Journal name
():–
©The Author(s) 2010
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI:10.1177/1081286510367554
http://mms.sagepub.com

Efficient Collision Checking in Sampling-based Motion Planning via Safety Certificates

Joshua Bialkowski, Michael Otte*, Sertac Karaman, and Emilio Frazzoli
Massachusetts Institute of Technology, Cambridge MA 02139, USA

Abstract

Collision checking is considered to be the most expensive computational bottleneck in sampling-based motion planning algorithms. We introduce a simple procedure that theoretically eliminates this bottleneck and significantly reduces collision checking time in practice in several test scenarios. Whenever a point is collision checked the normal (expensive) way, we store a lower bound on that point's distance to the nearest obstacle. The latter is called a "safety certificate" and defines a region of the search space that is guaranteed to be collision free. New points may forgo collision-checking whenever they are located within a safety certificate of an old point. Testing the latter condition is accomplished during the nearest-neighbor search that is already part of most sampling-based motion planning algorithms. As more and more points are sampled, safety certificates asymptotically cover the search space and the amortized complexity of (normal, expensive) collision checking becomes negligible with respect to the overall runtime of sampling-based motion planning algorithms. Indeed, the expected fraction of points requiring a normal collision-check approaches zero, in the limit, as the total number of points approaches infinity. A number of extensions to the basic idea are presented. Experiments with a number of proof-of-concept implementations demonstrate that using safety certificates can improve the performance of sampling based motion planning algorithms in practice.

Keywords
sampling-based, motion planning, collision checking, safety certificate, robotics, autonomous systems

1. Introduction

A motion planning problem is, roughly speaking, the search for a connected set of collision-free configurations that begin at a start region (or configuration) and end within a desired goal region. Such problems arise in a multitude of autonomous robotic applications including: aerospace, underwater exploration, manufacturing, and warehouse management. The future success or failure of automated personal transportation and mainstream consumer robotics will depend, in a large part, on the ability to solve motion planning problems more quickly.

* Corresponding author; e-mail: ottemw@gmail.com

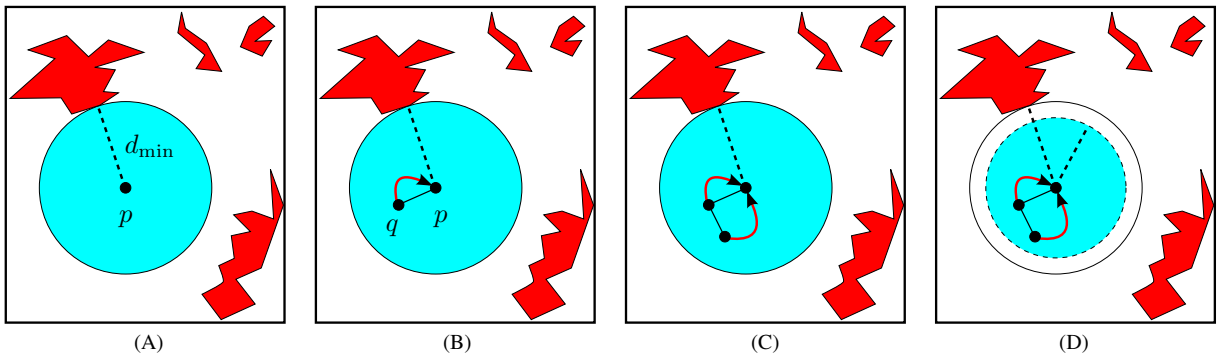

Fig. 1. (A) Collision checked nodes p store "safety certificates" (blue) defined by $d_{\min}$ (dashed black lines), a lower bound on the distance to the nearest obstacle (obstacles are red). (B) Future nodes, e.g., q, within a certificate can forgo collision checking. (D) Pointers (solid red arrows) are maintained to certifying nodes. (D) A lower bound on the distance to the nearest obstacle can also be used.

Finding a safe motion plan fundamentally involves comparing robot state values with obstacle locations. Yet, in general, these two things are defined in two different spaces, i.e., the configuration space and the workspace, respectively. Different classes of motion planning algorithms address this discrepancy in different ways.

Exact motion planning algorithms assume that an exact obstacle representation can be constructed in the configuration space (see, for instance, those in LaValle (2006)). While exact algorithms exist that are complete and optimal, the practical transformation of obstacles from the workspace into the configuration space is often intractable or even theoretically impossible.

Sampling-based motion planning algorithms are a more tractable alternative. These algorithms iteratively construct a graph of safe movement by randomly sampling the configuration space and then mapping points/trajectories from the configuration space into robotic positions in the workspace for collision checking vs. obstacles. The resulting algorithms trade absolute completeness/optimality for probabilistic completeness and asymptotic optimality.

Sampling-based motion planning algorithms are the focus of the current paper. Their main components include:

- A sampling scheme that provides a sequence of points from the configuration space.
- Collision-checking functions that determine if a configuration space point or trajectory is in collision with an obstacle, given a workspace obstacle set and a mapping function from configuration space to workspace.
- A proximity search function that returns "neighbors" to a point in the configuration space, given a point set in the configuration space.
- A local planning function that returns a configuration space trajectory between two given points.

Collision checking is widely considered to be the primary computational bottleneck of sampling based motion planning (see, e.g., LaValle (2006)). Although an individual collision check has runtime complexity that is upper-bounded by a constant[1], the constant is often overwhelmingly large in practice. *Our main contribution is to show that this does not need to be the case.* We introduce a novel collision checking technique that has negligible amortized complexity vs. the nearest-neighbor searches that are already at the core of sampling-based motion planning algorithms. As a consequence, nearest-neighbor searches become the main practical determinant of runtime—as complexity analysis suggests they should (e.g., see Karaman and Frazzoli (2011)).

Traditionally, sampling based motion planning algorithms have considered collision checking as a "Boolean black box" subroutine that returns either '`True`' or '`False`' depending on if a query point is in collision with obstacles or not,

[1]More formally: an individual collision check has runtime complexity that is upper-bounded by a constant, where the latter constant depends on both the geometry of the planning instance being solved and the particular collision checking algorithm being used. For example, the constant may be a function of the depth and branching factor of a tree-based bounding volume hierarchy, multiplied by the effort required to solve a finite number of equations/inequalities at each level of the hierarchy.

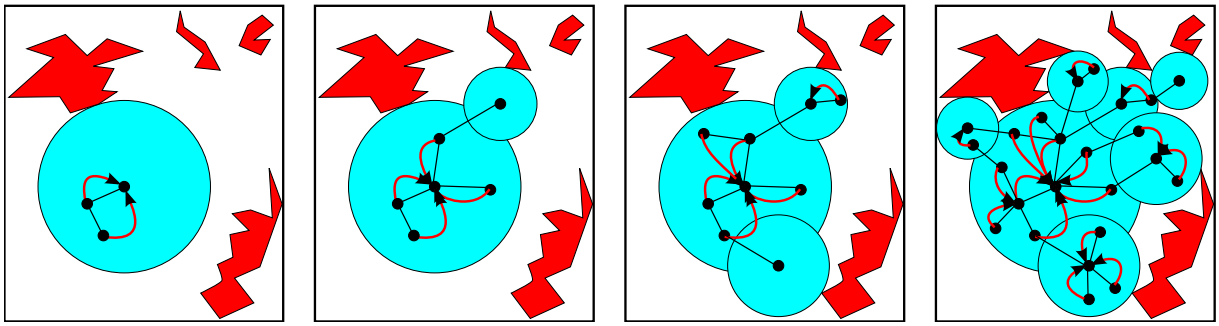

Fig. 2. Certificates (blue discs) asymptotically cover the space as graph size increases toward infinity. The ratio of collision checks vs. graph size (nodes) approaches zero in the limit as graph size approaches infinity.

respectively. We place a stronger requirement on the collision-checking procedure, and assume that it returns $d_{\min}$ which is a lower bound on the minimum distance to the obstacle set, see Figure 1. Although computing $d_{\min}$ may sometimes be harder, by up to a constant factor[2], than computing the standard binary '`True`'/'`False`' check, the extra information provided by $d_{\min}$ allows us to eliminate a large fraction of subsequent collision checks.

Each collision-checked point p remembers $d_{\min}$, which defines a "safety certificate" near p—i.e., a sub-set space containing p that is known to be collision free. New samples q may forgo collision-checking whenever they are located within a safety certificate, i.e., when $\|p - q\| < d_{\min}$. Certificates tend to cover more and more of the space as graph size increases, and thus the probability that a new sample will require a collision-check decreases, see Figure 2. In Section 5 we show that the amortized computational complexity due to collision checking converges to zero, in the limit, as the number of samples approaches infinity (this trend can be observed in Figure 3).

It is important to note that $d_{\min}$ may be defined with respect to *either* the configuration space or the workspace; thus, it is possible to use certificates in either space, assuming $d_{\min}$ can be calculated. In practice, the convenience and utility of using certificates in one space or the other depends on the implementation details of a particular motion planning system, e.g., the geometric constructs used to model the robot and the obstacles. We perform experiments using either type of certificate in Section 8. However, the implementations that we use are intended merely as proofs-of-concept that the general idea can be applied in a variety of ways, and are not designed for use with any specific existing motion planning packages. We hope that our results will encourage the developers of future motion planning systems to consider certificate representations alongside other geometric and topological factors when choosing how to model robots and obstacles.

The rest of this paper is organized as follows: Background and related work are discussed in Section 2. Section 3 contains nomenclature and problem statements. The basic state-space certificate algorithm is presented in Section 4, and its analysis is presented in Section 5. In Section 6 we demonstrate how workspace certificates can be implemented in the case where robots and obstacles are modeled as sets of polytopes. In Section 7 we demonstrate how symmetries within the state-space of a multi-robot problem can be exploited to achieve performance gains. Experiments are presented in Section 8 and Section 9 contains a discussion of the results. Conclusions are presented in Section 10.

[2]More precisely, the increase in effort is upper bounded by a finite constant factor that depends on both the geometry of the planning instance being solved and the obstacle representation. This is due to the fact that computing $d_{\min}$ usually requires solving multiple equations per obstacle surface (one per the surface itself and one for each boundary element of the surface — e.g., faces and edges, respectively, in the case of polytope), while computing a binary '`True`'/'`False`' check can often be accomplished by solving a single inequality per surface.

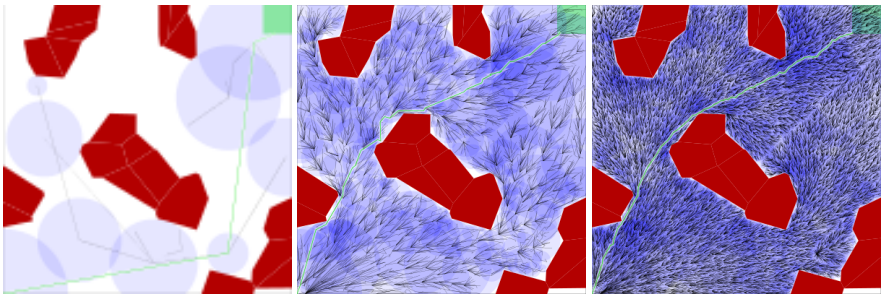

Fig. 3. Illustration of the certificate covering for an implementation of RRT* for a planar point robot. As the number of samples increase (images from left to right) the likelihood of sampling from outside the certificates goes down.

2. Background and Related Work

Most sampling-based motion planning algorithms can be classified along two different spectra: (1) single-query vs. multi-query planning and (2) feasible-path vs. shortest-path planning.

Single-query algorithms are used to find a single motion plan through a particular environment, while multi-query algorithms are used when many motion plans are expected to be found in the same environment. The Rapidly-exploring Random Tree (RRT) by LaValle and Kuffner (2001) and the Probabilistic RoadMap (PRM) by Kavraki et al. (1996), are popular single- and multi-query algorithms, respectively.

Feasible-path planning algorithms are designed to find a valid path between the start state and the goal set, while shortest-path planning algorithms attempt to find the *shortest* valid path with respect to a predefined metric over the search space. The aforementioned RRT and PRM are both feasible-planning algorithms, while PRM* and RRT* are shortest-path planning algorithms (Karaman and Frazzoli, 2011).

