c) ← f/tMeasure(v);
19     return (x, r) ;
20 else
21     u ← SampleUniform([0, weight(V)]) ;
22     if u ≤ weight(child_0(V)) then
23         (x, r) ← GenerateSample(child_0(V)) ;
24     else
25         (x, r) ← GenerateSample(child_1(V)) ;
26     weight(V) ← weight(child_0(V)) + weight(child_1(V)) ;
27     return (x, r)
```

Algorithm 4.1: GenerateSample(V)

T and F are used only by leaf nodes, where T is a score derived from total number of samples taken from H, and F is a score derived number of those samples that are collision free. When a leaf node generates a new sample and splits, each child inherits a weighted version of T and F from the parent. Both values are weighted by the relative measure of the space contained in the child vs. the parent, and both account for the successful sample before weighting. M is the estimate of the measure of free space contained in H. For leaf nodes $M = {}^{F}/_{T}\ \texttt{measure}(H)$. For non-leaf nodes, M is the sum of the values of M contained in the node's children (computed recursively).

New samples are generated by the procedure shown in Algorithm 4.1 (this recursive procedure may be replaced with two loops and a stack if desired). The procedure starts at the root of the tree, and then recursively picks a child using weighted coin flip. In particular, the chance of recursing on the i-th child is calculated as $w_i / \sum_k w_k$ (lines 5-6).

Once a leaf node is reached, the sample point is drawn from a uniform distribution over that leaf's hyper-rectangle.

The sampled point and the result of the collision check are propagated back up the tree so that the statistics of each interior node in the recursion can be updated (lines 16-18). As more samples are generated by descendants of a particular node, the estimate of that node's free space improves. Thus, future sampling more accurately reflect the true distribution of free space. Formal proofs are presented in Section 4.3.

When a sampled point is added to the tree as a leaf node, the total number of samples T and the number of free samples are both initialized to 1 (as the point itself was a sample drawn from that hyper-rectangle). This results in a prior belief that the hyper-rectangle of a new node is entirely collision free, but is necessary to ensure that all children have a probability greater than 0 of generating a sample.

Lastly, we recall the algorithm for performing nearest neighbor queries[3] in a *kd*-tree: For a query point $\mathbf{x}_q$, we perform a depth first search to find the leaf node V_0 containing $\mathbf{x}_q$. During this search we push each touched node into a stack S. We initialize the nearest neighbor $\mathbf{x}_{NN}$ with the point corresponding to V_0. Then, while the stack is not empty, we pop the top node V of the stack and do the following:

- for the point $\mathbf{x}$ associated with V, if $d(\mathbf{x}, \mathbf{x}_q) < d(\mathbf{x}_{NN}, \mathbf{x}_q)$ we replace $\mathbf{x}_{NN}$ with $\mathbf{x}$.
- if the hypercube of any unsearched children contains a point $\mathbf{x}$ such that $d(\mathbf{x}, \mathbf{x}_q) < d(\mathbf{x}_{NN}, \mathbf{x}_q)$, we push that child onto the stack.

Thus, we may *simultaneously* generate a new sample and perform nearest neighbor queries by replacing the initial depth first search with Algorithm 4.1, at the same runtime complexity of doing a nearest neighbor search.

[3] k-nearest neighbor, range search, and many other proximity queries follow this same general structure.

4.3 Analysis

4.3.1 Convergence to a uniform distribution over free space

We now prove that the sampling distribution induced by our algorithm converges to a uniform distribution over the free space. To improve readability, in this section we utlize the following shorthand notation for fields of a Node data structure V: $V.\mathbf{x} = \texttt{point}(V)$, $V.P = \texttt{parent}(V)$, $V.T = \texttt{count}(V)$, $V.F = \texttt{freecount}(V)$, and $V.M = \texttt{weight}(V)$.

Let S_{O}, S_{F}, and S denote the obstacle space, free space, and total space respectively. $S = S_{\mathrm{O}} \cup S_{\mathrm{F}}$ and $S_{\mathrm{O}} \cap S_{\mathrm{F}} = \emptyset$. We use the notation $\mathbb{S}(\cdot)$ to denote the subset of space associated with a data structure element, e.g., $H = \mathbb{S}(V)$ is the hyper-rectangle of V. We also use $\mathbb{P}(\cdot)$ to denote probability, and $\lambda(\cdot)$ to denote the Lebesgue measure, e.g., $\lambda(S_{\mathrm{F}})$ is the hyper-volume of the free space. We assume that the configuration space is bounded and that the boundaries of S_{O}, S_{F}, and S have measure zero.

Let C denote the set of children of V. Each child $c_i \in C$ represents a subset of $\mathbb{S}(V)$ such that $\bigcup_i \mathbb{S}(c_i) = \mathbb{S}(V)$ and $\mathbb{S}(c_i) \cap \mathbb{S}(c_j) = \emptyset$ for all $i \neq j$. In a *kd*-tree $|C| = 2$.

Note the wording in this section is tailored to the *kd*-tree version of our algorithm; however, the analysis is generally applicable to any related data structure [4].

Let $f_n(\cdot)$ be the probability density function for the sample returned by Algorithm 4.1, when the *kd*-tree contains n points. Let $f_{\mathrm{F}}(\cdot)$ represent a probability density function such that $f_{\mathrm{F}}(x_a) = f_{\mathrm{F}}(x_b)$ for all $x_a, x_b \in S_{\mathrm{F}}$ and $f_{\mathrm{F}}(x_c) = 0$ for all $x_c \in S_{\mathrm{O}}$. Let X_{F} and X_n denote random variables drawn from the distributions defined by $f_{\mathrm{F}}(\cdot)$ and $f_n(\cdot)$, respectively.

We now prove that $\mathbb{P}(\lim_{n\to\infty} f_n(x) = f_{\mathrm{F}}(x)) = 1$, for almost all

[4] In particular, we only require a tree-based space partitioning spatial index that is theoretically capable of containing any countably infinite set of points $\mathbf{X}$, and such that the hyper-space of the leaf nodes covers the configuration space $S = \bigcup \mathbb{S}(V^{leaf})$. Our proofs can be modified to the general case by replacing 'hyper-rectangle' with 'hyper-space' and assuming that a weighted die determines the recursion path (instead of a coin). When the die is thrown at V it has $|C|$ sides and the weight of the i-th side is determined by the estimated value of $\lambda(\mathbb{S}(c_i))/\lambda(\mathbb{S}(V))$ (i.e., the relative amount of free space believed to exist in child c_i vs. its parent V).

$x \in S$, i.e., that the induced distribution of our algorithm converges to a distribution that is almost surely equal to $f_{\mathrm{F}}(x)$ almost everywhere in S, possibly excluding a measure-zero subset.

We begin by observing that the nodes in the *kd*-tree may be classified into three sets:

- free nodes, $\mathcal{V}^{\mathrm{F}}$, the set of nodes V such that $\lambda(\mathbb{S}(V) \cap S_{\mathrm{O}}) = 0$ and $\exists \mathbf{x} \,|\, \mathbf{x} \in \mathbb{S}(V) \wedge \mathbf{x} \in S_{\mathrm{F}}$,
- obstacle nodes, $\mathcal{V}^{\mathrm{O}}$, the set of nodes, V such that $\lambda\left(\mathbb{S}\left(V^{\mathrm{O}}\right) \cap S_{\mathrm{F}}\right) = 0$ and $\exists \mathbf{x} \,|\, \mathbf{x} \in \mathbb{S}(V) \wedge \mathbf{x} \in S_{\mathrm{O}}$,
- mixed nodes, $\mathcal{V}^{\mathrm{M}}$ contains all nodes do not fit the definition of a free node or a mixed node.

Although our algorithm is ignorant of the type of any given node (otherwise we would not need it to begin with), an oracle would know that free nodes contain free space almost everywhere, obstacle nodes contain obstacle space almost everywhere, and mixed nodes contain both free space and obstacle space. Note that if $V \in \mathcal{V}^{\mathrm{M}}$ and $\lambda(\mathbb{S}(V)) > 0$ then $\lambda(\mathbb{S}(V) \cap S_{\mathrm{O}}) > 0$ and $\lambda(\mathbb{S}(V) \cap S_{\mathrm{F}}) > 0$. It is possible for V such that $\lambda(\mathbb{S}(V)) = 0$ to be both an obstacle node and a free node if it exists on the the boundary between S_{O} and S_{F}; because their cumulative measure is zero, such nodes can be counted as both obstacle nodes and free nodes (or explicitly defined as either one or the other) without affecting our results.

We are particularly interested in the types of leaf nodes, because they cover S and also hold all of the mass that determines the induced sampling distribution.

Let $\mathcal{V}^{\mathrm{L}}$ denote the set of leaf nodes, and let $\mathcal{V}^{\mathrm{FL}}$, $\mathcal{V}^{\mathrm{OL}}$, $\mathcal{V}^{\mathrm{ML}}$ denote the set of leaf nodes that are also free nodes, obstacle nodes, and mixed nodes, respectively. We use $\mathbb{S}(\mathcal{V}) = \bigcup_{V \in \mathcal{V}} \mathbb{S}(V)$ to denote the space contained in all nodes in a set $\mathcal{V}$. Figure 4-2 depicts the space contained in the set of leaf nodes, $\mathcal{V}^{\mathrm{L}}$, of a particular *kd*-tree.

Recall that $V.M$ is the estimated probability mass that our algorithm associates with node V. Let D denote tree depth.

Proposition 1. $\mathbb{P}(X_n \in \mathbb{S}(V)) = V.M / \sum_{V' \in \mathcal{V}^{\mathrm{L}}} V'.M$

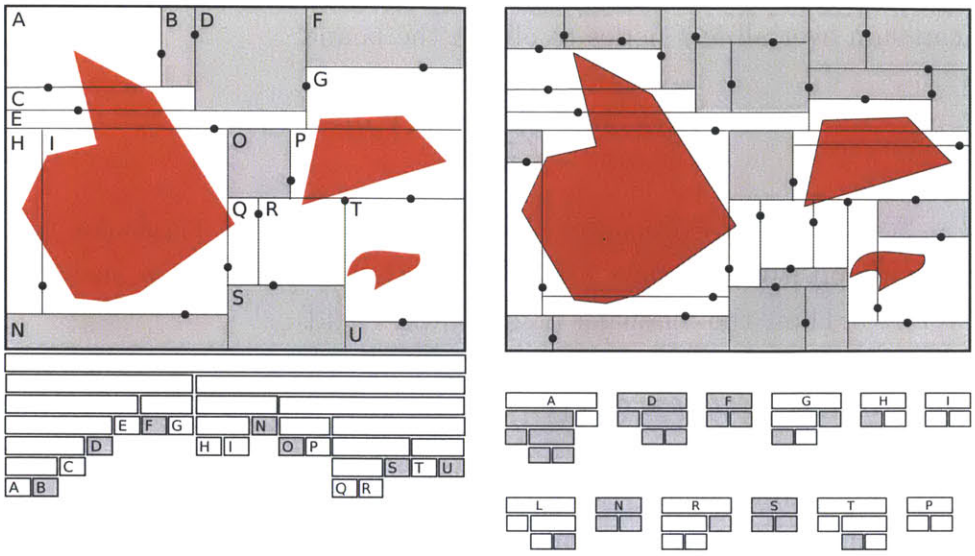

Figure 4-2: The hyper-rectangles of leaf nodes (Top) from the corresponding *kd*-trees (Bottom). Letters show the correspondence between nodes and their hyper-rectangles. Left and Right show 28 and 41 points, respectively. Obstacle space is red. Free nodes are gray and mixed nodes are white. Letters show the correspondence. Descendants of a free node are always free. Mixed nodes eventually produce free node descendants (the probability that an obstacle node is produced is 0).

Proof. This is true by the construction of our algorithm. In particular, from lines 16-17 and 21. □

Proposition 2. *For all V at depth $D > 1$ such that $\lambda(\mathbb{S}(V)) > 0$ there exists some $\delta > 0$ such that $V.F > \delta$.*

Proof. All nodes at depth $D > 1$ have a parent $V.P$, which must have generated at least one sample in order to create V. Since $\lambda(\mathbb{S}(V)) > 0$, we know that $\lambda(\mathbb{S}(V.P)) > 0$. Therefore, by construction (lines 7, 11, 13) we know $V.F \geq \frac{\lambda(\mathbb{S}(V))}{\lambda(\mathbb{S}(V.P))} > 0$. Thus, the lemma is true for δ such that $0 < \delta < \frac{\lambda(\mathbb{S}(V))}{\lambda(\mathbb{S}(V.P))}$. □

Lemma 7. *For a particular node V, let $N_n(V)$ be the number of times that a sample was generated from $\mathbb{S}(V)$ when the kd-tree has n nodes. Then, for all V such that $\lambda(\mathbb{S}(V)) > 0$, $\mathbb{P}(\lim_{n\to\infty} N_n(V) = +\infty) = 1$.*

Proof. We begin by obtaining two intermediate results:

First, $V.F \leq V.T$ for all V by construction (lines 3, 5, 7, 11, 13). Thus, for all leaf nodes $V \in \mathcal{V}^{\mathrm{L}}$ it is guaranteed $V.M \leq \lambda(\mathbb{S}(V))$ by the definition of M (line 12). Recall that the set of leaf nodes covers the space $\mathbb{S}(\mathcal{V}^{\mathrm{L}}) = S$ and that the space in each leaf node is non-overlapping $\mathbb{S}(V_i) \cap \mathbb{S}(V_j) = \emptyset$ for all $V_i, V_j \in \mathcal{V}^{\mathrm{L}}$, $V_i \neq V_j$. Thus, we

can sum over all leaf nodes to obtain the bound:

$$\sum_{V \in \mathcal{V}^{\mathrm{L}}} V.M \leq \sum_{V \in \mathcal{V}^{\mathrm{L}}} \lambda\left(\mathbb{S}\left(V\right)\right) = \lambda\left(S\right).$$

Second, using Proposition 2 we know that for any particular node V with positive measure $\lambda\left(\mathbb{S}\left(V\right)\right) > 0$ there exists some δ such that $V.F > \delta$.Thus, the following bound always holds:

$$V.M = \lambda\left(\mathbb{S}\left(V\right)\right) \frac{V.F}{V.T} \geq \lambda\left(\mathbb{S}\left(V\right)\right) \frac{\delta}{V.T}$$

(where the first equality is by definition). Note this is the *worst case* situation in which node V always samples from obstacle space (and thus V remains a leaf node forever). Thus, $V.M \geq \lambda\left(\mathbb{S}\left(V\right)\right) \frac{\delta}{V.T}$.

Combining the first and second results yields:

$$\mathbb{P}\left(X_n \in \mathbb{S}\left(V\right)\right) = \frac{V.M}{\sum_{V' \in \mathcal{V}^{\mathrm{L}}} V'.M} \geq \frac{\delta \lambda\left(\mathbb{S}\left(V\right)\right)}{V.T \lambda\left(S\right)}$$

Where the left equality is by Proposition 1. By definition $\frac{\delta \lambda(\mathbb{S}(V))}{\lambda(S)} = k$ is a constant, and so $\mathbb{P}\left(X_n \in \mathbb{S}\left(V\right)\right) \geq \frac{k}{V.T}$. By definition, $V.T$ only increase when we draw a sample from $\mathbb{S}\left(V\right)$. Let $\hat{n}$ be the iteration at which the previous sample was generated from $\mathbb{S}\left(V\right)$. The probability that we *never again* generate a sample from $\mathbb{S}\left(V\right)$ is bounded:

$$\mathbb{P}\left(\lim_{n \to \infty} N_n\left(V\right) = N_{\hat{n}}\left(V\right)\right) \leq \lim_{n \to \infty} \prod_{i=\hat{n}}^{n} \left(1 - \frac{k}{V.T}\right) = 0$$

for all $V.T < \infty$ and $N_{\hat{n}}\left(V\right) < \infty$ (and thus $\hat{n} < \infty$). The rest of the proof follows from induction. □

Lemma 8. *Let $\mathcal{V}_n^{\mathrm{F}}$ be the set of free nodes in the tree of n samples. Then, for all $\mathbf{x} \in S_{\mathrm{F}}$, $\lim_{n \to \infty} \mathbb{P}\left(\exists V \in \mathcal{V}_n^{\mathrm{F}} \mid \mathbf{x} \in \overline{\mathbb{S}}\left(V\right)\right) = 1$*

Proof. Let $\Xi_{\epsilon,\mathbf{x}}$ be the open L1-ball with radius ϵ that is centered at point $\mathbf{x}$. For all $\mathbf{x} \in \mathrm{int}(S_{\mathrm{F}})$ there exists some $\epsilon > 0$ for which $\Xi_{\epsilon,\mathbf{x}} \subset S_{\mathrm{F}}$. Therefore, it is sufficient to prove that for $\mathbf{x} \in S_{\mathrm{F}}$,

$$\lim_{n \to \infty} \mathbb{P}\left(\exists V \in \mathcal{V}_n^{\mathrm{F}} \mid V \subset \Xi_{\epsilon,\mathbf{x}}\right) = 1.$$

Without loss of generality, we now consider a particular $\mathbf{x}$. At any

point during the run of the algorithm there is some leaf node $V_{\mathrm{L}} \mid V_{\mathrm{L}} \ni \mathbf{x}$.

Lemma 7 guarantees that $V_{\mathrm{L}} \ni \mathbf{x}$ will almost surely split into two children, one of which will also contain $\mathbf{x}$, etc. Let $V_{D,\mathbf{x}}$ represent the node at depth D that contains $\mathbf{x}$. Let $X_D \in \mathbb{S}\left(V_{D,\mathbf{x}}\right)$ be the sample point that causes $V_{D,\mathbf{x}}$ to split. Let $\mathbf{x}_{\mathbf{x}} i$ refer to the i-th coordinate of $\mathbf{x}$. Thus, the splitting plane is normal to the $D \bmod d$-axis, and intersects that axis at $X_D[D \bmod d]$, where d is the dimensionality of the space. Each time the current leaf $V_{D,\mathbf{x}} \ni \mathbf{x}$ splits

$$\mathbb{P}\left(X_D \in \Xi_{\epsilon,\mathbf{x}} \wedge \mathbf{x}_{X_D} i < \mathbf{x}_{\mathbf{x}} i\right) = \frac{\lambda\left(\mathbb{S}\left(V_{D,\mathbf{x}}\right) \cap \mathbb{S}\left(\Xi_{\epsilon,\mathbf{x}}\right)\right)}{2\lambda\left(\mathbb{S}\left(V_{D,\mathbf{x}}\right) \cap S_{\mathrm{F}}\right)} > 0.$$

By construction

$$\frac{\lambda\left(\mathbb{S}\left(V_{D+d,\mathbf{x}}\right) \cap \mathbb{S}\left(\Xi_{\epsilon,\mathbf{x}}\right)\right)}{\lambda\left(\mathbb{S}\left(V_{D+d,\mathbf{x}}\right) \cap S_{\mathrm{F}}\right)} \geq \frac{\lambda\left(\mathbb{S}\left(V_{D,\mathbf{x}}\right) \cap \mathbb{S}\left(\Xi_{\epsilon,\mathbf{x}}\right)\right)}{\lambda\left(\mathbb{S}\left(V_{D,\mathbf{x}}\right) \cap S_{\mathrm{F}}\right)}$$

so

$$\lim_{D \to \infty} \mathbb{P}\left(\exists X_D \mid X_D \in \Xi_{\epsilon,\mathbf{x}} \wedge \mathbf{x}_{X_D} D \bmod d < \mathbf{x}_{\mathbf{x}} d \bmod D\right) = 1.$$

A similar argument can be made for $\mathbf{x}_X i > \mathbf{x}_{\mathbf{x}} i$,

$$\lim_{D \to \infty} \mathbb{P}\left(\exists X_D \mid X_D \in \Xi_{\epsilon,\mathbf{x}} \wedge \mathbf{x}_{X_D} D \bmod d > \mathbf{x}_{\mathbf{x}} D \bmod d\right) = 1.$$

Thus, in the limit as $D \to \infty$, there will almost surely be a set of $2d$ points $\{X_{D_1}, \ldots, X_{D_{2d}}\}$ sampled at levels $D_1, \ldots, D_{2d}$, such that $X_{D_i} \in \Xi_{\epsilon,\mathbf{x}}$ for $i = \{1, \ldots, 2d\}$, and $i = D_i \bmod d$ and $\mathbf{x}_{X_{D_i}} i < \mathbf{x}_X i$, and $i = D_{d+i} \bmod d$ and $\mathbf{x}_{X_{D_{d+i}}} i > \mathbf{x}_X i$. By construction, $X_{\max_i(D_i)}$ is on a splitting plane that borders a node V such that $\mathbb{S}(V) \ni \mathbf{x}$ and $V \subset \Xi_{\epsilon,\mathbf{x}}$ (and thus $V \in \mathcal{V}^{\mathrm{F}}$). Lemma 7 implies that $\mathbb{P}\left(\lim_{n \to \infty} D = \infty\right) = 1$ for $V_{D,\mathbf{x}} \mid \mathbb{S}\left(V_{D,\mathbf{x}}\right) \ni \mathbf{x}$. □

Corollary 1. $\mathbb{P}\left(\lim_{n \to \infty} \lambda\left(\mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right) \cap S_{\mathrm{F}}\right) = 0\right) = 1.$

Corollary 2. $\mathbb{P}\left(\lim_{n \to \infty} \lambda\left(S_{\mathrm{F}} \setminus \mathbb{S}\left(\mathcal{V}^{\mathrm{FL}}\right)\right) = 0\right) = 1.$

Corollary 3.
$\mathbb{P}\left(\lim_{n \to \infty} \sum_{V \in \mathcal{V}^{\mathrm{FL}}} \lambda\left(\mathbb{S}\left(V\right)\right) = \lambda\left(S_{\mathrm{F}}\right)\right) = 1.$

Lemma 9. $\mathbb{P}\left(\lim_{n \to \infty} V.M = 0\right) = 1$ *for all obstacle leaf nodes* $V \in \mathcal{V}^{\mathrm{OL}}$.

Proof. There are two cases, one for $\lambda(\mathbb{S}(V)) = 0$ and another for $\lambda(\mathbb{S}(V)) > 0$. The first is immediate given $V.M \triangleq \frac{V.F}{V.T}\lambda(\mathbb{S}(V))$. For the second, we observe that $\mathbb{P}(\exists \mathbf{x} \mid \mathbf{x} \in \mathbb{S}(V) \wedge \mathbf{x} \in S_{\mathrm{F}}) = 0$ by definition, and so $V.F$ will almost surely not change (and V will remain a leaf node almost surely). Thus, $\mathbb{P}(V.T = \infty) = 1$ by Lemma 7, and so $\mathbb{P}\left(\lim_{n\to\infty} \frac{V.F}{V.T} = 0\right) = 1$. Using the definition of $V.M$ finishes the proof. □

Corollary 4. $\lim_{n\to\infty} \mathbb{P}\left(X_n \in \mathbb{S}\left(\mathcal{V}^{\mathrm{OL}}\right)\right) = 0.$

Lemma 10. $\mathbb{P}(\lim_{n\to\infty} V.M = 0) = 1$ *for all mixed leaf nodes* $V \in \mathcal{V}^{\mathrm{ML}}$.

Proof. $V.F$ will almost surely not change by Corollary 1. The rest of the proof is similar to Lemma 9. □

Corollary 5. $\lim_{n\to\infty} \mathbb{P}\left(X_n \in \mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right)\right) = 0.$

Corollary 6.
$\lim_{n\to\infty} \mathbb{P}\left(X_n \in \left(\mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right) \cup \mathbb{S}\left(\mathcal{V}^{\mathrm{OL}}\right)\right)\right) = 0$

We observe that this result does not conflict with Lemma 7. Each node with finite space is sampled an infinite number of times; however, the proportion of samples from obstacle nodes and mixed nodes approaches 0 in the limit as $n \to \infty$.

Lemma 11. $\lim_{n\to\infty} \mathbb{P}(X_n \in S_{\mathrm{O}}) = 0$

Proof. This follows from Corollary 6 and the fact that

$$\lambda\left(S_{\mathrm{O}} \setminus \left(\mathbb{S}\left(\mathcal{V}^{\mathrm{ML}}\right) \cup \mathbb{S}\left(\mathcal{V}^{\mathrm{OL}}\right)\right)\right) = 0.$$

□

Lemma 12. $\mathbb{P}(\lim_{n\to\infty} V.M = \lambda(\mathbb{S}(V))) = 1$, *for all free nodes* $V \in \mathcal{V}^{\mathrm{F}}$.

Proof. There are two cases, one for when $\lambda(\mathbb{S}(V)) = 0$ and another for when $\lambda(\mathbb{S}(V)) > 0$. The former is immediate given the definition of $V.M$, and so we focus on the latter. When a new free node $V_D \in \mathcal{V}^{\mathrm{F}}$ is created at depth $D > 1$ of the tree it initializes $V_D.F > 0$ and $V_D.T > 0$ based on similar values contained in its parent (and weighted by the relative measures of V_D vs. its parent). By Lemma 7 we know

that V_D will almost surely generate two children $V_{D+1,0}$ and $V_{D+1,1}$. By construction (lines 11-13), they will be initialized with

$$V_{D+1,j}.F = (V_D.F + 1)\frac{\lambda(\mathbb{S}(V_{D+1,j}))}{\lambda(\mathbb{S}(V_D))}$$
$$V_{D+1,j}.T = (V_D.T + 1)\frac{\lambda(\mathbb{S}(V_{D+1,j}))}{\lambda(\mathbb{S}(V_D))}$$

, for $j \in \{0, 1\}$. These children will also generate their own children almost surely, etc. Because V_D is a free node, all samples from its subtree will result in more *free node* descendants being created almost surely. Let $\hat{C}_n$ be the set containing all leaf node descendants of V_D at iteration n. By construction (line 24), as soon as $|\hat{C}_n| \geq 1$, then $V_D.M = \sum_{V \in \hat{C}_n} V.M$. We now examine a single term of the latter summation, i.e., the term for node V_{D+k} at depth $D + k$. In particular. $V_{D+k}.M = \frac{V_{D+k}.F}{V_{D+k}.T}\lambda(\mathbb{S}(V_{D+k}))$. For the remainder of this proof we will abuse our notation and let $\|\cdot\| = \lambda(\mathbb{S}(\cdot))$ to make the following equations more readable. Unrolling the recurrence relation for $V_{D+k}.F$ gives:

$$V_{D+k}.F = \frac{\|V_{D+k}\|}{\|V_{D+k-1}\|}\left(\ldots\frac{\|V_{D+2}\|}{\|V_{D+1}\|}\left(\frac{\|V_{D+1}\|}{\|V_D\|}(V_D.F + 1) + 1\right)\ldots + 1\right)$$

where $V_{D+k-1}, \ldots, V_{D+2}, V_{D+1}, V_D$, are the ancestors of V_{D+k} going up the tree to V_D. This can be rearranged:

$$V_{D+k}.F = \frac{\|V_{D+k}\|}{\|V_D\|}V_D.F + \frac{\|V_{D+k}\|}{\|V_D\|} + \frac{\|V_{D+k}\|}{\|V_{D+1}\|} + \ldots + \frac{\|V_{D+k}\|}{\|V_{D+k-1}\|}.$$

Similarly, the $V_{D+k}.T$ recurrence relation is:

$$V_{D+k}.T = \frac{\|V_{D+k}\|}{\|V_D\|}V_D.T + \frac{\|V_{D+k}\|}{\|V_D\|} + \frac{\|V_{D+k}\|}{\|V_{D+1}\|} + \ldots + \frac{\|V_{D+k}\|}{\|V_{D+k-1}\|}.$$

$\lim_{k\to\infty}\frac{\|V_{D+k}\|}{\|V_D\|} = 0$, also $\mathbb{P}\left(\frac{\|V_{D+k}\|}{\|V_{D+k-1}\|} = 0\right) = 0$ given $\lambda(\mathbb{S}(V_D)) > 0$, where we resume our normal notation. Thus, $\mathbb{P}(\lim_{k\to\infty} V_{D+k}.M = \lambda(\mathbb{S}(V_{D+k}))) = 1$.

Lemma 7 guarantees that $\mathbb{P}(\lim_{n\to\infty} k = \infty) = 1$ for all V_{D+k} such that $\hat{C}_n \ni V_{D+k}$. Thus, by summing over the members of $\hat{C}_n$ we get: $\mathbb{P}\left(\lim_{n\to\infty}\sum_{V\in\hat{C}_n} V.M = \sum_{V\in\hat{C}_n}\lambda(\mathbb{S}(V))\right) = 1$. $V_D.M = \sum_{V\in\hat{C}_n} V.M$ by definition. Also by definition $\mathbb{S}(V_D) = \bigcup_{V\in\hat{C}_n}\mathbb{S}(V)$ and $V_i \cap V_j = \emptyset$

for all $V_i, V_j \in \hat{C}_n$ such that $i \neq j$; therefore, $\lambda(\mathbb{S}(V_D)) = \sum_{V \in \hat{C}_n} \lambda(\mathbb{S}(V))$. Substitution finishes the proof. □

Note, Corollary 7 depends on Lemma 12 and Corollary 2:

Corollary 7.
$\mathbb{P}\left(\lim_{n\to\infty} \sum_{V \in \mathcal{V}^{\mathrm{FL}}} V.M = \lambda(S_{\mathrm{F}})\right) = 1.$

Lemma 13. $\mathbb{P}(\lim_{n\to\infty} f_n(x) = c) = 1$ *for all* x *and* B *such that* $x \in B \subset S_{\mathrm{F}}$ *and* $\lambda(B) > 0$

Proof. By Proposition 1 and Lemmas 12 and Corollary 7 we know that $\lim_{n\to\infty} \mathbb{P}(\mathbf{x}_n \in \mathbb{S}(V)) = \frac{\lambda(\mathbb{S}(V))}{\lambda(S_{\mathrm{F}})}$ for all free nodes $V \in \mathcal{V}^{\mathrm{F}}$ almost surely. By construction (line 2) once a leaf node $V \in \mathcal{V}^{\mathrm{FL}}$ is reached, samples are drawn uniformly from within $\mathbb{S}(V)$. Thus, the uniform probability density of drawing $\mathbf{x}_n \in \mathbb{S}(V)$, given that the algorithm has decided to draw from within $\mathbb{S}(V)$, is $f_n(\mathbf{x}_n | \mathbf{x}_n \in \mathbb{S}(V)) = \frac{1}{\lambda(\mathbb{S}(V))}$. Therefore, the posterior probability density

$$\lim_{n\to\infty} f_n(\mathbf{x}_n) = \lim_{n\to\infty} f_n(\mathbf{x}_n | \mathbf{x}_n \in \mathbb{S}(V))\, \mathbb{P}(\mathbf{x}_n \in \mathbb{S}(V)) = \frac{1}{\lambda(S_{\mathrm{F}})}$$

almost surely, which is constant and *independent* of $V \in \mathcal{V}^{\mathrm{FL}}$, and thus holds almost everywhere in $\bigcup_{V \in \mathcal{V}^{\mathrm{FL}}} \mathbb{S}(V)$—and thus almost everywhere in S_{F} (by Corollary 2). □

Theorem 4. $\mathbb{P}(\lim_{n\to\infty} f_n(x) = f_{\mathrm{F}}(x)) = 1.$

Proof. This is proved by combining Lemmas 11 and 13. □

4.3.2 Performance and runtime

Our method is designed to increase the speed at which new points are added to the search graph by eliminating the work wasted on points that will be rejected. That said, our method cannot be expected to outperform more traditional rejection sampling in every case that may be encountered. We now discuss the runtime of the *kd*-tree version of our algorithm vs. rejection sampling, in order to evaluate when it should be used.

Let c_{kd} be the work associated with performing graph operations (nearest neighbor searching, or insertion) on a standard *kd*-tree. Let c_{gen}

denote the time to generate a sample from a distribution uniform over a hyper-rectangle, and let c_{cc} denote the time required to collision check a point. Note that $c_{kd} = O(\log n)$ for a balanced kd-tree of n points and $c_{gen} = O(d)$, where d is the dimensionality of the configuration space. As compared to rejection sampling, our method changes the computation time required to generate a candidate sample from c_{gen} to $c_{gen} \cdot c_{kd}$.

Our method also changes the expected number of trials required to sample a point $\mathbf{x} \in S_{\mathrm{F}}$ from S/S_{F} to $\mathbb{E}_{trials}$, where $\mathbb{E}_{trials} = S/S_{\mathrm{F}}$ for the first sample ($n = 1$) and then $\lim_{n\to\infty} \mathbb{E}_{trials} = 1$ (i.e., $\mathbb{E}_{trials}$ approaches 1 as the number of successful samples increases).

Thus, our method changes the expected time to both find a point $\mathbf{x} \in S_{\mathrm{F}}$ and then perform graph operations on kd-tree from $(c_{gen} + c_{cc})(S/S_{\mathrm{F}}) + c_{kd}$ to $(c_{gen} \cdot c_{kd} + c_{cc})\,\mathbb{E}_{trials}$. Note that S/S_{F} is a constant (assuming a static environment). We expect our method to have a practical improvement in cases where we may reasonably expect $\mathbb{E}_{trials} < (S/S_{\mathrm{F}})(1/c_{kd})$.

4.4 Experiments and Results

In order to validate our method and profile its performance we perform several experiments for different robotic systems and obstacle sets. We present results for three systems: a planar point, a concave planar object, and a 4-link thin-arm manipulator in the plane. Figure 4-3 shows the obstacle sets used for the the planar object and the manipulator, respectively. Those used for the planar point are identical to those used for the planar object.