Many variations of these basic ideas exist, each designed to improve performance in certain cases or to provide additional performance guarantees. For example, Hsu et al. (1998) and Boor et al. (1999) bias sampling to occur in narrow passages and near obstacle boundaries, respectively, and Bohlin and Kavraki (2000) delay collision checks until needed (LazyPRM). Karaman and Frazzoli (2011) provide asymptotic optimality guarantees on path quality with PRM* and RRT*. RRT# by Arslan and Tsiotras (2013) maintains graph consistency (the idea that cost decreases should be propagated through the search-graph) to improve practical convergence toward the optimal solution. Otte and Frazzoli (2015) formally prove that maintaining graph consistency leads to faster convergence in theory (i.e., in addition to practice); they also show that better performance can be achieved in the context of replanning by settling for ϵ-consistency in RRT$^{\mathrm{X}}$.

Much of the literature on collision checking focuses on *static collision checking*—determining the safety of a single point. More recent work has investigated the more difficult problem of *continuous collision checking*—determining the safety of a continuous trajectory between two points. In general, collision checking requires both *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 comprehensive survey appears in Jiménez et al. (2001).

Discrete interference testing between two convex objects is often implemented using the GJK algorithm by Gilbert et al. (1988). Efficient exact interference testing in 3d for a convex polytope undergoing smooth motion among convex polytope obstacles is given by Canny (1986). 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 by Redon et al. (2000), where "small" is defined as being application dependent and valid in cases where the approximation $\tan(\theta t/2)/\tan(\theta 2) \approx t$, for time t and rotation θ, is valid. For the case that the motion can be approximated by a sequence of screw motions (infinitesimal

translations/rotations), a method is given by Kim and Rossignac (2003). A combination of the two with the addition of conservative bounds given by interval arithmetic is given by Redon et al. (2002).

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 by Schwarzer et al. (2004). Methods based on conservative certificates and interval arithmetic are described in Redon et al. (2004, 2005). An extension using Taylor models to approximate incremental motion is described by Zhang et al. (2007). A method based on refitting bounding spheres is given by Kavan and Žára (2005).

Data structures used as spatial indexes generally fall under the category of spatial decompositions and bounding volume hierarchies (Samet, 2006). 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 (McAllister et al., 1996). In the case where the system and obstacles are polygons the Voronoi diagram of line segments can be used (Held, 2001; Karavelas and Yvinec, 2003).

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) by Tokuta (1991). Bounding volume hierarchies (BVHs), especially those based on R-trees (Guttman, 1984) 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 Sharifzadeh and Shahabi (2010). Voronoi diagrams, BSPs and BVHs are shown to have $O(\log n_{\text{obs}})$ expected search time for a static collision query, where n_{obs} is the size of the obstacle representation (usually number of vertices or faces of the obstacle mesh) Samet (2006). The latter is a consequence of requiring $O(\log n_{\text{obs}})$ expected interference tests per static collision query.

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. However, the low level mathematical operations required for interference testing and index searching tend to be elementary for hardware arithmetic units. Graphics co-processors (GPUs), designed with reduced capability but many more logical processors, are particularly well suited to solving the collision checking problem.

Govindaraju et al. (2005) and Govindaraju et al. (2006) present bounding volume strategies designed for GPUs. Another BVH construction technique is presented by Lauterbach et al. (2009). Collision checking based on the latter appears in Lauterbach et al. (2010). A parallel method based on maintenance of the colliding-front of the bounding volume tree is presented by Tang et al. (2010). Hou et al. (2011) provide an alternative to breadth-first construction order for GPU bounding volume hierarchies. A modern general implementation for production use is described by Pan and Manocha (2011) Pan et al. (2012) and Pan and Manocha (2012).

Workspace certificates are closely related to collision checking algorithms based on conservative advancement such as kinetic data structures (Basch et al., 1999; Kirkpatrick et al., 2000) and adaptive bisection (Redon et al., 2002; Schwarzer et al., 2004). The latter 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. These, in turn, are related to earlier methods that efficiently update pairwise collision distance information as objects move vs. time (Ponamgi et al., 1997; Mirtich, 1998).

Geometric coverings have also been used for the purposes of feasible control. In Yang and LaValle (2000) a ball-covering over the control space is used to construct a global control policy out of a set of local control policies (one per ball). The method is extended in Yang and LaValle (2002) and Yang and LaValle (2004) to achieve better speed and sampling performance. While Yang and LaValle (2000, 2002, 2004) focus on achieving a valid control policy (see Yang and LaValle (2004) for an insightful discussion on formulating robot kinematics or use with ball-decompositions), these

methods are concerned with finding a feasible solution given that all new points sampled within a previously defined ball are rejected. In contrast, our method can be used with a wide variety of sampling-based algorithms, including those that solve the shortest path-planning problem; we also allow using non-spherical/cylindrical certificates.

Brock and Kavraki (2001) also use balls in the workspace to find a collision-free "tunnel" (sequence of collision free balls) that is then mapped into the configuration space where artificial potential fields are used to determine the path traveled by the robot. During the tunnel creation phase new points are sampled on the surface of existing balls, Yang and Brock (2013) extend this idea by sampling on the medial between obstacles on the surface of balls. Deits and Tedrake (2014) use ellipsoids of free space to rapidly determining valid step locations in the context of legged robotics.

Portions of the current paper have previously appeared in a number of other places, albeit in preliminary forms. Bialkowski et al. (2013d) presents the first iteration of the basic state-space method and has significant overlap with Section 5 of the current paper. Section 7 and, to a lesser extent, Section 8 of the current paper borrow heavily from a technical report (Bialkowski et al., 2013a) that was written to accompany a workshop poster (Bialkowski et al., 2013b). Preliminary versions of Sections 2 and 6 and portions of Sections 5 and 8 have also appeared in the PhD dissertation of the first author (Bialkowski, 2013).

We have also presented an unrelated approach to collision-checking avoidance in which samples are drawn from a distribution that converges to uniform sampling over the free-space (Bialkowski et al., 2013c). While both Bialkowski et al. (2013c) and the current paper attempt to minimize collision checking, they do so via different modalities. In particular, Bialkowski et al. (2013c) seeks to avoid drawing samples from the obstacle space by updating the sampling distribution based on previous information, while the current paper leverages information from previous collision checks to determine when a new sample is collision free. Combining the two ideas is an interesting direction for future work, but is beyond the scope of the current paper.

3. Nomenclature and Problem Statements

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. We assume that X is a metric space, and that $\|\cdot\|$ is a metric over X (e.g., Euclidean distance—see LaValle and Kuffner (2001) for alternative choices).

ω is an infinite sample sequence such that $x \in X$ for all $x \in \omega$. We shall use S to denote a set of points, in general. $|\cdot| = \text{card}\left(\cdot\right)$ denotes the cardinality of '$\cdot$' and $\mathscr{L}(\cdot)$ denotes the Lebesgue measure of '$\cdot$' in $\mathbb{R}^D$.

X contains one dimension per each of the system's degrees-of-freedom (e.g., position, rotation, joint-angles, etc.). It is often the case that the configuration space of the system has more dimensions 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 $x \in X$, then $w\left(x\right) \subset \mathcal{W}$ is the set of points in the workspace which are occupied by the system at configuration x. If the obstacles are specified in terms of the workspace, $O_i \subset \mathcal{W}$, then free space is defined as $X_{\text{free}} = \{x \in X \mid w\left(x\right) \cap X_{\text{obs}} = \emptyset\}$.

Given a function $\sigma : [0, 1] \to \mathbb{R}^d$, *total variation* is defined as follows Karaman and Frazzoli (2011):

$$\text{TV}\left(\sigma\right) \triangleq \sup_{n \in \mathbb{N}, 0 = t_0 < t_1 < \ldots < t_n = 1} \sum_{i=1}^{n} \left|\sigma\left(t_i\right) - \sigma\left(t_{i-1}\right)\right|.$$

We say a function σ has *bounded variation* if $\text{TV}\left(\sigma\right) < \infty$.

A *path* is defined to be a continuous function $\sigma : [0, 1] \to \mathbb{R}^D$ with bounded variation. 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., $\sigma\left(0\right) \in X_{\text{start}}$ and $\sigma\left(1\right) \in X_{\text{goal}}$.

For the sake of our discussion, we will assume that sampling-based algorithms return a sequence of graphs. Let $G_n(\omega) = (V_n(\omega), E_n(\omega))$ denote the output of the algorithm, given the first n elements of a sample sequence ω, where $G_n(\omega)$ is the search graph defined by a set of nodes $V_n(\omega)$ and the set of edges between them $E_n(\omega)$.

$I_{\mathrm{path}}(n)$ is an indicator random variable for the event that the nth sample requires a "normal" collision check. $p_{\mathrm{cc}}(n) = \mathbb{E}[I(n)]$ is the related expectation.

The computational complexity of a sampling-based algorithm can be decomposed in terms of the complexity of its primitive operations. c_{sample}, c_{plan}, $c_{\mathrm{cc}}(X_{\mathrm{obs}})$, and $c_{\mathrm{path}}(X_{\mathrm{obs}})$ are upper-bounds on the complexity of drawing a sample, calculating a trajectory, collision checking a point, and collision checking a trajectory, respectively. The former two are constants, while the latter two are function of the obstacle representation.

3.1. Feasible Motion Planning

The *feasible motion planning problem* is to find a feasible path if one exists or to determine that one does not. Formally, given X_{free}, X_{start}, and X_{goal}, find $\sigma : [0,1] \to X_{\mathrm{free}}$ subject to $\sigma \in C^0$, $\mathrm{TV}(\sigma) < \infty$, $\sigma(0) \in X_{\mathrm{start}}$, and $\sigma(1) \in X_{\mathrm{goal}}$, where $\sigma = \emptyset$ if and only if all constraints are not met.

3.2. Optimal Motion Planning

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., $J(\sigma) = 0$ if and only if $\sigma(\tau) = \sigma(0)$, $\forall \tau \in (0,1]$, Karaman and Frazzoli (2011).

The *optimal motion planning problem* is to find a feasible path with minimum cost provided one exists, or determine if no feasible path exists. Formally, given X_{free}, X_{start}, X_{goal}, and J, find $\underset{\sigma}{\operatorname{argmin}}\ (J(\sigma))$ subject to $\sigma : [0,1] \to X_{\mathrm{free}}$, $\sigma \in C^0$, $\mathrm{TV}(\sigma) < \infty$, $\sigma(0) \in X_{\mathrm{start}}$, and $\sigma(1) \in X_{\mathrm{goal}}$.

4. Certificate Algorithm

In this section we present the most basic version of the certificate method: that in which certificates are defined as balls in the configuration space. This version has the advantage of being simple while also introducing all of the machinery required for in-depth analysis (presented in Section 5). Although this basic method can be applied directly to a handful of simple motion planning problems, its main purpose is to provide a strong foundation from which an entire family of more practical implementations can be generalized. In practice, we expect that each of the latter will have additional complications, e.g., as a result of being tailor-made to solve a particular problem (for example, those in Sections 6 and 7) or for the sake of compatibility with an existing motion planning code-base. In this section we ignore these complications and focus on the heart of the general idea itself.

Primitive subroutine and data structures are formalized in Section 4.1, while the resulting augmented collision checking procedures are presented in Section 4.2. Versions of RRT and PRM* that use the augmented collision checking procedures are presented in Section 4.3.

4.1. Data Structures and Primitive Subroutines

Data Structure: Any version of the certificate method works by storing additional data within the graph data structure that is already used by most sampling-based algorithms. Let $AG = (V, E, S_{\mathrm{free}}, S_{\mathrm{obs}}, \mathrm{Dist})$ represent the "augmented graph" 5-tuple that we use for the most basic version of the idea presented here. $V \subset X_{\mathrm{free}}$ and $E \subset V \times V$ are the usual vertex and edge sets that are standard data fields in most sampling based motion planning algorithms. We additionally include the sets $S_{\mathrm{free}} \subset X_{\mathrm{free}}$ and $S_{\mathrm{obs}} \subset X_{\mathrm{obs}}$, which are sets of points for which an explicit collision check has been made. Finally,

we include the map $\mathrm{Dist} : S_{\mathrm{free}} \cup S_{\mathrm{obs}} \to \mathbb{R}_{\geq 0}$, which stores a lower-bound on minimum distance to the obstacle set for points in S_{free} (or to the free space for points in S_{obs}). In the simple case presented in this section, $\mathrm{Dist}(x)$ for $x \in X$ defines the radius of the ball that is used for the certificate.