In each experiment we compare the results for both (1) sampling a collision-free configuration and (2) sampling a collision-free configuration *and* finding its nearest neighbor among previously sampled collision-free configurations. We compare the results for the (a) the kd-tree implementation described in Section 4.2 and (b) classical rejection sampling. We use the abbreviations KDS (1a), KDSS (2a), RS (1b) and RSS (2b). Note that (2b) is comprised of the subroutines (1b) would replace in an otherwise standard implementation of RRT, RRT*, PRM, or PRM*.

Results are averaged over thirty runs, and collision checking is performed using an axis-aligned bounding box tree.

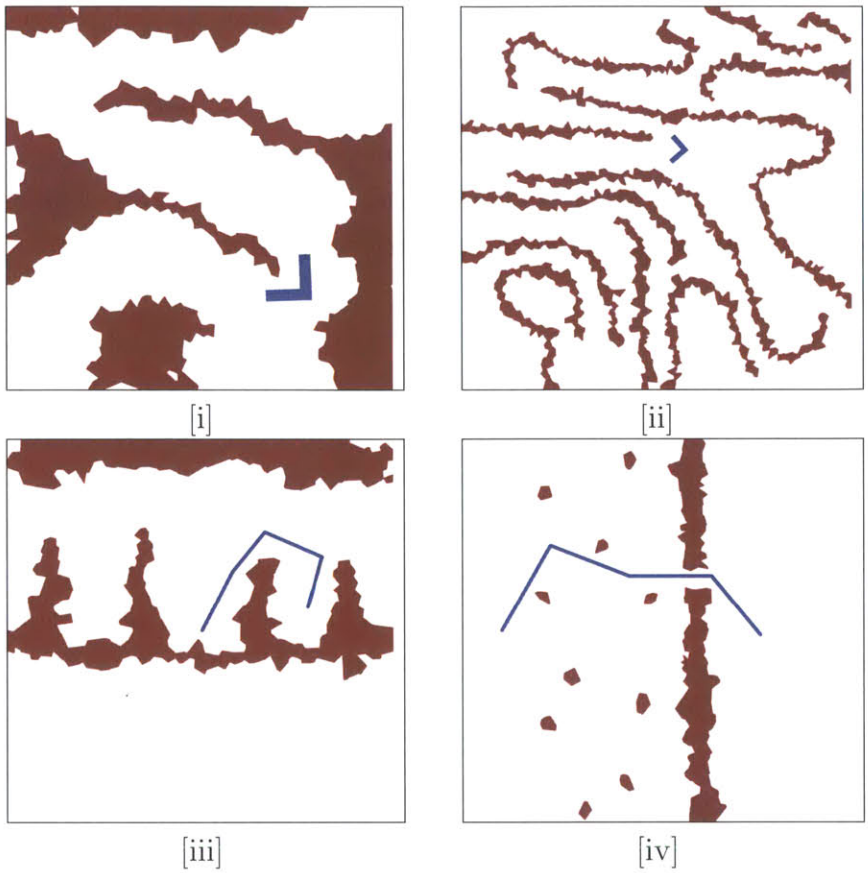

Figure 4-3: Obstacle sets for experiments. For experiments with the planar point and planar object, [i] and [ii] are mazes of different complexity. For the planar manipulator [iii] is a multi-crevice obstacle set with the manipulator in the center crevice and [iv] is a wall with a narrow opening.

4.4.1 Sampling Performance

The biased sampling method sacrifices independence of the sampling process, so we must question the quality of the induced sampling distribution. We compare the sample sets generated by KDS and RS by comparing the distribution of circumspheres in the Delaunay Triangulation of the point set (the radius of largest circumsphere being the L2 dispersionof the point set [56]).

Figure 4-4 illustrates the histogram of the radii of these circumspheres for obstacle sets [i] and [iii] using KDS and RS. As these plots show, the point set generated using KDS is quite similar to that of the point set generated using RS. The Kolmogorov-Smirnov p-values for the tests are 1.000, 0.893, and 0.999 respectively, suggesting the way in which KDS sacrifices independence in the sampling process does not

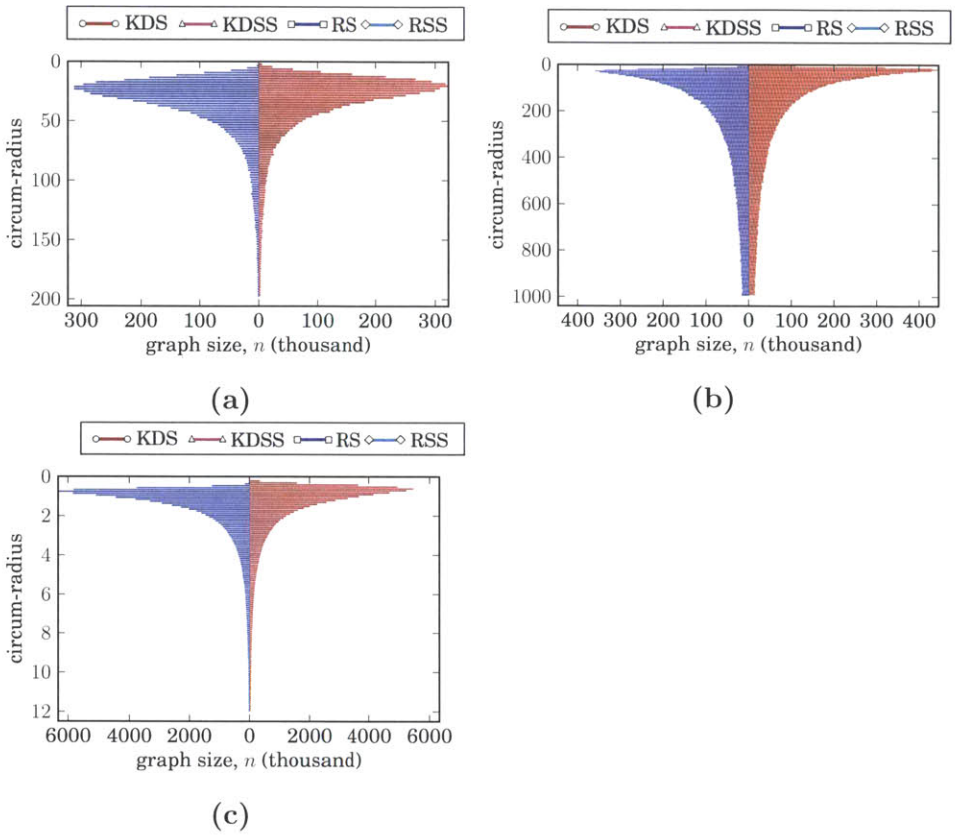

Figure 4-4: Histogram of circumsphere sizes for the Delaunay Triangulation of the point set generated after 1000 collision-free samples in experiments with (a) the planar point and obstacle set [i], (b) the planar object and obstacle set [i], (c) the planar manipulator and obstacle set [iii].

significantly affect the quality of the sampling sequence.

4.4.2 Planar Point Robot

Figure 4-5 shows the profiling results for the planar point experiments. In obstacle set [i] the obstacles are grouped to create wide regions of free space, while in [ii] they are arranged to divide up the free-space into many narrow passages. The latter should require a larger number of samples before the information encoded in the *kd*-tree is sufficient to improve the sampling success rate in KDSS. Indeed, as Figure 4-5d shows the sampling success rate for this obstacle set increases more slowly than for obstacle set [i], shown in Figure 4-5c. Note also that the proportion of the configuration space that is collision-free is higher for obstacle set [ii] than [i]. For these two reasons the iteration time for KDSS and obstacle set [ii] is not much different than for RSS, though

Figure 4-5b does show that there is a crossover point at around 5000 samples where KDSS becomes faster than RSS. We do see an approximately 5% improvement in iteration time of KDSS over RSS for obstacle set [i] as shown in Figure 4-5a. For both obstacle sets KDS less than 4% slower than RS alone.

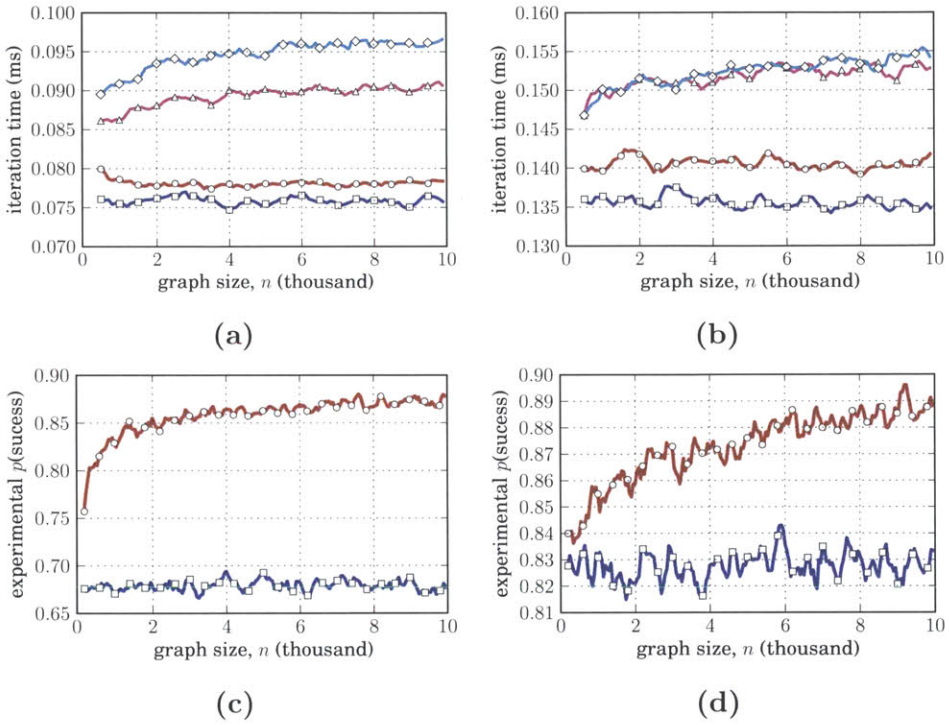

Figure 4-5: Observed iteration times (a), (b), and incremental sampling success rates (c), (d), for the planar point experiments. Figures (a) and (c) are for obstacle set [i] while (b) and (d) are for obstacle set [ii].

4.4.3 Planar Object Robot

While the planar point experiment is useful for gaining some intuition the low dimension of the configuration space and ease of collision checking are somewhat unrepresentative of the challenging planning problems which are typically solved with sampling-based planning algorithms.

Figure 4-6 shows the profiling results for the planar object experiments. KDSS has 25% lower incremental runtime than RSS for obstacle set [i] (Figure 4-6a) and a 15% lower runtime for obstacle set [ii] (Figure 4-6b). Note that for these obstacle sets, sampling generating (RS and KDS) alone takes roughly the same amount of time as sampling

and searching (RSS and KDSS). Clearly in these experiments sampling (and static collision checking) contributes to the majority of the runtime while graph searching, which is the theoretical bottleneck in sampling-based planning algorithms, has yet to begin dominating the runtime after 10,000 samples. The experimental sampling success rates (Figures 4-6c and 4-6d) show that the hyper-cubical decomposition of the configuration space does not cover the configuration space as well in three dimensions as it does in two, with the success rate topping out at around 60% in these experiments. Never the less, this is a significant improvement over the 30%-35% success rate of rejection sampling in these experiments, leading to the significant reduction in runtime.

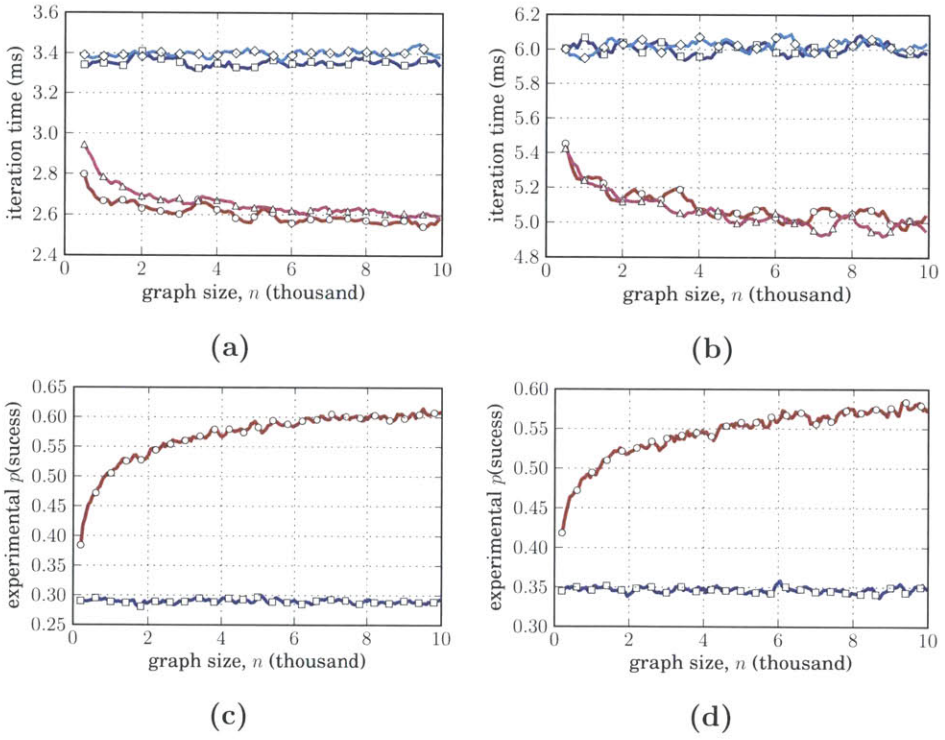

Figure 4-6: Observed iteration times (a), (b), and incremental sampling success rates (c), (d), for the planar object experiments. Figures (a) and (c) are for obstacle set [i] while (b) and (d) are for obstacle set [ii].

4.4.4 Planar Manipulator Robot

Experiments with the planar manipulator allows us to consider the performance of this method in higher dimension configuration spaces. Figure 4-7 shows the profiling results for the planar manipulator experiments. Obstacle set [iii] has a more complex representation with

more of the configuration space in collision, and thus KDS and KDSS show a high (65%) runtime improvement over both RS and RSS (Figure 4-7a). The improvement of the *kd*-tree based sampling technique for obstacle set [iv] is with a 5% runtime improvement of KDSS over RSS (Figure 4-7b). In this case the KDSS and RSS take roughly the same amount of time. We see in Figure 4-7d however that the proportion of the configuration space which is collision-free is quite high in this case (nearly 90% collision-free) while for obstacle set [iii] it is less than 10% collision-free. For obstacle set [iv] KDS provides only slightly higher sampling success rates. However, because collision checking is significantly more complex in this case we still see an improvement in the runtime of KDSS over RSS.

We also note that for obstacles set [iii] our method tops out at a sampling success rate of only 45% which is even less than in the planar object experiments. However, as with the planar object experiments, this increase in sampling success rate is significant enough to lead to the runtime improvement that we observe.

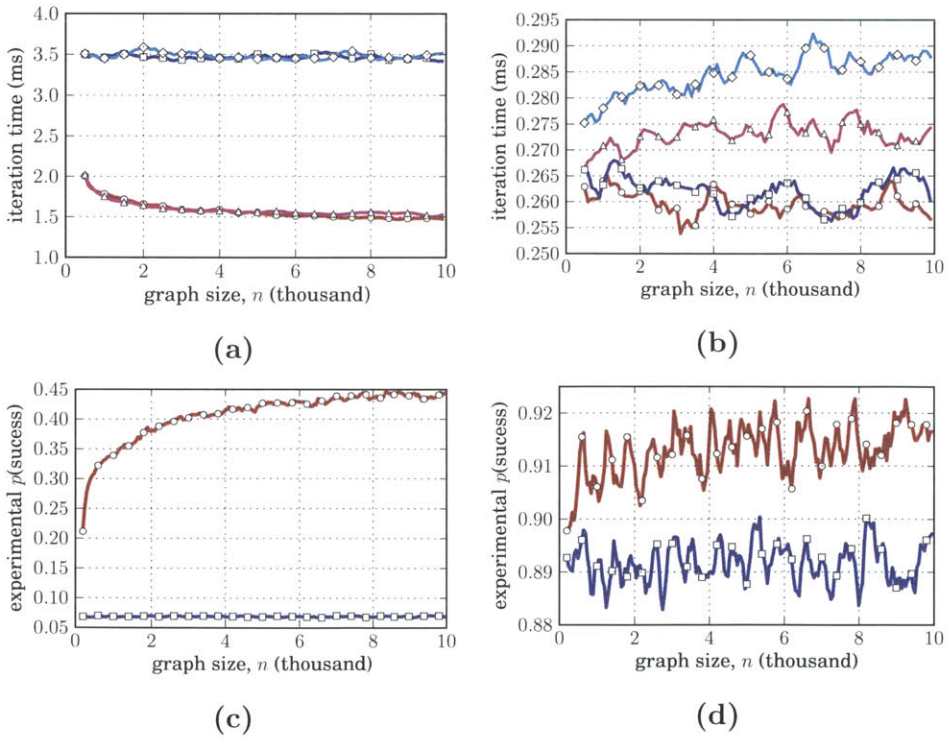

Figure 4-7: Observed iteration times (a), (b), and incremental sampling success rates (c), (d), for the planar point experiments. Figures (a) and (c) are for obstacle set [iii] while (b) and (d) are for obstacle set [iv].