V and E are initialized according to the particular sampling-based algorithm in use (e.g., RRT, PRM*, etc). S_{free} and S_{obs} are initialized as empty sets. In practice, and assuming that a node v is located at a point x for $x \in (S_{\mathrm{free}} \cup S_{\mathrm{obs}})$, it is convenient to store the results of $\mathrm{Dist}(x)$ as a sub-field of the data structure used to define v.

Sampling: ω is an infinite sequence of samples from X. The subroutine $\texttt{Sample}_i(\omega)$ returns the i-th element of ω. Formally, $\texttt{Sample} : \omega \mapsto \{\texttt{Sample}_i(\omega)\}_{i \in \mathbb{N}} \subset X$. We assume that all samples $x \in \omega$ are independent and identically distributed (i.i.d.). We also assume that samples are generated from a uniform distribution; however, our results naturally extend to any absolutely continuous distribution with density bounded away from zero on X.

Nearest, k-Nearest, and Near Neighbors: The subroutine $\texttt{Nearest}\,(S, x)$ returns the member of the point set S that is closest to x, i.e., $\texttt{Nearest} : (S, x) \mapsto s \in S$ such that

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

where $S \subset X$ by construction of the sampling based motion planning algorithm and $\mathrm{card}\,(S) < \infty$.

$\texttt{kNearest}(S, x, k)$ returns the k nearest-neighbors of x in S with respect to $\|\cdot\|$. That is, $\texttt{kNearest} : (S, x, k) \mapsto \{s_1, s_2, \ldots, s_k\}$. By convention $\texttt{kNearest}(S, x, k)$ returns S whenever $\mathrm{card}\,(S) < k$.

Similarly, $\texttt{Near}(S, x, r)$ returns the set S_{near} containing all members of S such that $s \in S \,|\, \|s - x\| \leq r$, where $r \in \mathbb{R}_{>0}$ is a positive real number. Formally, $\texttt{Near} : (S, x, r) \mapsto S_{\mathrm{near}} \subseteq S$ such that

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

Set Distance: The subroutine $\texttt{SetDistance}\,(S, x)$ returns a non-trivial lower bound on the minimum distance from a point x to S, i.e., for some $\alpha \in (0, 1]$. We assume that $S \subset X$ is a closed set and that the point $x \in X$. Formally,

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

In particular, $\texttt{SetDistance}\,(S, x)$ is used to determine the distance to obstacles, and thus the size of certificates. In practice, tighter bounds are desirable, assuming that they can be calculated efficiently. Additional discussion regarding α can be found in Section 5.

Segment Collision Test: The subroutine $\texttt{CFreePath}(x, y)$ returns `True` if the path segment between points x and y lies in X_{free}, otherwise it returns `False`. Formally,

$$\texttt{CFreePath}(x, y) \triangleq \begin{cases} \texttt{True}, & \text{if } [x, y] \cap X_{\mathrm{free}} = [x, y] \\ \texttt{False}, & \text{if } [x, y] \cap (X \setminus X_{\mathrm{free}}) \neq \emptyset \end{cases}$$

assuming that $x, y \in X$. In the most basic case of a single integrator holonomic system $[x, y]$ is a line segment, however in more complicated cases it may be some other continuous curve through X.

Certifying Node: It is advantageous for each node $x \in V$ to have access to the node that originally certified it as collision-free. In our pseudocode the subroutine $\texttt{CertifierOf}\,(x)$ returns the point y such that y certified x when the node located

at x was added to V. In practice, we expect that this will be implemented by simply storing a pointer to y directly in the data structure of the node at x.

Furthest Safe Point Along a Trajectory: The subroutine $\texttt{FarSafe}\,(x, y)$ returns the furthest point along the trajectory $[x, y]$ that is certified by the same certificate as x (without leaving and then re-entering that certificate). Formally, let $\sigma_{\mathrm{traj}} : [0, 1] \to [x, y]$ be the function that maps $[0, 1]$ to trajectory $[x, y]$. Let c be the point that certifies x. Recall that $\mathrm{Dist}(c)$ returns a bound on the distance between c and the obstacle set (as defined in Section 4.1). A point x' that is certified by c has the property $\|x' - c\| \leq \mathrm{Dist}(c)$.

$$\texttt{FarSafe}\,(x, y) \triangleq \sigma_{\mathrm{traj}}\,(t^*) \text{ where}$$

$$t^* = \underset{t \in [0,1]}{\mathrm{argmax}}\ (t \text{ such that, for all } t' \in [0, t], \|\sigma_{\mathrm{traj}}(t') - c\| \leq \mathrm{Dist}(c))$$

where $c = \texttt{CertifierOf}\,(x)$, In practice, this can be relaxed to return any point $\sigma_{\mathrm{traj}}(\tilde{t})$, such that $0 \leq \tilde{t} \leq t^*$, without invalidating the correctness of the algorithm (note that this relaxation has a similar effect to using a worse α). That said, using larger $\tilde{t} \leq t^*$ will yield better performance, assuming this can achieved in a similar computation time.

4.2. Modified Collision Checking Procedures

Our modified point and paths collision checking procedures are shown in Algorithms 1 and 2, respectively. For convenience, we assume the augmented data structure AG can be accessed directly, and do not list it as an input.

Point Collision Test: The point collision checking procedure $\texttt{CFreePoint}(x_{\mathrm{q}})$ appears in Algorithm 1. Given a query point $x_{\mathrm{q}} \in X$, it returns `True` if $x_{\mathrm{q}} \in X_{\mathrm{free}}$, and `False` otherwise, i.e.,

$$\texttt{CFreePoint}(x_{\mathrm{q}}) \triangleq \begin{cases} \texttt{True}, & \text{if } x_{\mathrm{q}} \in X_{\mathrm{free}} \\ \texttt{False}, & \text{if } x_{\mathrm{q}} \notin X_{\mathrm{free}} \end{cases}.$$

If possible, we use *previously computed* certificate information to *quickly* determine if x_{q} is guaranteed to be either collision-free or in-collision, lines 1.1-1.7. Let $x_{\mathrm{free}} \in S_{\mathrm{free}}$ be the configuration nearest to x_{q} that owns a collision certificate (i.e., for which $\mathrm{Dist}(\cdot)$ is defined). x_{free} is found on line 1.1. If x_{q} is closer to x_{free} than x_{free} is to an obstacle, then clearly x_{q} is collision free, lines 1.4-1.5. In this case x_{free} "certifies" x_{q} as being collision free (or, alternatively, x_{q} is within the certificate of x_{free}), and is recorded as doing so on line 1.4. Similarly, let $x_{\mathrm{obs}} \in S_{\mathrm{obs}}$ be the configuration nearest to x_{q} that owns a collision certificate. x_{obs} is found on line 1.2. If x_{q} is closer to x_{obs} than x_{obs} is to the free space, then clearly x_{q} is in collision, line 1.6-1.7. In this case x_{free} certifies x_{q} as being in-collision (i.e., x_{q} is within the certificate of x_{obs}).

If the existing certificate information is insufficient to quickly determine the collision status of x_{q} (i.e., when the two checks discussed above prove inconclusive), then an explicit check is performed using the underlying collision checking data structure (line 1.8). Note that the latter is essentially a "standard" collision check, except that it returns d_{obs} the approximate minimum distances from x_{q} to the obstacle set. If $d_{\mathrm{obs}} > 0$, then x_{q} is in X_{free} and so it is added to S_{free} (the set of vertices that have been explicitly determined to be collision-free), lines 1.9-1.10, and $\mathrm{Dist}(x_{\mathrm{q}})$ is set to d_{obs}, line 1.11, and recorded as certifying itself, line 1.12. Otherwise, x_{q} must be in collision, i.e., $x_{\mathrm{q}} \in X_{\mathrm{obs}}$; we add it to the set S_{obs} (the set of vertices that have been explicitly determined to be in-collision) and record its distance to the free set in $\mathrm{Dist}(x_{\mathrm{q}})$ (lines 1.14-1.16).

Algorithm 1: $\texttt{CFreePoint}(x_{\mathrm{q}})$

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

Algorithm 2: $\texttt{BatchCFreePath}(S_{\mathrm{near}}, x_{\mathrm{q}})$

```
1  H ← ∅;
2  c_q ← CertifierOf(x_q);
3  foreach x ∈ S_near do
4  |  if ‖x − c_q‖ ≤ Dist(c_q) then
5  |  |  H ← H ∪ {(x, x_q)};
6  |  else
7  |  |  c ← CertifierOf(x);
8  |  |  if ‖FarSafe(x_q, x) − c‖ ≤ Dist(c) then
9  |  |  |  H ← H ∪ {(x, x_q)};
10 |  |  else if CFreePath(x, x_q) then
11 |  |  |  H ← H ∪ {(x, x_q)};
12 return H
```

Path Set Collision Test: The procedure $\texttt{BatchCFreePath}(S_{\mathrm{near}}, x_{\mathrm{q}})$ appears in Algorithm 2 and returns the collision free subset of all edges from x_{q} to members of S_{near} (as well as *vice versa* if the motion graph is directed), where x_{q} is a query point and S_{near} depends on the particular planning algorithm being used, as follows:

- RRT: $S_{\mathrm{near}} = \texttt{Nearest}(V, x_{\mathrm{q}})$.
- k-PRM: $S_{\mathrm{near}} = \texttt{kNearest}(V, x_{\mathrm{q}}, k)$.
- RRT* and PRM*: S_{near} is either the expected $O(\log n)$ points within a ball of volume scaling as $\log(n)/n$ centered at x_{q}, or the $(k \log n)$-nearest neighbors to x_{q} (depending on the particular implementation being used).

Formally, $\texttt{BatchCFreePath}(S_{\mathrm{near}}, x_{\mathrm{q}})$ returns the edge set $H \subseteq \cup\{(x, x_{\mathrm{q}})\}$, where $x \in S_{\mathrm{near}}$, such that $\texttt{CollisionFreePath}(x, x_{\mathrm{q}})$ evaluates to `True`. Note that the simple version presented here assumes an undirected graph structure within the search graph. While modifying the algorithm to work with directed search-graphs is straightforward, merely requiring that (x, x_{q}) and (x_{q}, x) are handled individually, we ignore this complication in the interest of simplicity and brevity. In practice, $\texttt{CFreePoint}(x_{\mathrm{q}})$ should be called prior to $\texttt{BatchCFreePath}(S, x_{\mathrm{q}})$ in order to guarantee that x_{q} is collision-free. We assume this is the case, and thus we are guaranteed that $\|x_{\mathrm{q}} - c_{\mathrm{q}}\| \leq \mathrm{Dist}(c_{\mathrm{q}})$, where c_{q} is the point that certifies x_{q}.

Let $x_{\mathrm{near}} \in S_{\mathrm{free}}$ be the vertex in S_{free} that is nearest to x_{q}, line 2.2 (i.e., x_{near} is the particular point that certifies x_{q}). For each pair (x, x_{q}) such that $x \in S_{\mathrm{near}}$, the first step is to check whether the path segment $[x, x_{\mathrm{q}}]$ can be declared collision-free using data from AG (i.e., using certificates), lines 2.4-2.9. The details of how this procedure uses certificates to eliminate most of the (normal) collision checks depend greatly on practical details such as how certificates are defined and the shape of path-segment trajectories. In general, we seek to ensure that all points on $[x, x_{\mathrm{q}}]$ are collision free, which can be accomplished by showing that $[x, x_{\mathrm{q}}]$ is covered by certificates.