4.5 Summary and Conclusions

We present a new method for sampling such that samples are drawn from a distribution that provably converges to uniform sampling over an initially unknown but discoverable and static subset of space. Our method works by recording the observed distribution of the subset of interest vs. the total space within subspaces covered by a spatial index. We demonstrate the specifics of this method with an algorithm for augmenting *kd*-trees. The observed number of samples (e.g., from the subset of interest vs. the total number samples) at each node is used to guide future sampling.

Our method can also be viewed as constructing an *approximate* obstacle representation that becomes more refined as more samples are generated (and samples are generated from areas that contain obstacles with increasingly low probability). However, experiments show that the number of samples required to characterize the free space sufficiently well to reduce rejection sampling over head is relatively small in many complex planning problems. The complexity of generating new samples also inherits a $\log n$ factor of complexity from the *kd*-tree around which it is built, but in practice this appears to be a useful compromise. In particular, this compromise is worthwhile when

- the complexity of the obstacle field is high, e.g., a large number of obstacles, or obstacles with complex shapes;
- the complexity of interference testing is high, e.g., a complex mapping from (high) configuration space to workspace volumes;
- the proportion of the configuration space which is collision-free is low.

We expect our method to be especially useful as a subroutine in sample based motion planning which are already bound to search operations on a spatial index.

Chapter 5

Sampling directly from a triangulation

The sampling algorithm in the previous chapter describes a method of combining tree based proximity data structures with collision information to inform the sampling priors. In this chapter we extend this algorithm to triangulation proximity data structures.

5.1 Uniform sampling from a simplex interior

We begin the discussion by describing an algorithm to efficiently generate a point from the interior of a simplex. Let $S^1 \subset \mathbb{R}^n$ be the canonical (i.e., unit) simplex.

Let $z_i \sim U(0,1)$ for i in $1 \ldots n-1$ be n scalar values drawn i.i.d from a uniform distribution over $(0,1)$. Let $z_0 = 0$ and $z_n = 1$. Let us reassign the labels i in $1 \ldots n-1$ such that $z_i < z_{i+1}$. Let $x_i = z_{i+1} - z_i$ for i in $0 \ldots n-1$. Let $\mathbf{x} = [x_0 \ldots x_{n-1}]$ be the vector of these ordered values.

Theorem 5 (Uniform sampling on the canonical simplex)**.** $\mathbf{x}$ *is distributed uniformly over* S^1*.*

The proof of theorem 5 is given in [88]. Now let V be a set of $n+1$ points $\mathbf{v}_i \in \mathbb{R}^n$ for i in $0 \ldots n$. Let $S = \text{conv}(V)$ be the simplex which is the convex hull of the vertices in V. If S is not degenerate, i.e., not all $\mathbf{v}_i \in V$ lie in the same hyperplane, then any point $\mathbf{x} \in \mathbb{R}^n$ may be

represented by $\boldsymbol{\lambda} = [\lambda_0 \dots \lambda_{n_1}]$ satisfying

$$\begin{aligned} \mathbf{x} &= \sum_{i=0}^{n} \lambda_i \mathbf{v}_i, \\ 1 &= \sum_{i=0} \lambda_0. \end{aligned} \tag{5.1}$$

We refer to λ_i the *barycentric coordinates* of $\mathbf{x}$ with respect to S.

Theorem 6 (General measure rule for barycentric coordinates)**.** *Let $\mathbf{x} \in \mathbb{R}^n$ be any point and $\boldsymbol{\lambda} = [\lambda_0 \dots \lambda_{n_1}]$ its barycentric coordinates. Furthermore, let $S_i = conv(V \setminus \mathbf{v}_i \cup \mathbf{x})$. Then $\mu(S_i) = \lambda_i \mu(S)$ where $\mu(X)$ is the Lebesgue measure of the set X.*

Proof. Without loss of generality we consider only the case of S_0 as the result is symmetric and holds for any assignment of indices. The Lebesgue measure of a simplex $S = \text{conv}(V)$ is given by [99]

$$\begin{aligned} \mu(S) &= \frac{1}{n!} \det \left| (\mathbf{v}_1 - \mathbf{v}_0) \dots (\mathbf{v}_n - \mathbf{v}_0) \right|, \\ \mu(S_0) &= \frac{1}{n!} \det \left| (\mathbf{v}_1 - \mathbf{x}) \dots (\mathbf{v}_n - \mathbf{x}) \right|. \end{aligned}$$

Let $\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \dots & \mathbf{v}_n \end{bmatrix}$ be the horizontal concatenation of the vertices $V \setminus \mathbf{v}_0$, and let $\mathbf{1} = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^\top$ be the $n \times 1$ vector of ones. Note that we assume S is not degenerate and so $\mathbf{V}$ is invertible. Note then the following:

$$\mathbf{x} = \sum_{0}^{n} \lambda_i \mathbf{v}_i = \lambda_0 \mathbf{v}_0 + \mathbf{V} \boldsymbol{\lambda}_{\setminus 0},$$

where $\boldsymbol{\lambda}_{\setminus 0} = \begin{bmatrix} \lambda_1 \dots \lambda_n \end{bmatrix}^\top$ is the $n \times 1$ vector of barycentric coordinates excluding λ_0. We begin by rearranging equation (5.1).

$$\begin{aligned} 1 - \lambda_0 &= \mathbf{1}^\top \boldsymbol{\lambda}_{\setminus 0} \\ &= \mathbf{1}^\top \mathbf{V}^{-1} \mathbf{V} \boldsymbol{\lambda}_{\setminus 0} \\ &= \mathbf{1}^\top \mathbf{V}^{-1} \left(\lambda_0 \mathbf{v}_0 + \mathbf{V} \boldsymbol{\lambda}_{\setminus 0} - \lambda_0 \mathbf{v}_0 \right) \\ &= \mathbf{1}^\top \mathbf{V}^{-1} \left(\mathbf{x} - \lambda_0 \mathbf{v}_0 \right) \\ 1 - \mathbf{1}^\top \mathbf{V}^{-1} \mathbf{x} &= \lambda_0 \left(1 - \mathbf{1}^\top \mathbf{V}^{-1} \mathbf{v}_0 \right). \end{aligned}$$

Applying the matrix determinant lemma

$$\det\left(A + \mathbf{u}\mathbf{v}^\top\right) = \det\left(A\right)\left(1 + \mathbf{v}^\top A^{-1}\mathbf{u}\right),$$

$$\begin{aligned}
\det\left(1 - \mathbf{1}^\top\mathbf{V}^{-1}\mathbf{x}\right) &= \lambda_0 \det\left(1 - \mathbf{1}^\top\mathbf{V}^{-1}\mathbf{v}_0\right) \\
\det\left(\mathbf{V}\right)\det\left(1 - \mathbf{1}^\top\mathbf{V}^{-1}\mathbf{x}\right) &= \lambda_0 \det\left(\mathbf{V}\right)\det\left(1 - \mathbf{1}^\top\mathbf{V}^{-1}\mathbf{v}_0\right) \\
\det\left(\mathbf{V} - \mathbf{x}\,\mathbf{1}^\top\right) &= \lambda_0 \det\left(\mathbf{V} - \mathbf{v}_0\,\mathbf{1}^\top\right) \\
\det\left|(\mathbf{v}_1 - \mathbf{x})\ldots(\mathbf{v}_n - \mathbf{x})\right| &= \lambda_0 \det\left|(\mathbf{v}_1 - \mathbf{v}_0)\ldots(\mathbf{v}_n - \mathbf{v}_0)\right| \\
\frac{1}{n!}\det\left|(\mathbf{v}_1 - \mathbf{x})\ldots(\mathbf{v}_n - \mathbf{x})\right| &= \lambda_0 \frac{1}{n!}\det\left|(\mathbf{v}_1 - \mathbf{v}_0)\ldots(\mathbf{v}_n - \mathbf{v}_0)\right| \\
\mu\left(S_0\right) &= \lambda_0 \mu\left(S\right).
\end{aligned}$$

□

Corollary 8. *Let $S_A = conv(V_A)$ be a simplex in $\mathbb{R}^n$ and $S_a = conv(V_a)$ be a second simplex in its interior: $S_a \subset S_A$. Now let $S_B = conv(V_B)$ be some other simplex in $\mathbb{R}^n$, and let $\boldsymbol{\lambda}_i$ be the barycentric coordinates of vertex $\mathbf{v}_{a,i} \in V_a$ with respect to S_A. Also, let $\mathbf{v}_{b,i} \in \mathbb{R}^n$ be the point with barycentric coordinates $\boldsymbol{\lambda}_i$ with respect to S_B, and let $S_b = conv(V_b)$. Then $S_b \subset S_B$ and $\mu(S_b)/\mu(S_B) = \mu(S_a)/\mu(S_A)$*

Note that if $\forall i$ in $0 \ldots n-1$, $0 \leq \lambda_i \leq 1$ then $\mathbf{x}$ is a *convex combination* of the vertices of the simplex, and must lie in the interior. Thus for a point $\mathbf{x}$ and its barycentric coordinates $\boldsymbol{\lambda}$ with respect to some simplex S, we know that $\mathbf{x} \subset S$ if and only if $\boldsymbol{\lambda} \in S^1$.

Theorem 7 (Uniformly sampled barycentric coordinates sample uniformly over any simplex)**.** *Let $S^1 \subset \mathbb{R}^{n+1}$ be the canonical, i.e., unit simplex and $S \subset \mathbb{R}^n$ be any nondegenerate simplex, i.e., $S = conv(V)$ for $V = \{\mathbf{v}_0, \ldots, \mathbf{v}_n\}$ where $\mathbf{v}_i \in \mathbb{R}^n$. Let $\boldsymbol{\lambda} \sim U(S^1)$ be a random point distributed uniformly over S^1. Furthermore, let $\mathbf{x} = \sum_{i=0}^n \lambda_i \mathbf{v}_i$ be the the point in $\mathbb{R}^n$ with barycentric coordinates $\{\lambda_0, \ldots, \lambda_n\}$. Then $\mathbf{x} \sim U(S)$*

Proof Sketch. In order to show that $\mathbf{x} \sim U(S)$ we must show that for any open subset X of S, $P(\mathbf{x} \in X) = \mu(X)/\mu(S)$. We will show that this is true if X is the interior of a polytope $P \subset S$.

Every polytope P emits a finite triangulation, i.e., a finite set of simplices $\mathcal{S}$ such that $S_i, S_j \in \mathcal{S}$, $i \neq j$, $S_i \cap S_j = \emptyset$ and $P = \bigcup_{\mathcal{S}} S_i$.

By corollary 8, the barycentric coordinates of the vertices of P emit a Lebesgue measure ratio preserving map of the polytope to a new polytope P' in the interior of any other simplex in $\mathbb{R}^n$. In particular, the surface of the canonical simplex in $\mathbb{R}^{n+1}$ is itself a simplex in $\mathbb{R}^n$, embedded in the $n+1$ dimensional hyperplane normal to $\mathbf{1}$. Because barycentric coordinates are sampled uniformly from this surface and they provide a measure preserving map from S to that surface, the probability that a sample lies in the polytope is $P(\mathbf{x} \in P) = {}^{\mu(P)}/_{\mu(S)}$. □

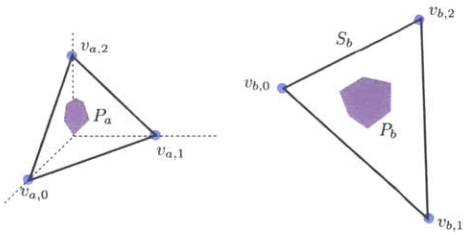

Figure 5-1: Barycentric coordinates provide a measure preserving map from the surface of the canonical simplex to any other simplex.

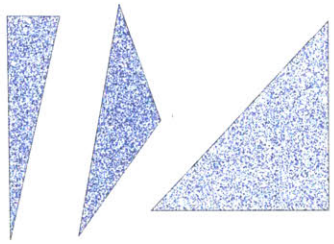

Figure 5-2: Some examples of uniform sampling within a 2-simplex (a.k.a. triangle)

Theorems 5 and 7 together yield a simple and efficient $O(n \log n)$ algorithm for sampling a point uniformly inside any simplex.

Input: U, a uniform RNG
Output: $\boldsymbol{\lambda} \in \mathbb{R}^n$
1 $Z \leftarrow \{0, 1\}$
2 **for** $i \in [1 \ldots n-1]$ **do**
3 sample $z_i \sim U(0, 1)$
4 $Z \leftarrow Z \cup \{z_i\}$
5 $Y \leftarrow \texttt{sort}(Z)$
6 **for** $i \in [0 \ldots n-1]$ **do**
7 $\lambda_i \leftarrow y_{i+1} - y_i$
8 **return** $\begin{bmatrix}\lambda_0 & \ldots & \lambda_{n-1}\end{bmatrix}^\top$

Algorithm 5.1: `SampleCanonical`(n) returns a point sampled uniformly from the surface of the canonical simplex in n dimensions

Input: V, a set of $n+1$ vertices $\mathbf{v}_i \in \mathbb{R}^n$
Output: $\mathbf{x} \sim U(\text{conv}(V))$
1 $\boldsymbol{\lambda} \leftarrow$ `SampleCanonical` $(n+1)$
2 $\mathbf{V} \leftarrow \begin{bmatrix}\mathbf{v}_0 & \dots & \mathbf{v}_n\end{bmatrix}$
3 **return** $\mathbf{V}\boldsymbol{\lambda}$

Algorithm 5.2: `SampleSimplex` (V) returns a point sampled uniformly from the interior of the simplex $S = \text{conv}(V)$

5.2 Sampling Uniformly from a Triangulation

The results of the previous section may be utilized in an efficient algorithm to sample a new point uniformly from the set covered by a triangulation.

Let V be a set of n vertices $\mathbf{v}_i \in \mathbb{R}^d$ with $n > d$. Also, let T be a *triangulation* of V, i.e., a set of non-overlapping simplices which cover $\text{conv}(V)$. Let $\mu(S)$ denote the Lebesgue Measure of a simplex $S \in T$, and let $\mu(T)$ be the Lebesgue Measure of $\text{conv}(V)$.

We may sample a point uniformly over the interior of $\text{conv}(V)$ by first selecting a simplex at random from a distribution where the probability of selecting a simplex S is proportional to $\mu(S)$. We may then apply algorithm 5.2 to sample a point uniformly over its interior. The resulting distribution is uniform over $\text{conv}(V)$.

An efficient algorithm for sampling from a discrete distribution with finite support can be realized by use of a *balanced tree of partial sums*: a type of binary tree. In the BTPS each element in the support is represented by a leaf node in the tree, and each interior node encodes the sum of the weight of its two children. We assume that if N is a node in the tree, then the following procedures are $O(1)$ time.

field	type	description
`weight` (N)	$\in \mathbb{R}$	The total weight summed over all leaf nodes in this node's subtree
`num` (N)	$\in \mathbb{N}$	The total number of descendants of this node
`parent` (N)	`Node`	The parent node
`child` (N, i)	`Node`	The ith child of N, $i \in 0, 1$
`payload` (N)	variable	if N is a leaf node, points to the corresponding element in the support of the distribution
`isLeaf` (N)	Boolean	true if N is a leaf node, false if it is an interior node

Table 5.1: `Node` Data Structure

As a notational convenience, let $\texttt{x}(N, i)$ denote the application of procedure `x` to the ith child of N, i.e., $\texttt{weight}(N, 0)$ denotes $\texttt{weight}(\texttt{child}(N, 0))$.

There is one leaf node in the tree for each simplex in the triangulation, and its weight is assigned the Lebesgue measure of that simplex. The interior nodes of the tree encode partial sum information, and information required to keep the tree balanced. In effect the tree is a binary search tree over the unit interval which is subdivided into segments of length proportional to the measure of the simplices in the triangulation. We may then draw a random number $X \sim U(0, 1)$ and then use the binary tree to discover the division of the unit interval on which X lies.

Input: P, a node in the tree
Input: L, a node to insert
1 $i \leftarrow \text{argmin}(\texttt{num}(P, i))$
2 **if** $\texttt{isLeaf}(P, i)$ **then**
3 $\quad L' \leftarrow \texttt{child}(P, i)$
4 $\quad \texttt{child}(P, i) \leftarrow \texttt{Node}(P, L, L')$
5 **else**
6 $\quad \texttt{Insert}(\texttt{child}(P, i), L)$
7 $\texttt{meas}(P) \leftarrow \texttt{meas}(P) + \texttt{meas}(L)$
8 $\texttt{num}(P) \leftarrow \texttt{num}(P) + \texttt{num}(L)$
9 **return**

Algorithm 5.3: `Insert` (P, L) add an element to the support of the distribution

Input: L, the leaf node to remove
1 $P' \leftarrow \texttt{parent}(L)$
2 $P \leftarrow \texttt{parent}(P')$
3 $\texttt{child}(P, i) \leftarrow \texttt{child}(P', i)$
4 **while** $P \neq \emptyset$ **do**
5 $\quad \texttt{weight}(P) \leftarrow$
$\quad \texttt{weight}(P) - \texttt{weight}(L)$
6 $\quad \texttt{num}(P) \leftarrow \texttt{num}(P) - \texttt{num}(L)$
7 $\quad P \leftarrow \texttt{parent}(P)$
8 **return**

Algorithm 5.4: `Remove` (L) remove an element from the support of the distribution

Algorithms 5.3 and 5.4 give the insert and removal procedure for the tree. Note that the removal procedure does not perform any rotations. Explicit maintenance of the tree's balance after removal is unnecessary

for our application since the support of the distribution is always growing: accounting for the number of leaves in each subtree is sufficient.

Algorithm 5.5 gives the recursive procedure for sampling from the distribution. The procedure starts at the root, R, such that to generate the sample one may apply $\texttt{Select}\left(R, \sim U\left(0,1\right), 0, \texttt{weight}\left(R\right)\right)$.

It is straightforward to see that insertion, removal, and sampling from the distribution are all $O\left(\log n\right)$ time, and the tree requires $O\left(n\right)$ storage.

```
   Input: P, an interior node
   Input: u ∈ [0, 1]
   Input: w ∈ ℝ, weight of pruned subtrees
   Input: W ∈ ℝ, total weight
1  if u < (w + weight(P, 0))/W then
2  |  C ← child(P, 0)
3  else
4  |  C ← child(P, 1)
5  |  w ← weight(P, 0)
6  if isLeaf(C) then
7  |  return payload(C)
8  else
9  |  Select(C, u, w, W)
```

Algorithm 5.5: $\texttt{Select}\left(P, u, w, W\right)$ sample an element from the support of a discrete distribution. Initially P is the root of the tree, W is $\texttt{weight}\left(P\right)$, w is zero, and $u \sim U\left(0,1\right)$.

5.3 k-NN Searching in a Delaunay Triangulation

The canonical algorithm for performing k nearest neighbor searching in a Delaunay Triangulation is more readily explained in terms of the Voronoi diagram (the dual of the Delaunay Triangulation).

The algorithm first finds the nearest neighbor of the query point $\mathbf{x}_q$ by finding the Voronoi cell containing $\mathbf{x}_q$ (a *point location query*) and then retrieving the generator site $\mathbf{x}_0$ associated with that cell. Equivalently, $\mathbf{x}_0$ is the nearest vertex to $\mathbf{x}_q$ among vertices of the Delaunay simplex containing $\mathbf{x}_q$.

The algorithm then proceeds by maintaining a priority queue of Voronoi cells to expand. When a cell is expanded, all of the neighbor-

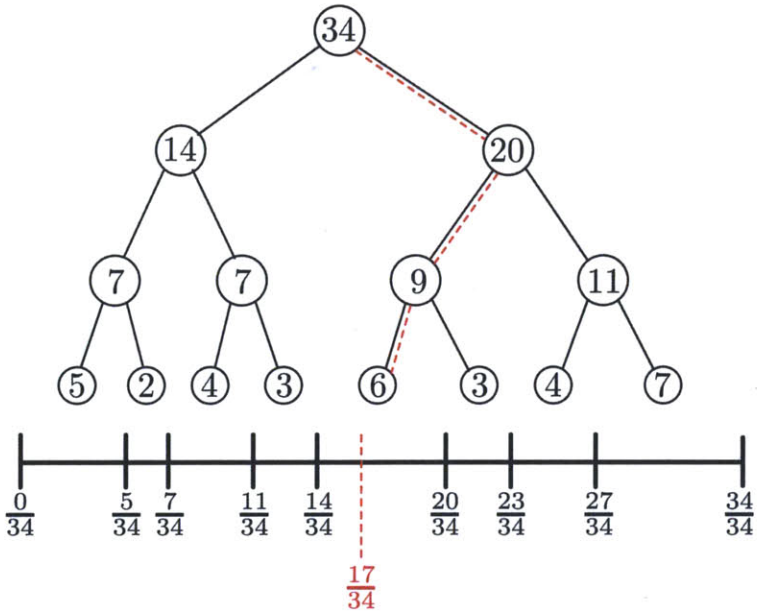

Figure 5-3: A balanced tree of partial sums for a distribution over eight elements where the weights of the elements are 5, 2, 4, 3, 6, 3, 4, and 7. Given a query value of $u = 1/2$ the dotted path shows the traversal of the `Select` procedure.

ing cells are evaluated to determine the distance of their associated generator site to the query point. Each of these sites are added to the priority queue, keyed on their distance. The expanded cell is then added to the result set. The processes is illustrated for a 2d point set in figure 5-4. Note that the sites associated with each Voronoi neighbor of a Voronoi cell generated by $\mathbf{x}_i$ are exactly the Delaunay neighbors $\mathbf{x}_i$. Pseudocode for this algorithm is given in algorithm 5.6.

Assuming that the point location query is efficient, algorithm 5.6 is

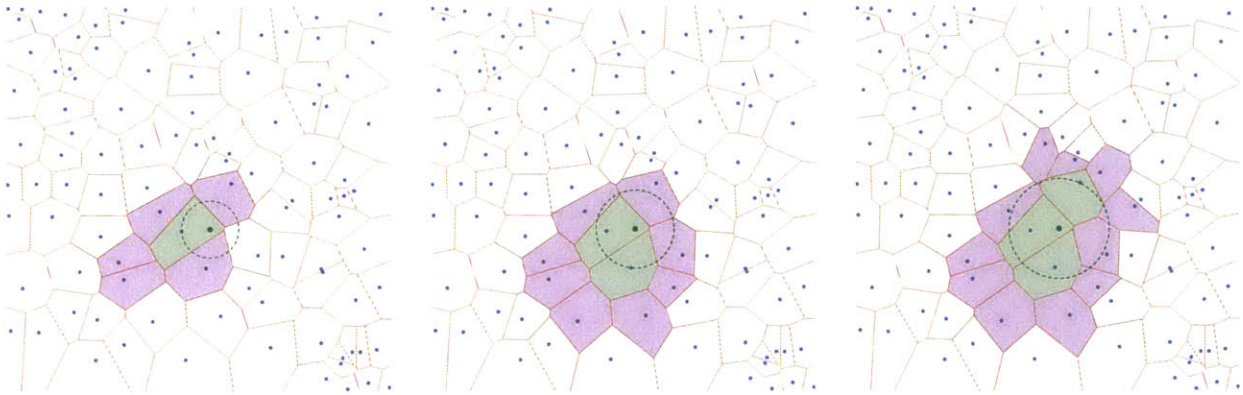

Figure 5-4: First three iterations of k-NN search in 2d. Q (in purple) is initialized with the cell containing the query point. The result set R (in green) is built up by successively adding the nearest site of Q.

```
   Input: T, a Delaunay triangulation
   Input: x_q, the query point
   Input: k, number of nearest neighbors to find
 1 initialize O,R as PriorityQueue
 2 initialize C as Set
 3 s ← containingSimplex(T, x_q)
 4 (x_0, d_0) ← nearestVertex(s, x_q)
 5 insert(O, (d_0, x_0))
 6 while |R| < k or MinKey(O) < MaxKey(R) do
 7     (d_i, x_i) ← popMin(O)
 8     insert(R, (d_i, x_i))
 9     if |R| > k then
10         popMax(R)
11     insert(C, x_i)
12     for x_j ∈ delaunayNeighbors(T, x_i) do
13         if x_j ∉ O and x_j ∉ C then
14             d_j ← ||x_j − x_q||
15             insert(O, (d_j, x_j))
16 return R
```

Algorithm 5.6: `findNN` $(T, \mathbf{x}_q, k)$ find the k nearest neighbors to $\mathbf{x}_q$ in a Delaunay triangulation T.

output sensitive optimal and has runtime $O(k)$ for constant dimension. The complexity of the point location query, however, is what dominates the overall algorithm runtime. Optimal indices of the DT may find the simplex containing the query point in $O(\log n)$ time for constant dimension. Each of these two procedures, however, scale exponentially in dimension.

5.4 Combined sampling and k-NN Searching in a Delaunay Triangulation

As with kd-trees in the previous section, if the query point is generated from a uniform distribution over the convex hull of the current point set, then we may combine sampling and proximity queries to eliminate the point location query. In doing so we exchange a separate sampling procedure followed by a $O\left(c^d \log n\right)$ point location query with a single $O(\log n)$ combined sampling and point location procedure whose complexity does not grow with dimension.

Chapter 6

Proximity queries for robotic systems

Proximity queries are of fundamental importance to sampling-based motion planning algorithms, as it drives their asymptotic incremental runtime. For many sampling-based planning algorithms these proximity queries are k-nearest neighbor or range queries. There are known data structures for Euclidean spaces which yield efficient proximity queries for several metrics, such as the spatial indices referred in the introduction, however they suffer either space or query time exponential in dimension. The often used kd-tree, for example, can find k nearest Euclidean neighbors in d dimensions in time $O\left(G(d)\log n\right)$ [27], where $G(d)$ is exponential in d. Nearest neighbor queries in a kd-tree are never worse than $O\left(dn\right)$.

Locality sensitive hashing in a Euclidean space [5] avoids the exponential size or query time of spatial tesselations and decompositions by finding a lower dimensional subspace on which to project and bin the sites of a data set. The problem finding a suitable subspace however precludes the use of such data structures when the point set is built incrementally as in many sampling-based planning algorithms.

In this chapter we begin by reviewing priority searching on spatial indices noting how these searches can be extended to different metrics and distance functions while preserving their correctness and complexity (section 6.1). The extensibility of priority searching algorithms beyond traditional metrics (Euclidean,$\ell-1$, etc) was previously discussed in [106] in which formulas are derived for utilizing kd-trees as an efficient index for a circle, real projective spaces, and Cartesian products

of these and Euclean spaces. In this chapter we further extend the set of configuration spaces which can be efficiently indexed with *kd*-trees by providing the requisite formulas for computing shortest path distances to hyperrectangular-cells in S^n, $SO(3)$, $R^3 \times SO(3)$, and Dubins vehicles.

6.1 Priority searching in spatial indices

Let $S \subset X$ be a finite set of n sites in a metric space X with metric d_X. A bounding volume hierarchy is a tree data structure, where each vertex represents some subset of X, usually compact and connected. Let $\mathbb{S}(v_i) \subset X$ denote the set covered by vertex v_i. Let v_0 be the root vertex of the tree. The hierarchy represents a successive decomposition of $\mathbb{S}(v_0)$. In other words if v_j is a decendent in the tree from v_i, then $\mathbb{S}(v_j) \subset \mathbb{S}(v_i)$. The leaf vertices in the tree reference a subset of the sites. Let v_i be a leaf node and S_i be the set of sites referenced by v_i, then $S_i \subset \mathbb{S}(v_i)$. Furthermore, these referenced sets are disjoint, i.e., for all leaves i, j, $S_i \cap S_j = \emptyset$.

Priority based k-NN searches on a bounding volume hierarchy can be performed with a straightforward branch-and-bound algorithm, as illustrated in algorithm 6.1.

```
1  initialize Q as a min-priority queue
2  initialize R as a max-priority queue
3  v_0 ← root(T)
4  insert(Q, (0, v_0))
5  while size(Q) > 0 and
6        size(R) < k or minKey(Q) < maxKey(R) do
7     (·, v) ← popMin(Q)
8     for x ∈ sites(v) do
9        insert(R, d_X(q, x), x)
10       if size(R) > k then
11          popMax(R)
12    for v' ∈ children(T, v) do
13       insert(Q, d_X(q, v'), v')
```

Algorithm 6.1: `BranchAndBound` $(q, k, d(\cdot,\cdot), T)$

This priority search algorithm is common for many BVH data structures including *kd*-trees [27], orthant-trees (n-dimensional quadtrees), ball-trees [71], etc.

Let T be a BVH, $q \in X$ be a query point, and $v \in T$ be a vertex of the T. We define $d_X(q, v)$ to be the normal Hausdorff distance

$$d_X(q, v) = \inf_{x \in \mathbb{S}(v)} d_X(q, x).$$

Theorem 8. *Given X,S,T,q,d_X as described, algorithm 6.1 is correct.*

Proof. The proof is by contradiction. Assume that s_i is one of the k nearest sites to q and that Algorithm 6.1 has terminated with $s_i \notin R$. Let v_i is the leaf node of T referencing s_i. Assume that v_i was visited. Then s_i was entered into R and must have been removed. This means that k other sites were entered into R with distance less than $d_X(q, s_i)$. Since each vertex is visited exactly once by construction, and vertices reference disjoint subsets of S, this contradicts the fact that s_i is one of the k nearest sites to q.

Now assume that v_i was not visited. We know that $d_X(q, v_i) \leq d_X(q, s_i)$. Furthermore we know that $d_X(q, v_j) \leq d_X(q, v_i)$ for all v_j in the ancestry of v_i, up to the root v_0. Since v_i was not visited, this means that either v_i or one of its ancestry is in Q, so $\texttt{size}\,(Q) > 0$. Because the algorithm terminated both $\texttt{size}\,(R) == k$ and $\texttt{minKey}\,(Q) \geq \texttt{maxKey}\,(R)$ must be true. However, since v_i or one of its ancestry is in Q, $\texttt{minKey}\,(Q) \leq d_X(q, v_i)$, and because v_i is one of the k nearest points to q, $\texttt{maxKey}\,(R) \geq d_X q, v_i$. And thus we have arrived at a contradiction. □

The branch-and-bound priority search algorithm is common to other spatial indices, notably tesselations. A tesselation is a graph data structure, where each vertex represents some compact, connected subset of X. The vertices are disjoint except at a common boundary, i.e., $\lambda\left(\mathbb{S}\left(v_i\right) \cap \mathbb{S}\left(v_j\right)\right) = 0$, and the vertices form cover of X, i.e., $\bigcup_{v_i \in V} \mathbb{S}\left(v_i\right) = X$.

Vertices v_i, v_j are connected by an edge in the graph if $\mathbb{S}\left(v_i\right) \cap \mathbb{S}\left(v_j\right) \neq \emptyset$. Each vertex v_i in the graph references the subset of sites S_i which lie in $\mathbb{S}\left(v_i\right)$, i.e., $S_i = \mathbb{S}\left(v_i\right) \cap S$.

The priority search algorithm may be applied to tesselations by replacing $\texttt{children}\,(\cdot, v)$ with $\texttt{neighbors}\,(\cdot, v)$, and adding additional accounting to avoid multiple visits to a cell (algorithm 6.2).

Note that q and $d_X(q, \cdot)$ together define both an orientation and an ordering on the graph. Algorithm 6.2, enumerates cells of the tes-

```
1  initialize Q as a min-priority queue
2  initialize R as a max-priority queue
3  v_0 ← cellContaining(G, q)
4  insert(Q, (0, v_0))
5  while size(Q) > 0 and
6        size(R) < k or minKey(Q) < maxKey(R) do
7     (·, v) ← popMin(Q)
8     for x ∈ sites(v) do
9        insert(R, d_X(q, x), x)
10       if size(R) > k then
11          popMax(R)
12    for v' ∈ neighbors(G, v) do
13       if notMarked(v') then
14          mark(v')
15          insert(Q, d_X(q, v'), v')
```

Algorithm 6.2: `BranchAndBound` $(q, k, d(\cdot,\cdot), G)$

selation under this topological ordering. Common tesselations include the Delaunay Triangulation and its dual the Voronoi Diagram [6]. Also, note that the leaves of a *kd*-tree or an orthant tree form a tesselation, and in fact algorithm 6.1 on these hierarchies will expand leaf nodes in the same order as would algorithm 6.2 on the tesselation of the leaves nodes.

Let G be a tesselation of X indexing S, and q a query point. Also, let $X_d = \{x \in X \mid d_X(q,x) \leq d\}$ be the sublevel set of $d_X(q,\cdot)$ on X. Let X and $d_X(q,\cdot)$ be such that $X_{d_0} \subseteq X_{d_1}$ for all $d_0 \leq d_1$, and X_d is connected for all $d \in \mathbb{R}$.