In the easiest case, and the one assumed in this section, $[x, x_{\mathrm{q}}]$ follows a geodesic of X when moving between x and x_{q} and the distance metric is such that certificates are balls. The former is a common assumption in asymptotically optimal sampling-based motion planning, while the latter happens, for instance, in Euclidean spaces. (We note that the use of balls can immediately be relaxed to include any convex shape with positive finite measure without altering the discussion that follows). Under these conditions $[x, x_{\mathrm{q}}]$ is guaranteed to be collision free whenever both x and x_{q} are closer to x_{near} than

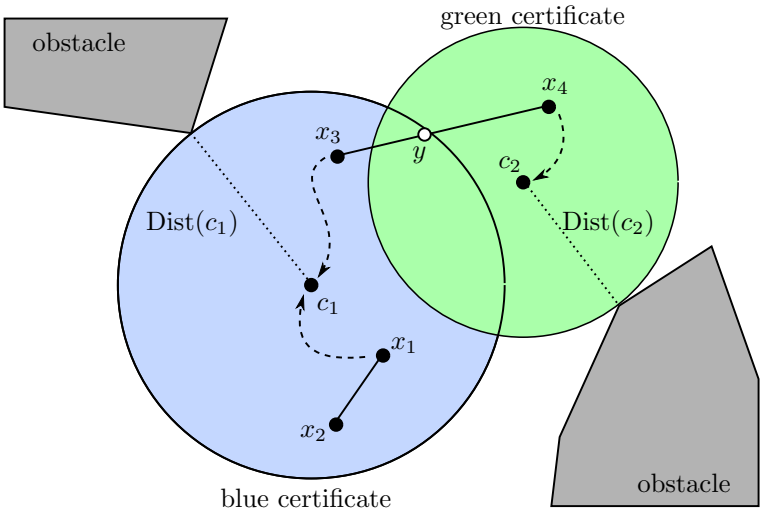

Fig. 4. Two certificates (blue and green) defined by points within distances (dotted lines) that are, respectively, less than or equal to $\mathrm{Dist}(c_1)$ from point c_1 and less than or equal to $\mathrm{Dist}(c_2)$ from point c_2. Dashed arrows represent software pointers from each point to the point that certifies it. Solid black lines represent path segments. Path segment $[x_1, x_2]$ is guaranteed to be collision-free because it exists completely within a single certificate. Path segment $[x_3, x_4]$ is guaranteed to be collision-free because it is exists within the union of both certificates. In practice the latter is calculated by determining that the furthest point along $[x_3, x_4]$ within the blue certificate, $y = \texttt{FarSafe}\,(x_3, x_4)$, is also within the green certificate.

x_{near} is to the obstacle set (see Figure 4). This is checked on lines 2.4-2.5 using the assumption that $\texttt{CFreePoint}(x_{\mathrm{q}})$ has previously been called by the motion planning algorithm, guaranteeing $\|x_{\mathrm{q}} - x_{\mathrm{near}}\| \leq \mathrm{Dist}(v_{\mathrm{near}})$.

Even if this first certificate-based test fails, it is still possible to guarantee $[x, x_{\mathrm{q}}]$ is collision free if another point c certifies x and every point along $[x, x_{\mathrm{q}}]$ is either in the certificate of x or the certificate of x_{q} (also in Figure 4). This is checked on lines 2.7-2.9, by testing if the point returned by $\texttt{FarSafe}\,(x_{\mathrm{q}}, x)$, which is guaranteed to be in the certificate of x_{q}, is also in the certificate of x. While additional tests could be concatenated in a similar way to handle cases when $[x, x_{\mathrm{q}}]$ is covered by no less than $i > 2$ certificates, such a strategy requires additional nearest neighbor queries and has quickly diminishing returns. Therefore, we recommend against doing this.

In the event that certificate-based checks are inconclusive, then a full explicit collision check is performed using a "standard" path collision checker, lines 2.10-2.11.

*4.3. Certificate Examples with RRT and PRM**

We now show how RRT and PRM*, two popular sampling-based motion planning algorithms, can be modified to use our certificate method. Appropriately modified versions of RRT and PRM* appear in Algorithms 3 and 4, respectively. Note that the pseudo-code used here is based on that by Karaman and Frazzoli (2011). AG is initialized on lines 3.1/4.1. Standard point collision checks are replaced with `CFreePoint` (i.e., Algorithm 1), lines 3.4/4.4. (The quantity $\gamma_{\mathrm{PRM}}\left(\log(|V|)/|V|\right)^{1/d}$, appearing on 4.7, is a parameter PRM* uses to determine the radius in which neighbor connections are considered, i.e., as a function of graph size). Neighbor connections are evaluated in a batch manner via a single call to `BatchCFreePath` (i.e., Algorithm 2), lines 3.6/4.8.

5. Analysis

Our collision certificate method is designed to be used with many different sampling-based motion planning algorithms. In order to make our analysis as general as possible, we only assume that the particular sampling-based motion planing

Algorithm 3: Modified RRT

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $S_{\text{free}} \leftarrow \emptyset$; $S_{\text{obs}} \leftarrow \emptyset$;
for $i = 1, \ldots, n-1$ **do**
  $x_{\text{rand}} \leftarrow \texttt{Sample}_{\texttt{i}}(\omega)$;
  if $\texttt{CFreePoint}(x_{\text{rand}})$ **then**
    $x_{\text{nearest}} \leftarrow \texttt{Nearest}(V, x_{\text{rand}})$;
    $H \leftarrow \texttt{BatchCFreePath}(\{x_{\text{nearest}}\}, x_{\text{rand}})$;
    if $H \neq \emptyset$ **then**
      $V \leftarrow V \cup x_{\text{rand}}$;
      $E \leftarrow E \cup H$;
return $G = (V, E)$;

Algorithm 4: Modified PRM*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $S_{\text{free}} \leftarrow \emptyset$; $S_{\text{obs}} \leftarrow \emptyset$;
for $i = 1, \ldots, n-1$ **do**
  $x_{\text{rand}} \leftarrow \texttt{Sample}_{\texttt{i}}(\omega)$;
  if $\texttt{CFreePoint}(x_{\text{rand}})$ **then**
    $V \leftarrow V \cup \{x_{\text{rand}}\}$;
foreach $x \in V$ **do**
  $S \leftarrow \texttt{Near}\left(V, x, \gamma_{\text{PRM}}\left(\log(|V|)/|V|\right)^{1/d}\right)$;
  $H \leftarrow \texttt{BatchCFreePath}(S \setminus \{x\}, x)$;
  $E \leftarrow E \cup H$;
return $G = (V, E)$;

algorithm being used has the same general structure that is shared by RRT, RRT*, PRM, PRM*, etc. Let SMP denote a particular sampling-based motion planning algorithm, e.g., RRT or PRM*.

The results derived in this section hold if SMP satisfies the first two following assumptions, and obstacles satisfy the third:

(A1) SMP calls the `CFreePoint` procedure $O(1)$ times per iteration.

(A2) At the n-th iteration, SMP calls `BatchCFreePath` procedure with $O(\log n)$ paths, in expectation, and such paths lie inside a ball of radius $o(|V|^{-1/(d+1)})$, almost surely, as $n \to \infty$.

(A3) Obstacle boundaries are locally Lipschitz-continuous and have finite measure.

where the n-th sample is drawn during the n-th iteration of SMP by convention. Assumption (A1) guarantees that a single point is added per iteration. Assumptions (A2) and (A3) are needed to ensure the asymptotic collision checking behavior result of Section 5.1. We note that (A2) is met by many popular sampling-based motion planning algorithms, including those that employ a shrinking neighborhood ball to ensure asymptotic optimality (PRM*, RRG, RRT*, RRT$^{\#}$, RRT$^{\text{X}}$), as well as those that iteratively connect a new point to a fixed number of nearest neighbors (RRT, k-nearest PRM). Assumption (A2) does *not* hold for the r-disc-graph-like version of PRM that used a fixed connection radius r. The validity of Assumption (A3) technically depends on both the motion planning problem being solved and the particular way that obstacles are represented in memory. That said, the danger of violating Assumption (A3) accidentally while applying our method to practical robotics problems is remote. For example, obstacles would need to be endowed with fractal like surfaces and/or infinite surface area. Both Assumptions (A1) and (A2) are needed to ensure the per-sample runtime of the underling sampling-based motion planning algorithms.

Let $I_{\text{point}}(n)$ denote the indicator random variable for the event that the `SetDistance` procedure (see Algorithm 1) is called during the n-th iteration of SMP. Similarly, let $I_{\text{path}}(n)$ be the indicator random variable for the event that the `CFreePath` procedure in Algorithm 2 is called during the n-th iteration of SMP. Define $I(n) \triangleq I_{\text{point}}(n) + I_{\text{path}}(n)$, and let $p_{\text{cc}}(n) = \mathbb{E}[I(n)]$.

5.1. Asymptotic Collision Checking Behavior

Our main result is stated in Theorem 1.

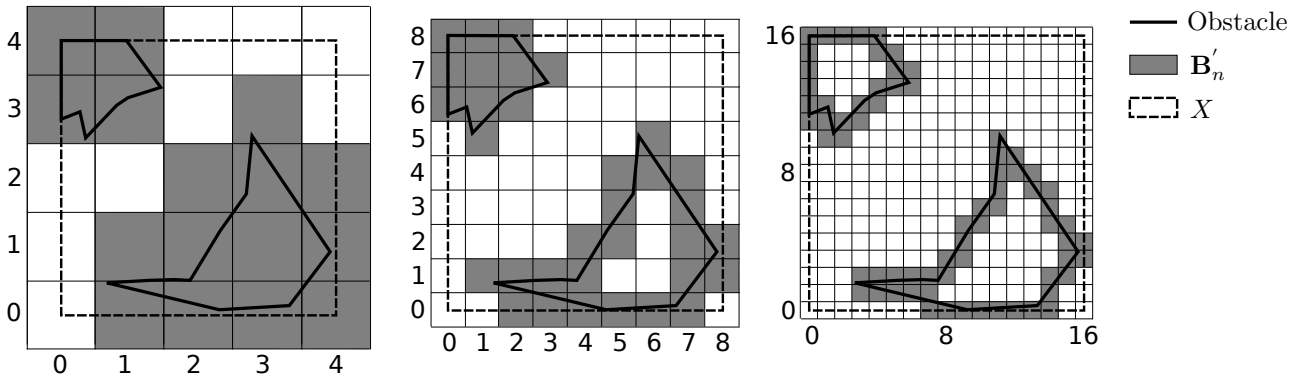

Fig. 5. The $[0,1]^D$ configuration space X (dashed) is partitioned into openly-disjoint hypercube cells with edge length $l_n := n^{-\frac{1}{D+1}}$, where n is the number of points that have been inserted into the motion planning graph (not shown). Left to right depict the partitioning for increasing values of n, respectively. Shaded cells contain part of the obstacle boundary and are therefore in $\mathbf{B}_n'$. Numbers indicate cell indices.

Theorem 1. *Assuming (A1), (A2), and (A3) hold and SMP is implemented using the proposed collision-checking certificate algorithm, then SMP has zero asymptotic expected incremental set-distance complexity, i.e.,*

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

The rest of this section is devoted to proving Theorem 1. The key idea is to use a convenient partitioning to divide the state space into several cells (see Figures 5 and 6). This allows us to compute a bound on the expected number of samples that require the set-distance procedure to be called in Algorithm 1. Finally, we show that this number grows more slowly than n, which implies Theorem 1.

Consider a partitioning of the configuration space $[0,1]^D$ into openly-disjoint hypercube cells with edge length $l_n := n^{-\frac{1}{D+1}}$. Note that the number of cells used in the partition increases vs. the number of samples n, while the size of a particular cell shrinks. Given $\mathbf{z} \in \mathbb{Z}^D$, the cell with index $\mathbf{z}$ is defined as the L^∞ ball of radius $l_n/2$ centered at $l_n\,\mathbf{z}$, i.e., $C_n(\mathbf{z}) := \left\{x' \in X : \|x' - l_n\mathbf{z}\|_\infty \leq l_n/2\right\}$, where $l_n\,\mathbf{z}$ is the scalar-vector multiplication of $\mathbf{z}$ by l_n, and $\|\cdot\|_\infty$ is the infinity norm. Let $\mathbf{Z}_n \subset \mathbb{Z}^D$ be the smallest set for which $\mathbf{C}_n := \{C_n(\mathbf{z}) : \mathbf{z} \in \mathbf{Z}_n\}$ covers the configuration space, i.e., $X \subseteq \bigcup_{\mathbf{z}\in\mathbf{Z}_n} C_n(\mathbf{z})$. Note that $\mathbf{Z}_n$ is a finite set because X is compact.

Let γ_u denote the maximum distance between two points in the unit hypercube using the distance metric $\|\cdot\|$—recall that $\|\cdot\|$ is employed in the `SetDistance` procedure. For example, $\gamma_u = \sqrt{D}$ if $\|\cdot\|$ is the L_2 norm and $\gamma_u = 1$ if $\|\cdot\|$ is the L_∞ norm.