Theorem 9. *Given X, S, G, q, d_X as described, algorithm 6.2 is correct.*

Proof. Proof sketch. The proof follows from the fact that cells of the tessellation are expanded in topological order. The set of cells that have been expanded at every iteration forms a connected subset of the metric space. Furthermore, every unexpanded cell sharing a boundary of this connected subset is in Q. Thus the sublevel set of d_X at `minKey` (Q) is fully contained in the set of expanded cells, and all unevaluated sites must be further away than `minKey` (Q). If R contains k sites and `maxKey` (R) < `minKey` (Q) then they must be the k nearest sites to the query. □

Note that the correctness of the priority search algorithm for bounding volume hierarchies depends only on the ability to evaluate the Haus-

dorff distance of the query point to a vertex volume in the hierarchy. Correctness for tesselations also requires that the distance function satisfies a notion of at least *directional continuity.*

6.2 Indexing S^n

We are concerned with the problem of providing an efficient search index for points distributed over an n-sphere,

$$S^n = \left\{ \mathbf{x} \in \mathbb{R}^{n+1} \,|\, \mathbf{x}^\top \mathbf{x} = 1 \right\},$$

under the *orthodromic* (great-circle distance) metric. The distance between two points $(\mathbf{x}, \mathbf{y}) \in S^n$ is defined as the length of the shortest arc along S^n connecting $\mathbf{x}$ to $\mathbf{y}$:

$$d_S(\mathbf{x}, \mathbf{y}) = \cos^{-1}\left(\mathbf{x}^\top \mathbf{y}\right). \tag{6.1}$$

We present two options: hyper-rectangular BVH of $\mathbb{R}^{n+1}$ (i.e., *kd*-tree or orthant tree), and a Voronoi Diagram of S^n.

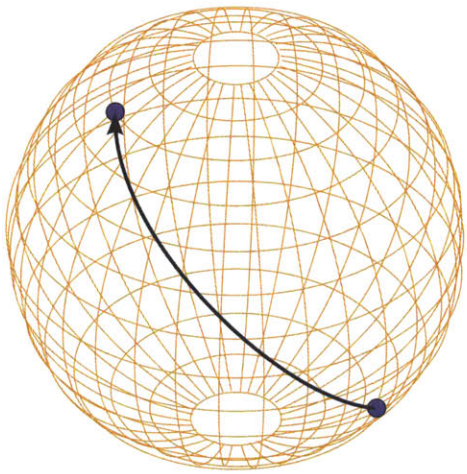

Figure 6-1: The orthodromic or great-circle distance between two points on S^2

6.2.1 Hyper-rectangular BVH of S^2

Let $X \subset S^n$ be a set of $k > n+1$ points. Furthermore, let the distance between two points $(\mathbf{x}, \mathbf{y}) \in S^n$ be defined as the orthodromic distance

between the points, equation (6.1). We may utilize a hyper-rectangular bounding volume hierarchy of $\mathbb{R}^{n+1}$ if we provide the Hausdorff distance kernel

$$d_S(\mathbf{q}, v) = \inf_{\mathbf{x} \in \mathbb{S}(v) \cap S^n} d_S(\mathbf{q}, \mathbf{x}).$$

Thus, $d_S(\mathbf{q}, v) = d_S(\mathbf{q}, \mathbf{x}^*)$ where $\mathbf{x}^*$ is the point in $\mathbb{S}(v) \cap S^n$ which minimizes $d_S(\mathbf{q}, \mathbf{y})$. We may simplify the derivation by noting that $\mathbf{x}*$ also minimizes the function $1 - \mathbf{q}^\top \mathbf{x}$. Thus we seek the solution the following optimization problem:

$$\begin{array}{ll} \text{minimize} & q - \mathbf{q}^\top \mathbf{x} \\ \text{subject to} & 1 - \mathbf{x}^\top \mathbf{x} = 0, \\ & a_0 \leq x_0 \leq b_1, \\ & \cdots \\ & a_{n+1} \leq x_{n+1} \leq b_{n+1}, \end{array}$$

where $\mathbf{a}$ and $\mathbf{b}$ are the min and max-extents of the hyper-rectangle $\mathbb{S}(v)$. This problem may be solved by combinatoral expansion of the active constraint set. For each constraint of i of v, at the optimal solution, either the min-extent is active ($x_i = a_i$), the max-extent is active ($x_i = b_i$), or neither is active. There are 3^n possible combinations of active constraints at the optimal solution. For a particular combination of active constraints, if the solution provided that constraint set also satisfies all non-active constraints, we say that the solution is *feasible.* Thus the procedure is to compute $\mathbf{x}_j^*$ assuming a given constraint set, $j \in 1 \ldots 3^n$, and then compute $\mathbf{x}^*$ as the minimum of feasible $\mathbf{x}_j^*$.

For a given assumed active constraint set, we may solve the optimization problem by forming the augmented cost function

$$J(\mathbf{x}) = \left(1 - \mathbf{q}^\top \mathbf{x}\right) + \lambda \left(1 - \mathbf{x}^\top \mathbf{x}\right).$$

The optimal solution then is found where $\partial J / \partial x_i = 0$ for all inactive

constraints i, and $\partial J/\partial \lambda = 0$.

$$\frac{\partial J}{\partial x_i} = -q_i - 2\lambda x_i = 0, \tag{6.2}$$

$$\frac{\partial J}{\partial \lambda} = 1 - \mathbf{x}^\top \mathbf{x} = 0. \tag{6.3}$$

Substituting equation (6.2) into equation (6.3), we find

$$\lambda^2 = \frac{1}{4}\left(\frac{\sum_{i \text{ inactive}} q_i^2}{1 - \sum_{j \text{ active}} x_j^2}\right). \tag{6.4}$$

From equation (6.4) we find two solution from λ (one positive, one negative), which we may then substitute into equation (6.2) to find $x_i = -q_i/2\lambda$. This gives us two candidates for $\mathbf{x}_j^*$. For each, we evaluate $a_j \leq x_j \leq b_j$ for all inactive constraints j, and if all are satisfied we mark the candidate feasible. We then take from these up to $2 \cdot 3^n$ feasible candidates for $\mathbf{x}^*$ the one with minimum distance.

6.2.2 Voronoi Diagram of S^n

Let $X \subset S^n$ be a set of $k > n+1$ points in general position (i.e., no $n+1$ points of X lie in the same hyperplane). Furthermore, let the distance between two points $(x, y) \in S^n$ be defined as the orthodromic distance between two points: $d_S(x, y) = \cos^{-1}(x \cdot y)$. Finally, consider the convex-hull of X in $\mathbb{R}^{n+1}$, and let T be the *surface triangulation* of the convex hull.

Theorem 10. *T is the Delaunay Triangulation of X under $d_S(\cdot, \cdot)$.*

Proof. We prove by contradiction. Let $\sigma \in T$ be a simplex of the surface triangulation of $\text{conv}(X)$, $\sigma = \text{conv}(x_0 \dots x_{d-1})$. Let $(\hat{n}_\sigma, d_\sigma)$ be the normal and offset of the hyperplane coincident to σ, with $\hat{n}$ oriented away from the origin, i.e., $\text{conv}(X) \subset \{x \,|\, \hat{n}_\sigma \cdot x \leq d_\sigma\}$. Let x_σ be the unique point on S^{d-1} satisfying $\hat{n}_\sigma \cdot x_\sigma > d_\sigma$, and $d(x_\sigma, x_i) = r_\sigma$, for all $(i) \in 0 \dots d-1$. We call r_σ the circumradius of σ, and x_σ the circumcenter. Also, assume that there exists $y \in X \setminus \sigma$ with $d(x_c, y) < r_\sigma$. The set $\{x \,|\, x \in S^{d-1}, d(x_c, x) < r_\sigma\}$ is equal to the intersection of S^{d-1} and the half-space $\{x \,|\, \hat{n}_\sigma \cdot x > d_\sigma\}$ with X. If y exists then this set is non-empty, which is a contradiction.

Thus, for all surface simplices σ, of the hull triangulation, with circumcenter x_σ and circumradius r_σ, X contains no point y with $d(x_\sigma, y) \leq r_\sigma$ other than $x \in \sigma$. □

Because simplices of T satisfies a notion of Delaunay for orthodromic distances, we may the dual of T as a Voronoi Diagram of X under this distance metric. In particular, we may apply algorithm 6.2 to the Delaunay graph. Thus we may build and maintain a Voronoi diagram of $X \subset S^n$ by building and maintaining the Convex Hull of X in $\mathbb{R}^{n+1}$ using, e.g. [19].

6.3 Indexing $SO(3)$

We may utilize the results of the previous section to build an index (either a BVH or a tesselation) for rigid body orientations, represented as unit quaternions, where the distance between two orientations is the angle of the smallest rotation between the two orientations. There are, however, a few caveats: the angle of this rotation is two times the angle between unit quaternions, and the space of unit quaternions is a double cover of $SO(3)$. We may address both facts by simply evaluating $d_S(q, v)$ and $d_S(-q, v)$ and taking the minimum of the two.

6.4 $\mathbb{R}^3 \times SO(3)$

Equipped with the ability to index $SO(3)$ we may now address point indices of rigid body configurations under distance metrics which integrate the Euclidean distance in $\mathbb{R}^3$ and the geodesic distance in SO(3) (such as a weighted norm of the two). We may unify the two spaces into a single seven dimensional *kd*-tree as in [106]. Figure 6-2 illustrates the runtime of algorithm 6.1 for k-nn queries with $k = 10$ on point sets of various size, where points are distributed uniformly over $\mathbb{R}^3 \times SO(3)$. Runtime results are shown for two implementations: a brute force search (bf) and a scheduled *kd*-tree in which cells are divide at the centroid of the longest (weighted) side (skd) [27]. Each data point is the averaged wall-clock runtime of 30 queries drawn at random from a uniform distribution. The distance function is a weighted 1-norm of the product

space:

$$d_X((\mathbf{x}_0, \mathbf{q}_0), (\mathbf{x}_1, \mathbf{q}_1)) = ||\mathbf{x}_1 - \mathbf{x}_0|| + w \cdot \cos^{-1}\left(2(\mathbf{q}_0^\top \mathbf{q}_1)^2 - 1\right).$$

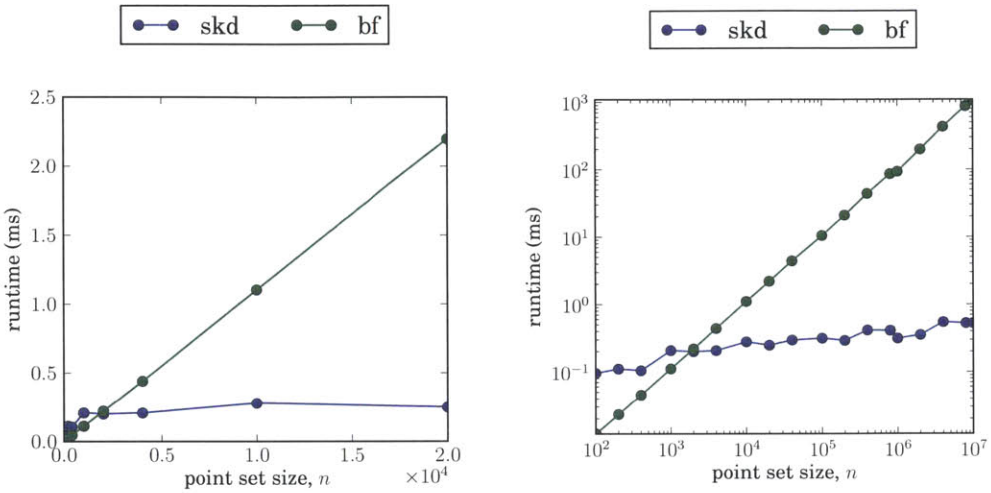

Figure 6-2: Runtime for 10-nearest neighbor queries in $\mathbb{R}^3 \times SO(3)$. (left) is in a linear scale and (right) is log-log.

The results in figure 6-2 demonstrate that the seven dimensional *kd*-tree is an effective proximity index for rigid body configurations under this distance metric. We see that the cost of brute force evaluations of the Hausdorff distance to a hyper-rectangle does induce some significant overhead: the time to evaluate the distance to a hyper rectangle is approximately 81x the cost of evaluating the distance to a point. However we see that this overhead is marginalized quite quickly and the *kd*-tree becomes faster after about 2000 points.

6.5 Dubins Vehicles

As shown at the beginning of this chapter, priority searching on spatial indices is correct even for distance functions which are not metrics, provided that we may compute the Hausdorff distance of a query point to a bounding volume node. In this section we derive this distance for a Dubins vehicle under the shortest path distance and for a hyper-rectangular volume of the configuration space.

A Dubins vehicle takes configurations from $\mathbb{R}^2 \times S^1$, however the vehicle is not free to move arbitrarily through configuration space. Rather

it is dynamically constrained with dynamics

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 1/R \end{bmatrix} u_2, \tag{6.5}$$

where u_1 is the velocity and u_2 is the turning rate. We may generate plans for a Dubins vehicle in a sampling-based planner using as a local steering function the shortest path between any two Dubins states. It is known that the shortest path between any two states takes is composed of three or fewer segments where each segment is either a full left turn, a full right turn, or a straight segment [98].

Let H by a hyper rectangle of the state space, as defined by its min and max-extents $\mathbf{a} = (x_a, y_a, \theta_a)$ and $\mathbf{b} = (x_b, y_b, \theta_b)$. Given a query point $\mathbf{q} = (x_q, y_q, \theta_q)$ the point $\mathbf{x}^* = (x^*, y^*, \theta^*) \in H$ nearest the query is given by the solution to the following minimization problem:

$$\begin{aligned} \text{minimize} \quad & d_D(\mathbf{q}, \mathbf{x}), \\ \text{subject to} \quad & x_a \leq x \leq x_b, \\ & y_a \leq y \leq y_b, \\ & \theta_a \leq \theta \leq \theta_b. \end{aligned}$$

The Hausdorff distance then is given by $d(q, \mathbf{x}^*)$. We may solve this optimization problem as with the shortest arc distance on S^n by combinatoral expansion of the constraint set and shortest path primitives. For each constraint, at the optimal solution $\mathbf{x}^*$, either the min-extent is active, the max-extent is active, or neither is active. There are $3^3 = 27$ possible combinations of active constraints. For a particular combination of active constraints one or more of the six primitive solution types must be considered. If a primitive solution $\mathbf{x}_j^*$ satisfies all non-active constraints for a particular assumed set of active constraints, then we say that the solution is feasible. $\mathbf{x}^*$ is then found by taking the minimum of all feasible solutions.

In the following subsections we enumerate the possible active constraint sets and derive the set of potentially feasible optimal paths given the active constraints.

We will use the abreviations L for a full left turn, R for a full right

turn, and S for a straight segment, so the designation LSR, for example, signifies a Dubins path with a full left turn followed by a straight segment and then a full right turn. The geometry of the Dubins vehicle state is depicted in figure 6-3, where r is the minimum turning radius corresponding to a maximum turning rate u_2. In many cases it is easier to work with the geometry of the optimal path components. We denote the center of the ith arc segment as $c_i = (x_{c,i}, y_{c,i})$. We may also refer to the angular coordinate of the vehicle with respect to the circle of radius r around c_i. Angular coordinates at points of interest are shown in figure 6-4 and labeled α_i.

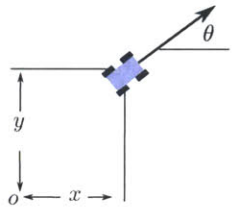

Figure 6-3: Geometry of a Dubins state

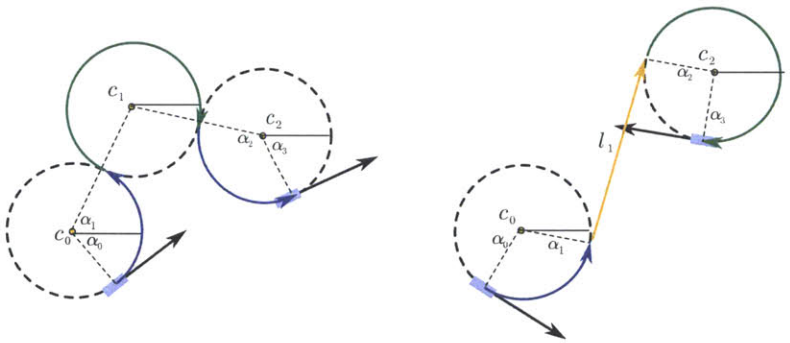

Figure 6-4: Geometries of Dubins paths. c_i is the center of the i'th segment if the segment is an arc. l_1 is the length of the straight segment if it is part of the path. α_i is the angular coordinates of the start or end of an arc segment with respect to the circle which the arc follows.

6.5.1 0 active constraints

If there are no active constraints than the solution path is a degenerate zero length path and it is feasible if $\mathbf{q} \in H$.

6.5.2 1 active constraint, x or y

If there is a single active constraint for either x or y, then there are four cases to consider. The shortest path to the constraint will start with an arc segment as all optimal paths do, so we consider both an initial path of L and an initial path of R. The procedure for the two cases are symmetric so we consider only an initial arc segment of type L. Let $c_0 = (x_0, y_0))$ be the center of the initial arc, and α_0 the inital angular coordinate of the vehicle on the circle coincident to the arc. Let the constraint be $y = y_a$. If $y_a < y_0$, the path follows nominally follows L until $\alpha_1 = \pi$, and then switches to a straight segment until reaching $y = y_a$. If $y_a > y_c$ the path nominally follows L until $\alpha_1 = 0$, and then switches to a straight line segment. If the path reaches y_a before reaching α_1 then there is no straight line segment. These three cases are illustrated in figure 6-5.

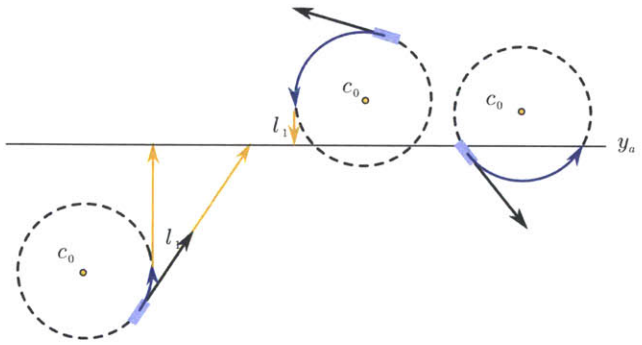

Figure 6-5: One active constraint in either x or y has solution primitive L,R,LS, or RS.

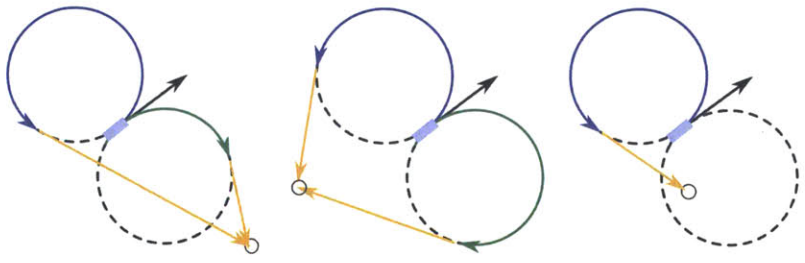

Figure 6-6: Two active constraints specifying x and y yields either LS or RS solutions.

6.5.3 1 active constraint, θ

The optimal path is a single arc segment. We evaluate both the distance of an L segment and an R segment required to achieve a terminal orientation of θ, and take the minimum of the two.

6.5.4 2 active constraints, x and y

The optimal path is either RS or LS. Several constrained points (x, y) are illustrated in figure 6-6 along with the RS and LS candidate to reach each point. Note that if (x, y) lies inside the circle coincident to either the L arc or the R arc than that solution is infeasible and the only solution to consider is the alternate solution.

6.5.5 2 active constraints, x or y and θ

In this case all six solution classes are valid (LSL,RSR,LSR,RSL,LRL,RLR). Because half of these are symmetric, we will only consider three of them here. Without loss of generality, let $y = y_a$ be the active constraint. For a given solution class, say LSL, the initial circle is fixed in the plane, and the final circle is fixed in y but free to slide along x. The value of x which minimizes the length of the path must also minimize one of the segments on the path. Thus we may evaluate the (up to) three values of x that minimize each segment, and add those results to the list of $\mathbf{x}^*$ candidates.

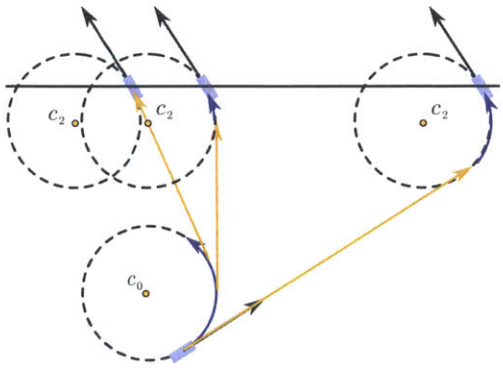

Figure 6-7: LSL solutions for two active constraints including θ. There are up to three cases. Each case to consider minimizes one of the three segments.

Figure 6-7 shows the three solutions for LSL minimizing the final

L, the S, and the initial L segments (in order from left to right). Note that if θ_0 is such that it points away from the constraint, the solution minimizing the initial L is infeasible. Likewise if θ_1 points towards c_0, then the solution minimizing the final L is infeasible.

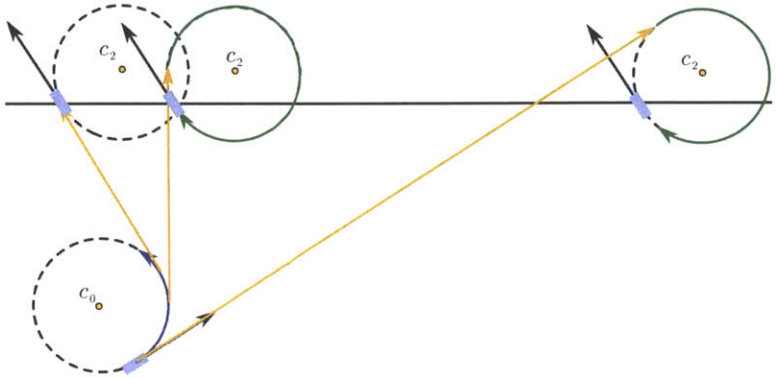

Figure 6-8: LSR solutions for two active constraints including θ. There are up to three cases. Each case to consider minimizes one of the three segments.

Figure 6-8 shows the three solutions for LSR minimizing the final R, the S, and the initial L segments (in order from left to right). Note that, as with LSL, if θ_0 is such that it points away from the constraint, the solution minimizing the initial L is infeasible. Likewise if θ_1 points towards c_0, then the solution minimizing the final L is infeasible.

Figure 6-9 shows the three solutions for LRL minimizing the initial L, the R, and the final L segments (in order from left to right). Note that if y_a is sufficiently far from the query one or more of the solutions may be infeasible.

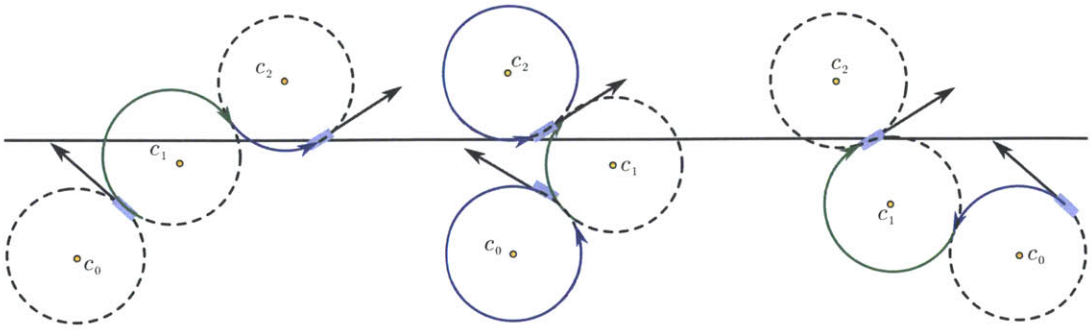

Figure 6-9: LRL solutions for two active constraints including θ. There are up to three cases. Each case to consider minimizes one of the three segments.

6.5.6 3 active constraints

If there are three active constraints then the state is fully constrained and the candidate solution is merely the Dubins shortest path to that state. All such candidates are feasible.

6.5.7 Experiments

Figure 6-10 illustrates wall-clock query time of the Dubins *kd*-tree implemented as described in this section. Points are drawn uniformly from within a square in the plane with side length 100. The Dubins vehicle is given a turning rate such that its minimum turning radius is 1. The results shown are averaged over 30 queries, selected uniformly at random from the state space. We see that, despite the large number of evaluations required to expand all possible solution paths, the *kd*-tree proves to be an effective index. There is indeed overhead in using the *kd*-tree as it is initially slower than a brute force search, but we see that it becomes faster for database sizes larger than about three thousand points.

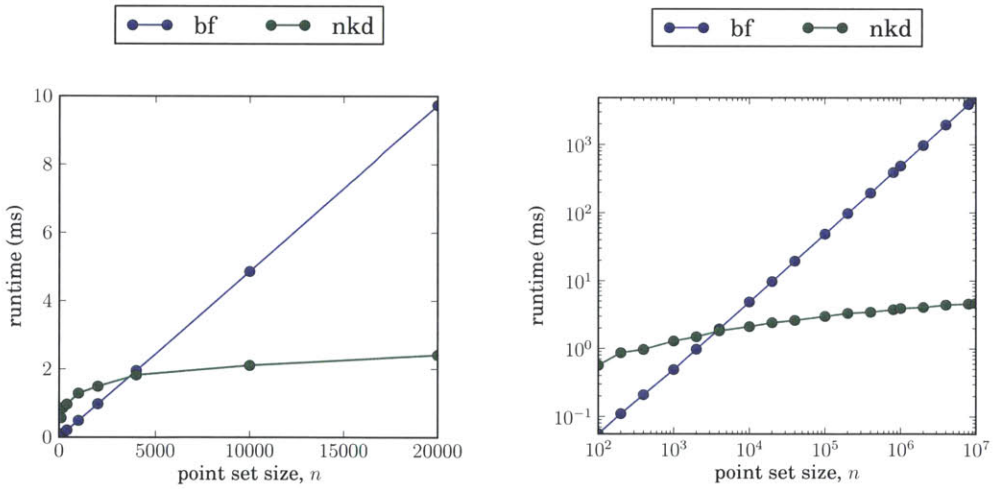

Figure 6-10: Runtime for 10-nearest neighbor queries in $\mathbb{R}^2 \times S^1$ for a Dubins vehicle. (left) is in a linear scale and (right) is log-log.

Appendix A

GJK algorithm

The Gilbert, Johnson, Keerthi (GJK) algorithm [29] is a popular efficient algorithm for discrete collision checking used in many collision checking implementations [21, 26, 73, 97]. Given two convex compact subsets of $\mathbb{R}^n$ the algorithm determines if the set intersection is non-empty.

The algorithm is straightforward to implement, and requires only an efficient implementation of a *support function*. This appendix briefly reviews the geometric interpretation of the algorithm.

Let $\mathbf{X}_A, \mathbf{X}_B \subset \mathbb{R}^n$ be two compact sets. The Minkowski sum $\mathbf{X}_{A+B} = \mathbf{X}_A \oplus \mathbf{X}_B$ is also a compact set defined as:

$$\mathbf{X}_A \oplus \mathbf{X}_B \triangleq \{\mathbf{x} \,|\, \exists\, \mathbf{x}_A \in \mathbf{X}_A, \mathbf{x}_B \in \mathbf{X}_B \; \mathbf{x} = \mathbf{x}_A + \mathbf{x}_B\}$$

We use the notation $\mathbf{X}_{A-B} = \mathbf{X}_A \ominus \mathbf{X}_B$ to denote the Minkowski sum of $\mathbf{X}_A$ with the algebraic inverse of $\mathbf{X}_B$:

$$\mathbf{X}_A \ominus \mathbf{X}_B \triangleq \{\mathbf{x} \,|\, \exists\, \mathbf{x}_A \in \mathbf{X}_A, \mathbf{x}_B \in \mathbf{X}_B \; \mathbf{x} = \mathbf{x}_A - \mathbf{x}_B\}$$

Lemma 14. *If $\mathbf{X}_A$ and $\mathbf{X}_B$ have non-empty intersection, then the Minkowski sum $\mathbf{X}_A + \mathbf{X}_B$ contains the origin.*

Proof. Let $\mathbf{x} \in \mathbf{X}_A$, $\mathbf{x} \in \mathbf{X}_B$ be a point in the non-empty intersection of $\mathbf{X}_A$ and $\mathbf{X}_B$. Then, by definition $0 = \mathbf{x} - \mathbf{x} \in \mathbf{X}_A \ominus \mathbf{X}_B$. □

The GJK algorithm relies on this fact and iteratively searches the $\mathbf{X}_A \ominus \mathbf{X}_B$ for the origin. If it can certify that the origin lies outside the Minkowski sum, then the two sets have an empty intersection. Likewise

if the origin is found inside the Minkowski sum, then the two sets have a non-empty intersection.

However, rather than explicitly computing a representation of the Minkowski sum the GJK algorithm relies on a support function to return the surface point in a particular search direction. Let $\mathbf{v} \subset \mathbb{R}^n$, $\|\mathbf{v}\| = 1$ be a search direction. The support function $\texttt{support}\,(\mathbf{X}_A, \mathbf{X}_B, \mathbf{v})$ evaluates to the maximal point of the Minkowski sum in the direction of $\mathbf{v}$, by returning the maximal point in $\mathbf{X}_A$ and $-\mathbf{X}_B$:

$$\texttt{support}_{\texttt{A}\ominus\texttt{B}}\,(\mathbf{v}) \triangleq \underset{(\mathbf{x}_A,\mathbf{x}_B)\in\mathbf{X}_A\times\mathbf{X}_B}{\operatorname{argmax}} (\mathbf{x}_A - \mathbf{x}_B)\cdot\mathbf{v}$$

Efficient implementations of support functions exist for many geometric primitives as well as triangular meshes commonly used in 3D graphics.

The GJK algorithm, given in algorithm A.1 attempts to discover a (possibly degenerate) simplex interior to the Minkowski sum which contains the origin. We define a simplex as the convex hull of a set of $k \leq n+1$ points in $\mathbb{R}^n$.

```
1  v ← −(x_a − x_b) ;
2  S ← {(x_a − x_b)} ;
3  while true do
4      (x_a, x_b) ← support_{A⊖B}(v) ;
5      if (x_a − x_b) · v < 0 then
6          return (false, x_a, x_b) ;
7      S ← S ∪ (x_a − x_b) ;
8      if containsOrigin(S) then
9          return true
10     else
11         (v, S) ← advanceSimplex(S)
```

Algorithm A.1: $\texttt{gjk}\,(\texttt{support}_{\texttt{A}\ominus\texttt{B}}, \mathbf{x}_a, \mathbf{x}_b)$ Returns `true` if $\mathbf{X}_A \cap \mathbf{X}_B \neq \emptyset$. Otherwise returns `false` along with the last point pair expanded in the search.

The $\texttt{containsOrigin}\,(S)$ procedure returns true if the convex hull of the point set S contains the origin. The `advanceSimplex` procedure finds the point on the (closed) convex hull of S closes to the origin. It then returns in S the set of points defining the sub-simplex in which

the closes point lies (i.e., a point, line segment, triangle, etc.), as well as a new search vector $\mathbf{v}$ in the direction of the origin from S.

The terminating criteria on line 5 may be interpreted as follows: $\mathbf{x}_{ab} = \mathbf{x}_a - \mathbf{x}_b$ is a maximal point on the boundary of the Minkowski sum in the direction of $\mathbf{v}$. The entire set $\mathbf{X}_A \ominus \mathbf{X}_B$ lies on one side of the hyperplane coincident to $\mathbf{x}_{ab}$ and orthogonal to $\mathbf{v}$. If $\mathbf{x}_{ab} \cdot \mathbf{v} < 0$ then the origin lies on the other side of the hyperplane, and therefore cannot lie inside the $\mathbf{X}_A \ominus \mathbf{X}_B$.

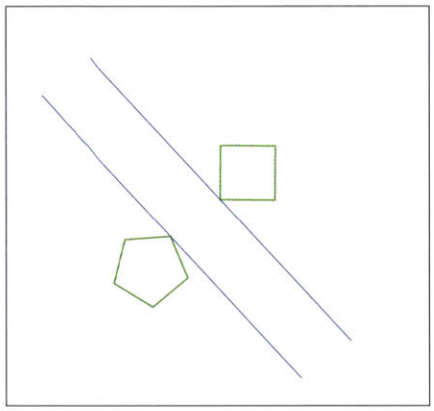

Figure A-1: Supporting hyperplanes of the GJK output for a pair of separated polygons

The output of GJK also provides a conservative estimate of the distance between the two volumes. Points $\mathbf{x}_a$ and $\mathbf{x}_b$ lie on the surface of $\mathbf{X}_A$ and $\mathbf{X}_B$ respectively. Furthermore they are the furthest points in the direction of $\mathbf{v}$ and $-\mathbf{v}$ respectively. The hyperplane normal to $\mathbf{v}$ and coincident to $\mathbf{x}_a$, and the hyperplane normal to $\mathbf{v}$ and coincident to $\mathbf{x}_b$ are separated by a distance of $(\mathbf{x}_b - \mathbf{x}_a) \cdot \mathbf{v}$. Furthermore, the space between them contains no point of $\mathbf{X}_A$ or $\mathbf{X}_B$. Thus $d(\mathbf{X}_A, \mathbf{X}_B) \geq (\mathbf{x}_b - \mathbf{x}_a) \cdot \mathbf{v}$.

Appendix B

Fast indexing for support functions in GJK

In this chapter we demonstrate how the support function of triangulation meshes in 3D and polygons in 2D implicitly define a lower dimensional Voronoi diagram which may be used as a fast index for the `support` procedure in GJK.