We group the cells in $\mathbf{C}_n$ into two disjoint subsets: $\mathbf{B}_n$ contains all *boundary cells*, while $\mathbf{Q}_n$ contains all *interior cells*, see Figure 6. 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., $C_n(\mathbf{z}) \cap \partial X_{\mathrm{obs}} \neq \emptyset$. By construction $\mathbf{B}_n$ contains exactly those cells that are within $2\left(\lceil(1/\alpha)\,\gamma_u\rceil + 1\right)$ cell distance to some cell in $\mathbf{B}_n'$. Formally,

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

where α is a constant defined in the `SetDistance` procedure (in particular, α is a constant factor that defines the tightness of the obstacle distance lower bound). Finally, $\mathbf{Q}_n$ is defined as the set of all cells that are not boundary cells, i.e., $\mathbf{Q}_n := \mathbf{C}_n \setminus \mathbf{B}_n$. The choice of the number $2(\lceil(1/\alpha)\gamma_u\rceil + 1)$ will become clear shortly.

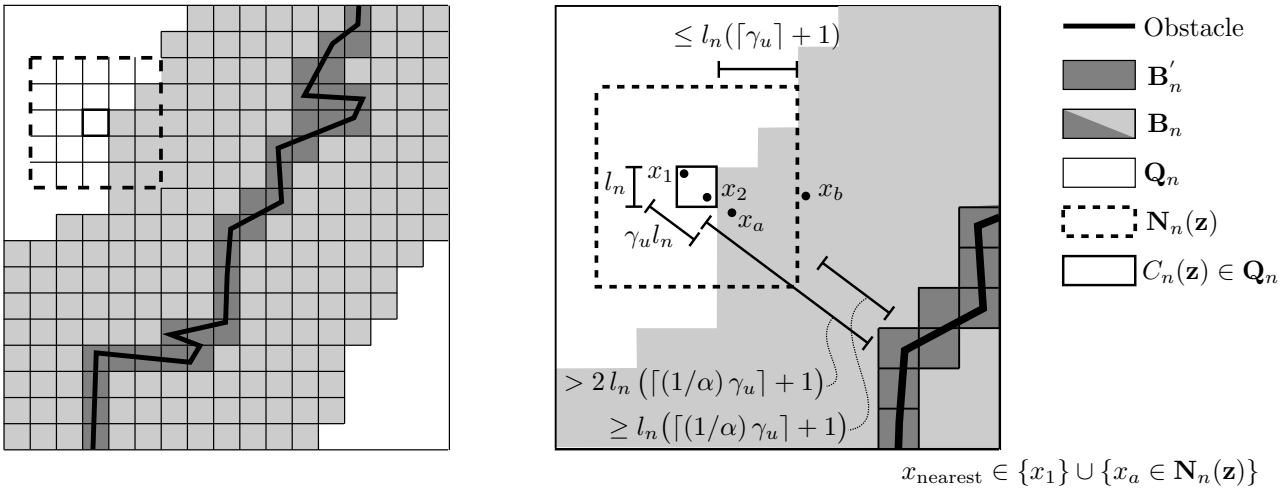

Fig. 6. Two close-up views of the cell partitioning of X used in Lemmas 1-4. For the purposes of clarity, only the boundaries of select cells are depicted. l_n is the side-length of a cell and $\gamma_u l_n$ is the maximum distance between any two points in the same cell, where γ_u depends on the distance norm being used ($\|\cdot\|_\infty$ in this figure). α is used to handle cases when the `SetDistance` procedure returns α-factor bound of the actual distance to the obstacle set boundary. Assuming that $x_1, x_2 \in C_n(\mathbf{z})$ such that $C_n(\mathbf{z}) \in \mathbf{Q}_n$, then x_{nearest} the nearest neighbor of x_2 is either x_1 or some other point $x_a \in \mathbf{N}_n(\mathbf{z})$, and so x_{nearest} is guaranteed to be at least $l_n\left(\lceil(1/\alpha)\,\gamma_u\rceil+1\right)$ from the obstacle boundary. See Lemmas 1-4 for more details.

Recall that $\mathscr{L}(\cdot)$ is used to denote the Lebesgue measure in $\mathbb{R}^D$. The total number of cells K_n can be bounded for all n as follows [3]:

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

Note that K_n is an increasing and sub-linear function of n.

Recall that $|\mathbf{B}_n| = \text{card}\,(\mathbf{B}_n)$ denotes the number of boundary cells.

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

Proof. This is true by construction and Assumption (A3). In particular, if K_n is the number of equal-size cells that cover the configuration space, then the obstacle boundary is covered by $c_1 K_n^{(D-1)/D} = c_1 K_n^{1-1/D}$ cells. □

The validity of Lemma 1 stems from the fact that, given Assumption (A3), obstacle *boundaries* are essentially $(D-1)$-dimensional structures embedded in a D-dimensional state space. The following two paragraphs (after this one), respectively describe two self-contained examples designed highlight the geometric aspects of Lemma 1; these are depicted in Figures 7 and 8, respectively. These examples are included to provide intuition; they are not part of the proof of Lemma 1.

Our first intuition-building example considers the case of polytopal obstacles. Given Assumption (A3), it is possible to decompose the obstacle space X_{obs} into $c_{\text{A}} < \infty$ non-overlapping convex polytopal obstacles $O_1, \ldots, O_{c_{\text{A}}}$ — at the cost of *increasing* the total number of boundary cells (i.e., new boundary cells appear wherever a concave obstacle has been split into two or more convex obstacles). c_{A} can be interpreted as the minimum number of convex obstacles needed to cover the obstacle space (note that such a minimum decomposition is used here only as an analytical tool). Each of the resulting obstacles O_i, being both convex and no larger than X, requires a number of cells to cover its boundary that is no more than the number of cells required to cover the boundary of X. In particular, each $(D-1)$-dimensional face of (hypercube) X requires $(K_n)^{(D-1)/D}$ cells to cover it, due to the fact that each 1-dimensional edge of X is $(K_n)^{1/D}$ cells long. There are c_{B} faces of a D-dimensional hypercube, so the total number of cells required to cover all obstacle boundaries

[3] Strictly speaking, $K_n \leq \lceil \mathscr{L}(X)\, n^{D/(D+1)} \rceil$, where $\lceil a \rceil$ returns the smallest integer greater than a (i.e., ceiling function). For brevity we omit these technical details in the sequel.

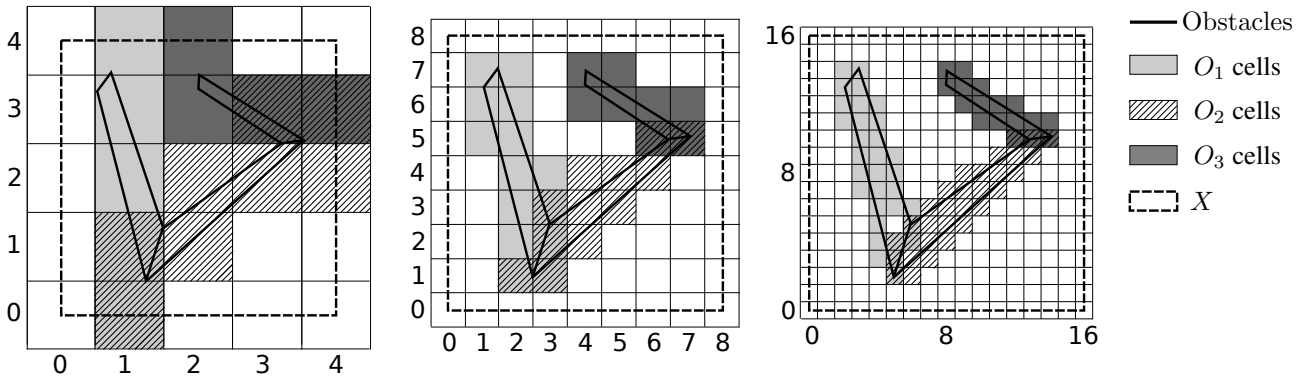

Fig. 7. The $[0,1]^D$ configuration space X (dashed) is partitioned into openly-disjoint hypercube cells with edge length $l_n := n^{-\frac{1}{D+1}}$, where n is the number of points that have been inserted into the motion planning graph (not shown). Left to right depict the partitioning for increasing values of n, respectively. A single non-convex polytopal obstacle is divided into three convex polytopal obstacles O_1, O_2, O_3 (dark black lines). Shaded cells contain part of an obstacle boundary (light gray, wavy, dark gray for O_1, O_2, O_3, respectively). Axis numbers indicate cell indices.

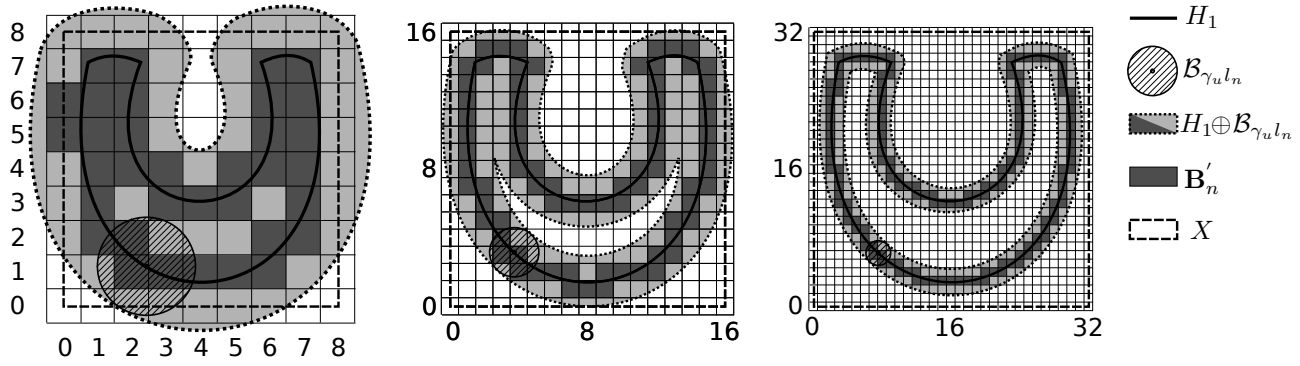

Fig. 8. The $[0,1]^D$ configuration space X (dashed) is partitioned into openly-disjoint hypercube cells with edge length $l_n := n^{-\frac{1}{D+1}}$, where n is the number of points that have been inserted into the motion planning graph (not shown). Left to right depict the partitioning for increasing values of n, respectively. A single non-convex non-polytopal obstacle is shown. The set $\mathbf{B}_n^{'}$ of grids that contain part of the obstacle boundary are dark gray. The former are always inside $H_1 \oplus \mathcal{B}_{\gamma_u l_n}$ (light gray, dotted boundary), which is the Minkowski sum of the obstacle boundary H_1 and $\mathcal{B}_{\gamma_u l_n}$, a ball of radius $\gamma_u l_n$ (striped). Axis numbers indicate cell indices.

is bounded by $|\mathbf{B}_n^{'}| \leq c_{\mathrm{A}} c_{\mathrm{B}} (K_n)^{(D-1)/D}$. Any cell in $\mathbf{B}_n$ is within some constant cell distance $c_{\mathrm{C}} < \infty$ of some cell in $\mathbf{B}_n^{'}$, where c_{C} depends on D, $\|\cdot\|$, and α. Therefore, there exists some constant $c_{\mathrm{D}} < \infty$ such that $|\mathbf{B}_n| \leq c_{\mathrm{D}} |\mathbf{B}_n^{'}|$, and so $|\mathbf{B}_n| \leq c_{\mathrm{A}} c_{\mathrm{B}} c_{\mathrm{D}} (K_n)^{(D-1)/D}$.