Consider first a convex polygon in 2D. For a given face, any point along that face is maximal in the direction of the face normal. Consider a pair of adjacent faces and the vertex that joins them. For any search direction in the range bounded by the angles of the two face normals, the vertex is the single maximal point in that direction.

Thus the angle of face normals divide up the unit circle into regions for which a particular vertex is the support in that direction, as illustrated in figure B-1. This emits a simple index for support vector searches: find the angle of each face normal and store them in a sorted array. Given a search direction, an $O(\log n)$ binary search will return the interval containing the query, and thus the maximal vertex in that direction.

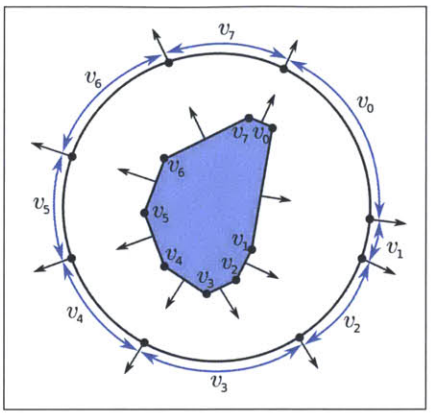

Figure B-1: Support set Voronoi diagram for a polygon in 2D.

Note the similarity between this index and a Voronoi diagram on a circle. If a search vector lies in the support region of a particular vertex this is analogous to a query point falling within the Voronoi cell of a site. This similarity readily extends to three dimensions as illustrated in figure B-2.

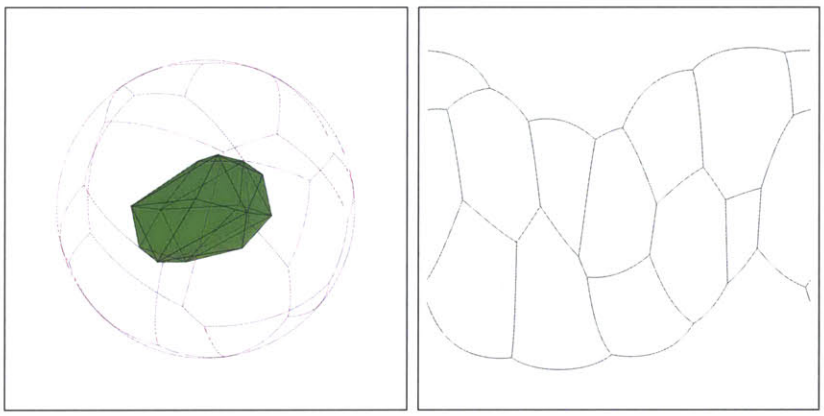

Figure B-2: Support set Voronoi diagram for a polygon in 3D.

In three dimensions the generator site for each cell in the tesselation is a vertex of the object. Vertices of the tesselation occur at the intersection of the face normals with S^2. The edges of the tesselation are arcs on S^2 joining two vertices of the tesselation that correspond to adjacent faces in the object.

Because the direction index is equivalent to a 2D Voronoi diagram we may utilize the theory of Voronoi diagrams to create an efficient search algorithm: a tesellation walk takes $O\left(\sqrt{n}\right)$ time, a hierarchy of Voronoi diagrams takes $O\left(\log^2 n\right)$ time, and a persistent search tree

of the Voronoi diagram can be searched in $O(\log n)$ time (which is optimal).

Bibliography

[1] N Amato and G Song. Using motion planning to study protein folding pathways. *Journal of Computational Biology*, 9(2):149–168, March 2004.

[2] Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. Obprm: an obstacle-based prm for 3d workspaces. In *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, pages 155–168, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *COMMUNICATIONS OF THE ACM*, 51(1):117, 2008.

[4] Alexandr Andoni, Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In G. Shakhnarovich, T. Darrell, and P. Indyk, editors, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press, 2006.

[5] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 459 –468, oct. 2006.

[6] Franz Aurenhammer. Voronoi diagrams — a survey of a fundamental data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[7] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, December 1996.

[8] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[9] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.

[10] Jon Louis Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, April 1980.

[11] P. Bessiere, J.M. Ahuactzin, E.G. Talbi, and E. Mazer. The ariadne's clew algorithm: global planning with local methods. In *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 2, pages 1373–1380. IEEE, 1993.

[12] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *siam Journal on computing*, 15:364, 1986.

[13] V. Boor, M. H. Overmars, and A. F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *IEEE Int. Conf. on Robotics and Automation*, pages 1018–1023, 1999.

[14] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[15] M.S. Branicky, S.M. LaValle, K. Olson, and Libo Yang. Quasi-randomized path planning. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1481–1487 vol.2, 2001.

[16] Kevin Q. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223 - 228, 1979.

[17] J. Canny. *The complexity of robot motion planning.* The MIT Press, 1988.

[18] John Canny. Collision detection for moving polyhedra. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(2):200 –209, march 1986.

[19] Kenneth L. Clarkson, Kurt Mehlhorn, and Raimund Seidel. Four results on randomized incremental constructions. *Computational Geometry*, 3(4):185 - 212, 1993.

[20] J. Cortes, T. Simeon, V. Ruiz de Angulo, D. Guieysse, M. Remaud-Simeon, and V. Tran. A path planning approach for computing large-amplitude motions of flexible molecules. *Bioinformatics*, 21(1):116–125, June 2005.

[21] E Coumans. Bullet physics library, 2006.

[22] J. Culberson. Sokoban is pspace-complete. In *Fun With Algorithms*, volume 4, pages 65–76. Citeseer, 1999.

[23] O. Devillers. The delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13:163–180, 2002.

[24] R.A. Dwyer. The expected number of k-faces of a voronoi diagram. *Computers & Mathematics with Applications*, 26(5):13 – 19, 1993.

[25] Rex a. Dwyer. Higher-dimensional voronoi diagrams in linear expected time. *Discrete and Computational Geometry*, 6(1):343–367, December 1991.

[26] Daniel Fisher. libccd, 2012.

[27] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[28] Roland Geraerts and Mark H. Overmars. A comparative study of probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics*, pages 43–57, 2002.

[29] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, 4(2):193 –203, apr 1988.

[30] W.R. Gilks and P. Wild. Adaptive rejection sampling for gibbs sampling. *Applied Statistics*, pages 337–348, 1992.

[31] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases*, pages 518–529, 1999.

[32] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 1718 –1723 vol.3, may 1990.

[33] J.E. Goodman and J. O'Rourke. *Handbook of discrete and computational geometry*. CRC press, 2004.

[34] N.K. Govindaraju, M.C. Lin, and D. Manocha. Quick-cullide: fast inter- and intra-object collision culling using graphics hardware. In *Virtual Reality, 2005. Proceedings. VR 2005. IEEE*, pages 59 –66, march 2005.

[35] N.K. Govindaraju, M.C. Lin, and D. Manocha. Fast and reliable collision culling using graphics hardware. *Visualization and Computer Graphics, IEEE Transactions on*, 12(2):143 –154, march-april 2006.

[36] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[37] Martin Held. Vroni: An engineering approach to the reliable and efficient computation of voronoi diagrams of points and line segments. *Computational Geometry*, 18(2):95 – 123, 2001.

[38] D. G. Hook and P. R. McAree. Using sturm sequences to bracket real roots of polynomial equations. In Andrew S. Glassner, editor, *Graphics gems*, pages 416–422. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[39] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[40] Qiming Hou, Xin Sun, Kun Zhou, C. Lauterbach, and D. Manocha. Memory-scalable gpu spatial hierarchy construction. *Visualization and Computer Graphics, IEEE Transactions on*, 17(4):466 –474, april 2011.

[41] D. Hsu, Tingting Jiang, J. Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 4420 – 4426 vol.3, sept. 2003.

[42] D. Hsu, J.C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2719–2726. IEEE, 1997.

[43] David Hsu, Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Stephen Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective*, WAFR '98, pages 141–153, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[44] P. Jimnez, F. Thomas, and C. Torras. 3d collision detection: a survey. *Computers & Graphics*, 25(2):269 – 285, 2001.

[45] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30(7):846–894, June 2011.

[46] Menelaos Karavelas and Mariette Yvinec. The voronoi diagram of planar convex objects. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003*, volume 2832 of *Lecture Notes in Computer Science*, pages 337–348. Springer Berlin / Heidelberg, 2003.

[47] L. Kavan and J. ra. Fast collision detection for skeletally deformable models. *Computer Graphics Forum*, 24(3):363–372, 2005.

[48] L. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on ...*, January 1996.

[49] Byungmoon Kim and Jarek Rossignac. Collision prediction for polyhedra under screw motions. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM '03, pages 4–10, New York, NY, USA, 2003. ACM.

[50] David Kirkpatrick, Jack Snoeyink, and Bettina Speckmann. Kinetic collision detection for simple polygons. In *Proceedings of the sixteenth annual symposium on Computational geometry*, SCG '00, pages 322–330, New York, NY, USA, 2000. ACM.

[51] Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of abstract voronoi diagrams. *Computational Geometry*, 3(3):157 – 184, 1993.

[52] J. Latombe. Motion planning: A journey of molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, September 2007.

[53] Jean-Claude Latombe. *Robot motion planning*. Kluwer Academic Publishers, 1991.

[54] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast bvh construction on gpus. *Computer Graphics Forum*, 28(2):375–384, 2009.

[55] C. Lauterbach, Q. Mo, and D. Manocha. gproximity: Hierarchical gpu-based operations for collision and distance queries. *Computer Graphics Forum*, 29(2):419–428, 2010.

[56] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[57] S. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[58] S.M. LaValle and J.J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and computational robotics: new directions: the fourth Workshop on the Algorithmic Foundations of Robotics*, page 293. AK Peters, Ltd., 2001.

[59] P. L'Ecuyer. Tables of linear congruential generators of different sizes and good lattice structure. *Mathematics of Computation*, 68(225):249–260, 1999.

[60] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, SPM '07, pages 121–131, New York, NY, USA, 2007. ACM.

[61] M. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, University of California at Berkeley, 1993.

[62] Stephen R. Lindemann and Steven M. LaValle. Current issues in sampling-based motion planning. In Paolo Dario and Raja Chatila, editors, *Robotics Research*, volume 15 of *Springer Tracts in Advanced Robotics*, pages 36–54. Springer Berlin / Heidelberg, 2005. 10.1007/11008941_5.

[63] David B. Lomet and Betty Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM Trans. Database Syst.*, 15(4):625–658, December 1990.

[64] K. Mamou and F. Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3501 –3504, nov. 2009.

[65] G. Marsaglia and A. Zaman. A new class of random number generators. *The Annals of Applied Probability*, pages 462–480, 1991.

[66] George Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 7 2003.

[67] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[68] M. McAllister, D. Kirkpatrick, and J. Snoeyink. A compact piecewise-linear voronoi diagram for convex sites in the plane. *Discrete & Computational Geometry*, 15:73–105, 1996.

[69] T. Ohya, M. Iri, and K. Murota. Improvements of the incremental method for the voronoi diagram with computational comparison of various algorithms. *J. OPER. RES. SOC. JAPAN.*, 27(4):306–336, 1984.

[70] A. Okabe, B.N. Boots, K. Sugihara, and S.N. Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams.* Wiley & Sons Chichester., 1992.

[71] Stephen M. Omohundro. Five balltree construction algorithms. Technical report, University of California at Berkeley, 1989.

[72] J p. Laumond and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Journal of Advanced Robotics*, 14(6):477–493, 2000.

[73] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *IEEE Int. Conference on Robotics and Automation*, Minneapolis, Minnesota, 05/2012 2012.

[74] Jia Pan, C. Lauterbach, and D. Manocha. Efficient nearest-neighbor computation for gpu-based motion planning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2243 –2248, oct. 2010.

[75] Jia Pan and Dinesh Manocha. Fast gpu-based locality sensitive hashing for k-nearest neighbor computation. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 211–220, New York, NY, USA, 2011. ACM.

[76] Jia Pan and Dinesh Manocha. Gpu-based parallel collision detection for real-time motion planning. In David Hsu, Volkan Isler, Jean-Claude Latombe, and Ming Lin, editors, *Algorithmic Foundations of Robotics IX*, volume 68 of *Springer Tracts in Advanced Robotics*, pages 211–228. Springer Berlin / Heidelberg, 2011.

[77] Jia Pan and Dinesh Manocha. Gpu-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, 31(2):187–200, 2012.

[78] Jeff M. Phillips, L. E. Kavraki, and N. Bedrosian. Spacecraft rendezvous and docking with real-time randomized optimization.

In *AIAA (American Institute of Aeronautics and Astronautics) Guidance, Navigation and Control Conference*, Austin, TX, August 2003.

[79] J.M. Phillips, N. Bedrossian, and L.E. Kavraki. Guided expansive spaces trees: a search strategy for motion- and cost-constrained state spaces. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 4, pages 3968 – 3973 Vol.4, 26-may 1, 2004.

[80] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, February 1977.

[81] Octavian Procopiuc, Pankaj Agarwal, Lars Arge, and Jeffrey Vitter. Bkd-tree: A dynamic scalable kd-tree. In Thanasis Hadzilacos, Yannis Manolopoulos, John Roddick, and Yannis Theodoridis, editors, *Advances in Spatial and Temporal Databases*, volume 2750 of *Lecture Notes in Computer Science*, pages 46–65. Springer Berlin / Heidelberg, 2003.

[82] S. Redon, A. Kheddar, and S. Coquillart. An algebraic solution to the problem of collision detection for rigid polyhedral objects. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 4, pages 3733 –3738 vol.4, 2000.

[83] Stephane Redon, Young J. Kim, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection for articulated models. In *Proceedings of the ninth ACM symposium on Solid modeling and applications*, SM '04, pages 145–156, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[84] Stephane Redon, Ming C. Lin, Dinesh Manocha, and Young J. Kim. Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering*, 5(2):126–137, 2005.

[85] Stphane Redon, Abderrahmane Kheddar, and Sabine Coquillart. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–287, 2002.

[86] J.H. Reif. Complexity of the mover's problem and generalizations extended abstract. In *Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science*, pages 421–427, 1979.

[87] John T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, SIGMOD '81, pages 10–18, New York, NY, USA, 1981. ACM.

[88] Donald B Rubin. The bayesian bootstrap. *The annals of statistics*, 9(1):130–134, 1981.

[89] Mitul Saha, Jean-Claude Latombe, Yu-Chi Chang, and Friedrich Prinz. Finding narrow passages with probabilistic roadmaps: The small-step retraction method. *Autonomous Robots*, 19(3):301–319, December 2005.

[90] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[91] G. Sánchez and J.C. Latombe. On delaying collision checking in prm planning: Application to multi-robot coordination. *The International Journal of Robotics Research*, 21(1):5–26, 2002.

[92] Neil Sarnak and Robert E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, July 1986.

[93] Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe. Exact collision checking of robot paths. In Jean-Daniel Boissonnat, Joel Burdick, Ken Goldberg, and Seth Hutchinson, editors, *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics*, pages 25–42. Springer Berlin / Heidelberg, 2004.

[94] R Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, STOC '86, pages 404–413, New York, NY, USA, 1986. ACM.

[95] Raimund Seidel. Linear programming and convex hulls made easy. In *Proceedings of the sixth annual symposium on Computational geometry*, SCG '90, pages 211–215, New York, NY, USA, 1990. ACM.

[96] Mehdi Sharifzadeh and Cyrus Shahabi. Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *Proc. VLDB Endow.*, 3(1-2):1231–1242, September 2010.

[97] Russell Smith. Open dynamics engine, 2006.

[98] P. Soures and J. Boissonnat. Optimal trajectories for nonholonomic mobile robots. In J. Laumond, editor, *Robot Motion Planning and Control*, volume 229 of *Lecture Notes in Control and Information Sciences*, pages 93–170. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0036072.

[99] P. Stein. A note on the volume of a simplex. *The American Mathematical Monthly*, 73(3):pp. 299–301, 1966.

[100] Min Tang, Dinesh Manocha, and Ruofeng Tong. Mccd: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models*, 72(2):7 – 23, 2010.

[101] Xiang Tian and K. Benkrid. Mersenne twister random number generation on fpga, cpu and gpu. In *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, pages 460 –464, 29 2009-aug. 1 2009.

[102] A.O. Tokuta. Motion planning using binary space partitioning. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 86 –90 vol.1, nov 1991.

[103] Marc van Kreveld and Mark Overmars. Divided k-d trees. *Algorithmica*, 6:840–858, 1991.

[104] R. Wenger. Randomized quickhull. *Algorithmica*, 17:322–329, 1997.

[105] 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 *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1024 –1031 vol.2, 1999.

[106] A. Yershova and S.M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *Robotics, IEEE Transactions on*, 23(1):151–157, 2007.

[107] Xinyu Zhang, Stephane Redon, Minkyoung Lee, and Young J. Kim. Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Trans. Graph.*, 26(3), July 2007.