Our second example follows. We consider a set $\{O_1, \ldots, O_{c_{\mathrm{A}}}\}$ of obstacles, where each obstacle O_i is defined by a finite continuous surface H_i. As in the previous example, X is defined by a D-dimensional hypercube (Figure 8 depicts such a case for $c_{\mathrm{A}} = 1$ and $D = 2$). Let $\mathcal{B}_{\gamma_u l_n}$ be a D-ball of radius $\gamma_u l_n$. By construction, the Minkowski sum $H_i \oplus \mathcal{B}_{\gamma_u l_n}$ will cover more volume than the set constructed from all grids containing a piece of H_i (i.e., the subset of $\mathbf{B}_n^{'}$ relevant to O_i). By extension, the sum of all such volumes over all O_i is at least that contained in the union of all grids in $\mathbf{B}_n^{'}$; formally, $\mathscr{L}(\mathbf{B}_n^{'}) \leq \sum_{i=1}^{c_{\mathrm{A}}} \mathscr{L}(H_i \oplus \mathcal{B}_{\gamma_u l_n})$, where we abuse our notation by letting $\mathscr{L}(\mathbf{B}_n^{'})$ denote the cumulative volume covered by all grids in $\mathbf{B}_n^{'}$. It follows from geometry and Assumption (A3) that $\lim_{n \to \infty} \mathscr{L}(H_i \oplus \mathcal{B}_{\gamma_u l_n}) = 2\gamma_u l_n \mathscr{L}(H_i)$; in other words, as the thickness of $H_i \oplus \mathcal{B}_{\gamma_u l_n}$ decreases as a function of n, the volume $\mathscr{L}(H_i \oplus \mathcal{B}_{\gamma_u l_n})$ becomes increasingly well approximated by the surface area of H_i multiplied by the diameter of $\mathcal{B}_{\gamma_u l_n}$. This, along with the fact that $\mathscr{L}(H_i \oplus \mathcal{B}_{\gamma_u l_n})$ is finite for all $n \in \mathbb{N}$, guarantees the existence of a constant c_{E} such that $\sum_{i=1}^{c_{\mathrm{A}}} \mathscr{L}(H_i \oplus \mathcal{B}_{\gamma_u l_n}) \leq c_{\mathrm{E}} \sum_{i=1}^{c_{\mathrm{A}}} 2\gamma_u l_n \mathscr{L}(H_i)$ for all $n \in \mathbb{N}$. Let O_{big} denote the obstacle with the largest surface area, $O_{\mathrm{big}} = \underset{O_i \in \{O_1, \ldots, O_{c_{\mathrm{A}}}\}}{\operatorname{argmax}} \mathscr{L}(H_i)$. Using substitution

we get $\mathscr{L}(\mathbf{B}_n^{'}) \leq c_{\mathrm{A}} c_{\mathrm{E}} 2\gamma_u l_n \mathscr{L}(H_{\mathrm{big}})$. Next, we observe that dividing $\mathscr{L}(\mathbf{B}_n^{'})$ by the volume of a grid gives, to within some constant factor, $|\mathbf{B}_n^{'}|$; thus, we can choose a finite constant c_{F} such that $|\mathbf{B}_n^{'}| \leq c_{\mathrm{F}} \mathscr{L}(\mathbf{B}_n^{'})/l_n^D$ for all $n \in \mathbb{N}$. Let H_X denote the boundary of X. We also observe that the surface area (i.e., $(D-1)$-dimensional hypervolume) of both H_X and H_{big} is constant vs. n; thus, $\mathscr{L}(H_{\mathrm{big}}) = c_{\mathrm{G}} \mathscr{L}(H_X)$ for some finite constant c_{G}. Finally, we observe that because X is a hypercube, it is possible to calculate $\mathscr{L}(H_X)$ by summing over the area contained in each of its D-dimensional facets. This can be bounded in terms of K_n, the total number of grids, as follows: $\mathscr{L}(H_X) \leq c_{\mathrm{B}} c_{\mathrm{H}} (K_n)^{(D-1)/D} (l_n)^{(D-1)}$, where $(K_n)^{(D-1)/D}$ is the number of grids per facet of X, and $(l_n)^{(D-1)}$ is the surface area of a single facet on a single grid, the (finite) constant c_{B} is the number of facets in a D-dimensional hypercube, and the finite constant c_{H} is chosen to be large enough to account for the non-constant (yet deterministic) growth of K_n vs. n. Combining these observations, along with the result (from the previous paragraph) that $|\mathbf{B}_n| \leq c_{\mathrm{D}} |\mathbf{B}_n^{'}|$ for some constant c_{D}, via substitution into $\mathscr{L}(\mathbf{B}_n^{'}) \leq c_{\mathrm{A}} c_{\mathrm{E}} \mathscr{L}(H_{\mathrm{big}}) 2\gamma_u l_n$ and then simplifying gives: $|\mathbf{B}_n| \leq c_{\mathrm{A}} c_{\mathrm{B}} c_{\mathrm{D}} c_{\mathrm{E}} c_{\mathrm{F}} c_{\mathrm{G}} c_{\mathrm{H}} 2\gamma_u (K_n)^{(D-1)/D}$, where γ_u is a finite constant given D. This concludes our second self-contained example; we now resume building toward the proof of Theorem 1.

Lemma 1 implies that the fraction of all cells that are boundary cells is bounded by:

$$\frac{c_1 \, |\mathbf{B}_n|}{K_n} \leq \frac{c_1 \, (K_n)^{1-1/D}}{K_n} = c_1 \, (K_n)^{-1/D} \leq c_1 \, (\mathscr{L}(X))^{-1/D} \, n^{-\frac{1}{D+1}} = c_2 \, n^{-\frac{1}{D+1}},$$

where c_2 is a constant.

We now bound the number of calls to the collision-distance procedure. This is accomplished by separately considering the mutually exclusive sets of samples that fall into interior cells and boundary cells, respectively. Recall that S_{free} and S_{obs} denote the sets of vertices that are explicitly checked and found to be in X_{free} and X_{obs}, respectively. Let $S_{\mathrm{checked}} := S_{\mathrm{free}} \cup S_{\mathrm{obs}}$. The following lemma will later be used to derive a bound on the number of points in S_{checked} that fall into interior cells.

Lemma 2. *Assuming (A1) and (A2) hold, SMP is implemented using the proposed collision-checking certificate algorithm, and $C_n(\mathbf{z}) \in \mathbf{Q}_n$ is some interior cell; then there exists at most one vertex from S_{checked} in $C_n(\mathbf{z})$, i.e.,*

$$|C_n(\mathbf{z}) \cap S_{\mathrm{checked}}| \leq 1 \textit{ for all } C_n(\mathbf{z}) \in \mathbf{Q}_n.$$

Before proving Lemma 2, we must first introduce some additional notation and establish two intermediate results (Lemmas 3-4). 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 $C_n(\mathbf{z})$; formally, $\mathbf{N}_n(\mathbf{z}) := \{C_n(\mathbf{z}') : \|\mathbf{z}' - \mathbf{z}\|_\infty \leq \lceil \gamma_u \rceil + 1\}$. $\mathbf{N}_n(\mathbf{z})$ includes the cell $C_n(\mathbf{z})$ as well as all cells that are within distance $\lceil \gamma_u \rceil + 1$ of $C_n(\mathbf{z})$. Note that $\mathbf{N}_n(\mathbf{z})$ is defined such that the space occupied by its members excludes points sufficiently far from $C_n(\mathbf{z})$; and thus, $\mathbf{N}_n(\mathbf{z})$ includes points sufficiently distant from the obstacle set boundary whenever $C_n(\mathbf{z}) \in \mathbf{Q}_n$. These properties are formalized in the following two lemmas (see Figure 6 for an example illustrating the quantities used).

Lemma 3. *Any point in a cell outside of $\mathbf{N}_n(\mathbf{z})$ has distance at least $\gamma_u \, l_n$ to any point 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. This is true by the construction of $\mathbf{N}_n(\mathbf{z})$. □

Lemma 4. *For all $\mathbf{z}$ such that $C_n(\mathbf{z}) \in \mathbf{Q}_n$, any point in a member of $\mathbf{N}_n(\mathbf{z})$ is at least $\big(\lceil (1/\alpha) \, \gamma_u \rceil + 1\big) l_n$ distant to any point on the obstacle set boundary, i.e., $\|x - x'\| \geq \big(\lceil (1/\alpha) \, \gamma_u \rceil + 1\big) l_n$ for all $x \in \partial X_{\mathrm{obs}}$ and $x' \in C_n(\mathbf{z}')$ such that $C_n(\mathbf{z}') \in \mathbf{N}_n(\mathbf{z})$.*

Proof. By Construction, cell $C_n(\mathbf{z})$ is at least distance $2(\lceil (1/\alpha) \, \gamma_u \rceil + 1)$ from any cell that intersects the boundary of the obstacle set. Additionally, any cell in $\mathbf{N}_n(\mathbf{z})$ is at most distance $\lceil \gamma_u \rceil + 1 \leq \lceil (1/\alpha) \, \gamma_u \rceil + 1$ from $C_n(\mathbf{z})$. Thus, by the

triangle inequality, any cell in $\mathbf{N}_n(\mathbf{z})$ is at least distance $\lceil (1/\alpha)\,\gamma_u \rceil + 1$ away from any cell that intersects the obstacle boundary. □

We can now prove Lemma 2.

Proof of Lemma 2. (Proof by contradiction). Suppose $x_1, x_2 \in S_{\text{checked}}$ are two points that fall into $C_n(\mathbf{z})$. Without loss of generality, suppose that x_1 is added to S_{checked} before x_2. Let $x_{\text{nearest}} \in S_{\text{checked}}$ denote the nearest point to x_2 that is in S_{checked} (i.e., when Algorithm 1 is called with x_2).

First, we claim that x_{nearest} must lie 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 from x_2 than x_1. Note that the distance between x_1 and x_2 is at most $\gamma_u\, l_n$. By Lemma 3, any point that lies in a cell outside of $\mathbf{N}_n(\mathbf{z})$ must be at least distance $\gamma_u\, l_n$ from x_2. Thus, x_{nearest} must lie in some cell in $\mathbf{N}_n(\mathbf{z})$. However, Lemma 4 guarantees that $\texttt{Dist}(x_{\text{nearest}}) \geq (1/\alpha)\,\|x_2 - x_{\text{nearest}}\|$; and thus x_2 cannot be in S_{checked}, even when the `SetDistance` procedure returns the α-factor bound of the actual distance to the obstacle set boundary. i.e., x_2 must have been certified (either as collision-free or in-collision) by x_{nearest} and therefore x_2 could not have been added to $S_{\text{checked}} = S_{\text{free}} \cup S_{\text{obs}}$. This provides the necessary contradiction. □

The following lemma (Lemma 5) provides an upper bound on the *expected* number of samples that fall into boundary cells. This result is then used in the subsequent lemma (Lemma 6) to calculate an upper bound on the expected number of points in S_{checked} that fall into boundary cells. Let X_n denote a set of n samples drawn independently and uniformly from X, and let S_n denote the number of samples that fall into boundary cells, i.e.,

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

Lemma 5. $\mathbb{E}[S_n] \leq c_3\, n^{\frac{D}{D+1}}$ *for some constant* c_3 *that is independent of* n

Proof. Let E_i denote the event that the i-th sample falls into a boundary cell, and let Y_i denote the indicator random variable corresponding to the event E_i. Lemma 1 guarantees that $c_2 n^{-\frac{1}{D+1}}$ is the fraction of cells that are boundary cells. 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, and c_3 is independent of n. □

The following provides a bound on the expected number of points in S_{checked}.

Lemma 6. $\mathbb{E}\left[\text{card}\,(S_{\text{checked}})\right] \leq c_4\, n^{\frac{D}{D+1}}$ *for some constant* c_4 *independent of* n.

Proof. By Lemma 2, the number of points in S_{checked} that fall into an interior cell is at most the total number of interior cells. Thus the number of points in S_{checked} that fall into an interior cell is less than the total number of cells K_n, for $K_n \leq \mathscr{L}(X)\, n^{\frac{D}{D+1}}$ (see Equation (1)). Clearly, the expected number of points in S_{checked} that fall into a particular boundary cell is no more than the expected number of samples that fall into all boundary cells. By Lemma 5 the latter number is bounded by $c_3\, n^{\frac{D}{D+1}}$, where c_3 is a constant independent of n. Thus, $\mathbb{E}\left[\text{card}\,(S_{\text{checked}})\right] \leq c_4\, n^{\frac{D}{D+1}}$ for some constant c_4. □

We are finally ready to prove Theorem 1.

Proof of Theorem 1. We begin by using a standard coupling argument to show that p_{cc} is a non-increasing function of n. Consider the run of the algorithm with $n+1$ samples. The events $\{I(n) = 1\}$ and $\{I(n+1) = 1\}$ are coupled as follows. Let A_n and A_{n+1} denote the events that the nth and the $(n+1)$th samples are explicitly checked, respectively. Clearly, $\mathbb{P}(A_{n+1}) \leq \mathbb{P}(A_n)$ in this coupled process, since the first $(n-1)$th samples are not affected by the act of drawing the n-th and $(n+1)$-th samples. Furthermore, $\mathbb{P}(A_n) = \mathbb{P}(\{I(n) = 1\}) = p_{\text{cc}}(n)$ 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 $n \in \mathbb{N}$. This implies that $\lim_{n\to\infty} p_{\text{cc}}(n)$ exists.

We now prove $\lim_{n\to\infty}(1/n)\ \sum_{k=1}^{n} p_{\mathrm{cc}}(k) = 0$. Clearly, $\sum_{k=1}^{n} I_{\mathrm{point}}(k) = |S_{\mathrm{checked}}|$. Hence,

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

where the inequality follows from Lemma 6. Similarly,

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

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,

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

Together, $p_{\mathrm{cc}}(n+1) \leq p_{\mathrm{cc}}(n)$ for all $n \in \mathbb{N}$ and $\lim_{n\to\infty}(1/n)\ \sum_{k=1}^{n} p_{\mathrm{cc}}(k) = 0$ imply that $\lim_{n\to\infty} p_{\mathrm{cc}}(n) = 0$. □

5.2. Effects on Runtime

Recall that the complexities of drawing a sample and performing a collision check are bounded by c_{sample} and $c_{\mathrm{cc}}(X_{\mathrm{obs}})$, respectively, where the former is a constant and the latter is a function that depends on the obstacle description. The expected complexity of checking collisions with n_{obs} convex obstacles is $O\left(\log(n_{\mathrm{obs}})^D\right)$ Samet (2006). While a variety of methods achieve this bound[4], e.g., see Samet (2006), our discussion in this section does not assume any particular method is used; we assume only that whatever method is used runs in $O\left(\log(n_{\mathrm{obs}})^D\right)$.

The complexity of (approximate) proximity searches[5] that return the $O(\log n)$ nearest elements to a query point from a set of cardinality n is $O(\log n)$ (Samet, 2006). The latter applies to both k-nearest neighbor searches (both for a fixed k fixed and for k scaling as $\log n$), and range searches among uniform random samples for those points within a ball of volume scaling as $\log(n)/n$. Again, the discussion in this section assumes only that the method used runs in time $O(\log n)$, and does not assume a particular method is used.

The complexity of generating a trajectory σ between two points (i.e., with a procedure often called the "local steering function") is bounded by an environmentally independent constant c_{plan}. For convenience we assume that c_{plan} also includes the time required by all other bookkeeping operations that run in time $O(1)$ per sample point. Local trajectories (generated by the local steering function) must also be collision checked. The time required for a single trajectory collision check is bounded by a constant $c_{\mathrm{path}}(X_{\mathrm{obs}})$ that depends on the particular obstacle description that is used.

Using our certificate method causes standard collision checks to be replaced with approximate minimum-distance computations. In particular, we require a lower bound $\bar{d}$ on the minimum distance d^* that satisfies $\alpha d^* \leq \bar{d} \leq d^*$, for some $\alpha \in (0, 1]$ (i.e., a non-trivial lower bound on distance from a point in free-space to an obstacle, or the equivalent for a point in collision to free-space). The computational cost of such a procedure depends on the description of the environment and

[4]For example, range-trees achieve this bound for points as well as for 'orthogonal objects' (axis aligned boxes and lower dimensional primitives, like an axis aligned line segment), and can be extended to more general objects assuming the latter are 'well behaved' in the sense of being finitely-featured (no fractal representations) and convex.

[5]For example, a hierarchy of Delaunay triangulations that is relaxed to perform an *approximate* proximity search (e.g., using projection and/or some other well known approximation technique) requires $O(\log n)$ to find the Delaunay simplex containing the query, and then floodfill can be used to find the $O(\log n)$ outputs. An 'in general position' (IGP) assumption is often used in the analysis of such methods. That is, no $nD + 2$ points can lie on the same hypersphere. We note that the probability of the latter event is 0 assuming that random sampling is used; however, if points are not IGP, then a random perturbation of the points by some small ϵ is sufficient to recover this runtime.

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

Table 1: Asymptotic bounds on the expected incremental complexity of standard sampling-based motion planning algorithms (with and without using certificates). Note that $\limsup_{n\to\infty} p_{\mathrm{cc}}(n) = 0$ (see Theorem 1). c_{sample}, c_{plan}, $c_{\mathrm{cc}}(X_{\mathrm{obs}})$, and $c_{\mathrm{path}}(X_{\mathrm{obs}})$ are upper bounds on the complexity of drawing a sample, calculating a trajectory, collision checking a point, and collision checking a trajectory, respectively.

will be indicated with $c_{\mathrm{dist}}(X_{\mathrm{obs}})$. In many cases $c_{\mathrm{dist}}(X_{\mathrm{obs}})$ is relatively small vs. the time required for standard collision checking.

In general, the expected iterative complexity of sampling-based motion planning algorithms is a function of the size of the potential neighbor set of a candidate node, as well as which points and trajectories are collision checked. The following list outlines the major steps of four common motion planning algorithms:

- RRT: Sample a point, find the nearest neighbor, saturate (generate a new point between the sample and its nearest neighbor), check the new point for collision, and connect the new point to the nearest neighbor (trajectory collision check).
- k-PRM: Sample a point, collision check the new point, find the new point's k-nearest neighbors, connect to reachable subset of the k-nearest neighbors (k trajectory collision checks).
- RRT*, PRM*: Sample a point, saturate, collision check the sample, find the neighbors (all nodes within a ball of volume scaling as $\log(n)/n$—or, alternatively, the $(k \log n)$-nearest neighbors, for a fixed k), connect to reachable subset of the aforementioned nodes (one trajectory collision check per member of the neighbor set).

Table 1 summarizes the resulting expected incremental complexities of these four algorithms—both with and without the modifications necessary to use collision certificates. Recall that local planning and trajectory collision checking occur only if a new sample is not in collision. Note that $\limsup_{n\to\infty} p_{\mathrm{cc}}(n) = 0$ (see Theorem 1) as the number of samples increases. Thus the computational cost of collision detection is shifted more-and-more from standard collision checking to a nearest neighbor search as n increases. Indeed, this is a nearest-neighbor search that is already used for the graph-building aspects by most sampling-based motion planning algorithms.

Our main result from Section 5.1 shows that certificates will approach a covering of the space X as $n \to \infty$, *even* if we can only calculate a non-trivial lower bound on point-obstacle distance, as paramiterized by α. Obviously, tighter lower-bounds will cause faster convergence (vs. n) toward the full covering due to the fact that each point will prevent future (normal) collision checks in a greater subset of X. Assuming all other factors remain unchanged, using a tighter α will decrease expected per-sample runtime. That said, the tightness of the bound is likely to be dependent on the obstacle representation used in practice and the motion planning problem being solved — and therefore fixed for a particular application.

If a system is designed such that it is possible to either set α or tune α 'on-the-fly' (note that such an ability would only make sense if calculating a closer bound requires more work than calculating a loose bound), then choosing the correct 'α' must be done with some care. While in-depth discussion of this trade-off is beyond the scope of our current paper, we note that the bound should only be tightened as long as the marginal decrease in future collision checking time (due to more nodes falling within an old certificate) is greater than the marginal increase in computation time (required for a single point-obstacle distance query).

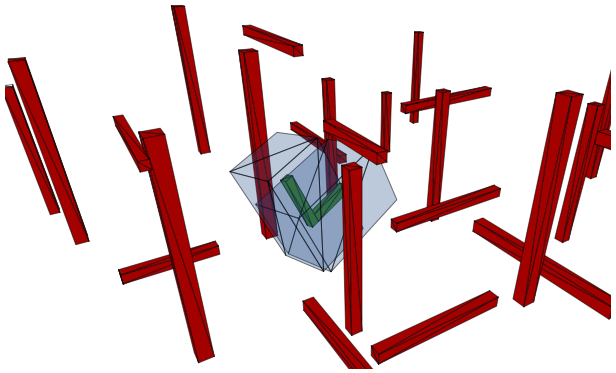

Fig. 9. Example of a workspace certificates for an L-shaped volume found from conservative dilation of the volume faces by the collision distance. The example volume is in green, the obstacles in red, and the certificate volumes in translucent blue.

6. Extension 1: Workspace Certificates

The basic collision certificate that has been presented so far has assumed that certificates are located in the configuration space. In this section we show one way that certificates can alternatively be used in the workspace to solve problems involving obstacles and robots that are modeled as sets of polytopes.

The use of certificates within a particular workspace is intimately related to both the robotic system itself and the obstacle representation that is used to model the environment. Any non-trivial workspace representation involves many practical details that we have been able to ignore until now. However, in this section we must ground our discussion in a particular obstacle representation, and thus must flesh-out many of these details.

The main point of this section is to provide a proof-of-concept that certificates can be used directly in the workspace for a family of non-trivial motion planning problems. We believe that the particular representation we choose to study is widely applicable and the implementation details that we discuss are well motivated; however, we make no claim that these are the optimal representation or implementation for using certificates in the workspace, in general. Moreover, the idea may not even be immediately applicable to many popular "off-the-shelf" motion planning code packages.

6.1. Workspace Representation and Additional Notation

We assume that the workspace is 3D, $\mathcal{W} \subset \mathbb{R}^3$ and that the robot and obstacles are each modeled as sets of polytopes in the workspace, and that each certificate is defined by the interior of a polytope in the workspace (see Figure 9). A polytope $P = (V, F)$ is a volumetric set of points defined by a set of vertices $V = \{v_j\}$ and a set of bounding planes $F = \{f_i\}$, where V and F are mutually dependent. (Note that these vertices should not be confused with graph nodes in the configuration space). Each plane is defined by a normal vector $\mathbf{n}$ and an offset distance d. Due to convexity, for $P \subset \mathbb{R}^n$, $n = 2, 3$, we define the ith *face* $f_i \subset P$ as the set of points x satisfying $\mathbf{n}_i \cdot x = d_i$ and $\mathbf{n}_j \cdot x \leq d_j$ for $j \neq i$. With some abuse of notation, we say face $f_i = (\mathbf{n}_i, d_i)$. For our purposes, we assume $P \subset \mathcal{W}$. We will often refer to a polytope as a "volume," e.g., a polytope used as an obstacle is a "collision volume" and a polytope used for a certificate is a "certificate volume."

Given a robot defined by a set of polytopes $\{P_0, \ldots, P_m\}$ such that $P_i \subset \mathcal{W}$ for $i \in \{0, \ldots, m\}$, the configurations space is given by $X \subset \mathbb{R}^{D_0} \times SO(n_0) \times, \ldots, \mathbb{R}^{D_m} \times SO(n_m)$, where $n_i \leq 3$. Without loss of generality, we focus our discussion on a single polytope robot $P_0 \subset X \subset \mathbb{R}^D \times SO(n)$. In practice, robots containing multiple polytopes, including articulated arms, can be handled by considering each polytope independently according to the method presented in this section. 'Robot' should be understood to mean 'polytope' in the sequel.

We assume that motion of the robot is constrained to translations and rotations. In general, workspace transformations between a robot's local coordinate system to a global frame are often handled using an *armature*, i.e., a tree of reference frames each one referred to as a *link* of the armature. Each link is assigned a unique index i, and we 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 point x represented in F_i, the same point represented in its parent frame F_j is given by:

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

We employ the Canny method (Canny, 1986) of interpolation to model motion between two points. The Canny method uses an interpolation that accounts for rotation based on a stereographic projection of quaternions; it becomes equivalent to linear interpolation as the magnitude of rotation goes to zero. Given two configurations $x_a, x_b \in X \subset \mathbb{R}^D \times SO(n)$, the Canny method provides a trajectory $\sigma \subset X$ that is convenient for moving from x_a to x_b and both collision checking and using collision certificates in the workspace $\mathcal{W}$ (as we will describe shortly). Formally, $\sigma : [0,1] \to X \subset \mathbb{R}^n \times SO(n)$, where $\sigma(0) = x_a$, and $\sigma(1) = x_b$. We use s as the interpolation parameter along σ, i.e., $s \in [0,1]$.

Let $F_{i,\sigma}(s)$ denote the transform from the local (workspace) coordinate system of link i to that of its (workspace) parent as parameterized by $s \in [0,1]$. Thus, given the chain $J = \{1, \ldots, i\}$ of links through the armature from the global coordinate frame 0 to frame i, it is possible to construct a transformation from link i to the global coordinate frame as $\prod_{j \in J} F_{j,\sigma}(s)$. The chain J commonly corresponds to the links in an articulated arm, however, the same formulation can be used to account for the motion of a vehicle that has undergone a sequence of motions, i.e., along the edges of motion-planning graph. In the latter case $\prod_{j \in J \setminus i} F_{j,\sigma}(s) = \prod_{j \in J \setminus i} F_{j,\sigma}(1)$ is constant, which conveniently allows us to store the result of $\prod_{j \in J \setminus i} F_{j,\sigma}(1)$ at each node in the motion planning graph in order to quickly compute $\prod_{j \in J} F_{j,\sigma}(s) = \left(\prod_{j \in J \setminus i} F_{j,\sigma}(1) \right) F_{i,\sigma}(s)$.

Appendix A contains derivations of $F_{j,\sigma}(s)$ Canny frame transforms that have been specifically tailored to the workspace certificate method presented in this section (additional details regarding articulated motion appear in Appendix B, and a thorough treatment also appears in Chapter 3 of Bialkowski (2013)).

In practice $F_{j,\sigma}(s)$ takes the form

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

where $R_{j,\sigma}(s)$ and $\mathbf{T}_{j,\sigma}(s)$ represent the rotational and translational components of movement along σ as parameterized by s. Given this matrix based machinery, it is notationally convenient to define $\underline{\mathbf{n}} = \begin{bmatrix} \mathbf{n} \\ 1 \end{bmatrix}$ and $\underline{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$.

6.2. Querying workspace certificates

Let P_0 be a polytope robot undergoing a smooth motion through the workspace $\mathcal{W}$ as described by the Canny interpolated path $\sigma \subset X$. Let P_1 be a convex polytope collision certificate such that at $\sigma(0) \subset P_1$ and $P_0 \subset P_1$. We wish to determine if P_0 remains within its convex polytope certificate P_1 for $\sigma(s)$ for all values of the interpolation parameter $s \in [0,1]$. In general,

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

We may restrict our focus to "Canny type A" collision predicates, where a vertex of the robot pierces a face of the certificate (see Figure 10-Right), i.e., because the robot is already inside the certificate, the certificate is a convex polytope, and faces include their vertices..

Theorem 2. *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 v of P_0 and some face $f = (\mathbf{n}, d)$ of P_1 such that*

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

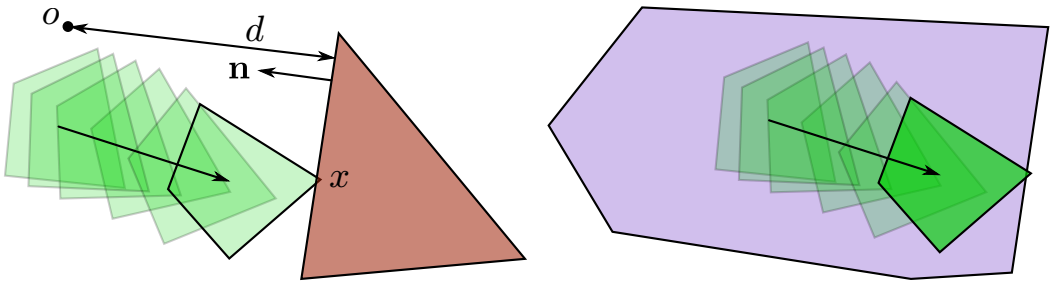

Fig. 10. Canny type A contact occurs when a vertex of the moving polytope pierces a face of the stationary polygon (Left and Right). We are concerned with the case where a convex polytopal certificate is pierced by a smaller polytopal robot (Right).

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 x be some non-vertex point $x \in P_0$ and outside of P_1, $x \notin P_1$. By definition of a polytope, x may be represented by a convex combination of the vertices of P_0. By assumption all vertices of P_0, $v_j \in P_1$, so by convexity of P_1, x must all be in P_1, which is a contradiction. □

Theorem 2 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 from 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, we compute the Sturm polynomial Hook and McAree (1990) 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 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 in continuous collision checking.

6.3. 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 Lin (1993) or extended GJK Gilbert et al. (1988) 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.

Let R_i, d_i be workspace the orientation and collision distance of the ith collision volume C_i of the system that is located at a particular configuration x_i. The Minkowski sum $C_i \oplus \mathcal{B}_{d_i}$, of the ith collision volume with a ball of radius d_i yields a convex certificate whose interior is collision free, but for which path containment is difficult to evaluate. We can select instead any polytope contained in $C_i \oplus \mathcal{B}_{d_i}$ which also contains C_i (see Figure 11). We may consider various certificate volume families for selecting such polytope certificates, trading off between complexity of the volume and spatial coverage, just as in how we select the collision volume itself. We now focus on using certificates which are dilated copies of the collision volume.

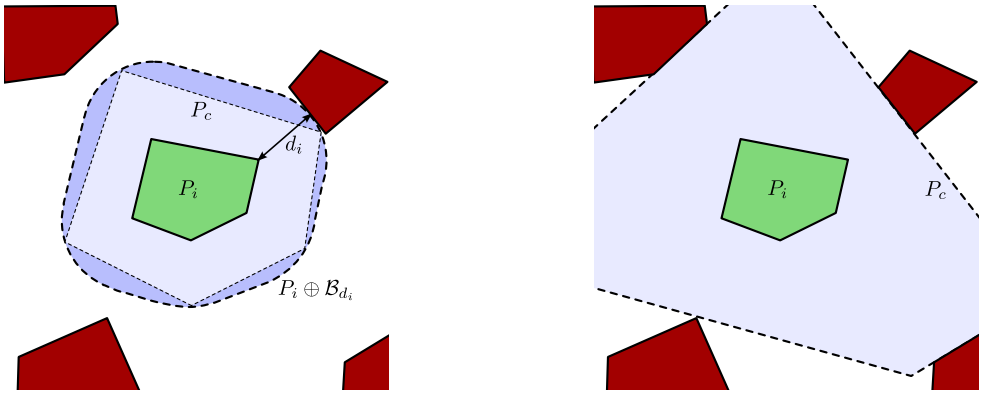

Fig. 11. Example of 2d certificates generated using dilated collision volume and collision distance (Left) and successive pruning by supporting hyperplane of nearest obstacle (Right).

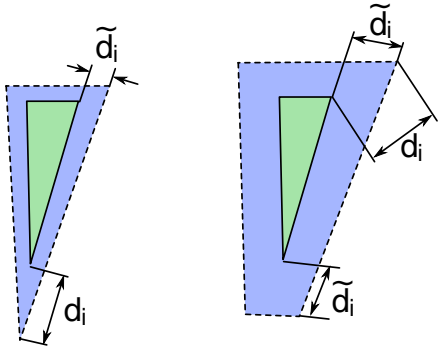

Fig. 12. The addition of a redundant face allows for larger collision volumes.

A certificate volume can be generated quite easily by dilating each of the faces of the collision volume by a distance $\tilde{d}_i \leq d_i$ 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 $\tilde{d}_i$ 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 12. Note that we may also simply assume such options where exploited in the modeling phase.

6.4. Generating certificates by priority pruning

The method of computing certificates in Section 6.3 has the advantage of requiring only collision distance output from the collision checker, the same as the basic method presented in Section 4. Certificates 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 is less likely to encounter this problem, but requires a deeper interaction with the standard collision checking system. The algorithm is summarized in Algorithm 5.

Algorithm 5 builds a polyhedral certificate P_c by first finding the nearest obstacle to the obstacle set. The algorithm returns the point $x \in X_o$ which is nearest to the collision volume (line 3). The supporting hyperplane at x is a hyperplane coincident to x and oriented so that the normal $\mathbf{n}$ points from x toward a nearest point in P to x (Gilbert et al., 1988) (line 6). This hyperplane is added as a face of P_c, and the obstacle set is pruned to remove obstacle points on the outer halfspace of the supporting hyperplane (lines 7-8). The algorithm is then repeated on the remaining obstacle set until all obstacles lie outside of P_c (line 9) or the point is found to be in collision (lines 4-5). Note that this method may actually return a

Algorithm 5: `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.

input : X_o the workspace obstacle set
input : P a workspace collision volume
initialize $P_c = \mathbb{R}^n$;
while $X_o \neq \emptyset$ **do**
  $x \leftarrow$ `Nearest` (X_o, P) ;
  if $x \in P$ **then**
    return *in-collision* ;
  $(\mathbf{n}, d) \leftarrow$ `SupportingHyperplane` (P, x) ;
  $P_c \leftarrow P_c \cap \{x \,|\, \mathbf{n} \cdot x \leq d\}$;
  $X_o \leftarrow X_o \cap \{x \,|\, \mathbf{n} \cdot x \leq d\}$;
return P_c ;

polytope with some vertices and or faces unset (i.e., located at infinity), this is why use the term 'polyhedral' to describe the certificate. However, it is important to note that the certificate is not concave.

While this technique 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 (see Figure 11).

7. Extension 2: Exploiting Symmetries

Some motion planning problems involve configuration spaces and/or workspaces with symmetries that can be exploited to make the certificate method more efficient. A canonical case, and the running example used in this section, is centralized multi-robot motion planning. We note that similar ideas can be used with the workspace collision certificate methods of Section 6.

Assuming a multi-robot team consists of R members that share an environment[6] , then the configuration space X of the centralized problem is a Cartesian product of the space of each robot, $X = X_1 \times \ldots \times X_R$. Points are defined $\mathbf{x} = x_1 \times \ldots \times x_R$, where $\mathbf{x} \in X$ and $x_i \in X_i$. Collision checking vs. obstacles for any robot i can be accomplished piece-wise in X_i (robot vs. robot collision checking can similarly be reduced to checking one robot vs. another in a separate local coordinate system). We now explore two variations of the certificate method that are designed to exploit such symmetries.

7.1. Partial Certificates

In the *partial certificate* variation of the certificate method each robot performs collision checking in its own projection of the full space, i.e., robot i collision checks in X_i.

If $\mathbf{x} \in X$ is *not* certified as either collision-free or in-collision with respect to all robots, then only a partial collision check is required vs. the projection(s) that were not individually certified. For example, let two points in the team's configuration space $X \subset \mathbb{R}^3$ be defined as $\mathbf{x} = x_1 \times x_2 \times x_3$ and $\mathbf{y} = y_1 \times y_2 \times y_3$. If y_1 and y_3 are within the certificates of some points x_1 and x_2, respectively, but y_2 is not within the certificate of x_2, then only $1/3$ check is required to determine the safety of $\mathbf{y}$, i.e., because we only need to explicitly check y_2 instead of y_1, y_2, and y_3.

[6]If all members of the team do not share an environment, then it may be possible to reduce the problem by dividing it into a set of disjoint sub-problems, one per each set of robots that are common to a particular environment, and such that each robot belongs to only one team.

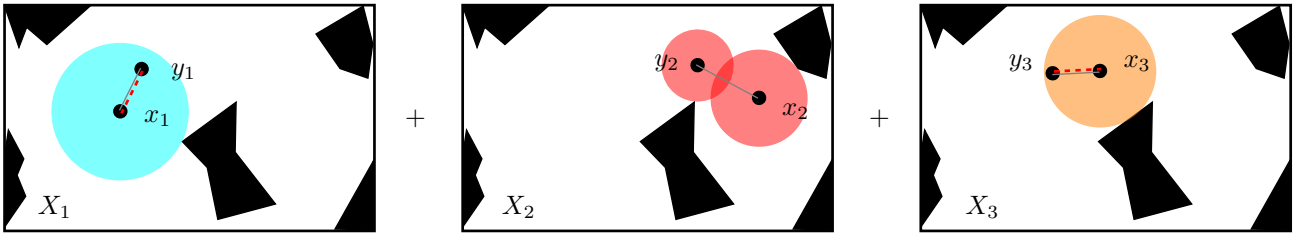

Fig. 13. ***Partial certificate***: If a point is *not* certified as safe with respect to a subspace projection, then only a partial collision check is required. $\mathbf{x} = x_1 \times x_2 \times x_3$ and $\mathbf{y} = y_1 \times y_2 \times y_3$ are two configurations in $X \subset \mathbb{R}^3$. The projections y_1 and y_3 correspond to robots 1 and 3 are within the certificates of x_1 and x_3, respectively, but y_2 is not within the certificate of x_2. Thus, only $1/3$ of a "full" team check is required for robot 2.

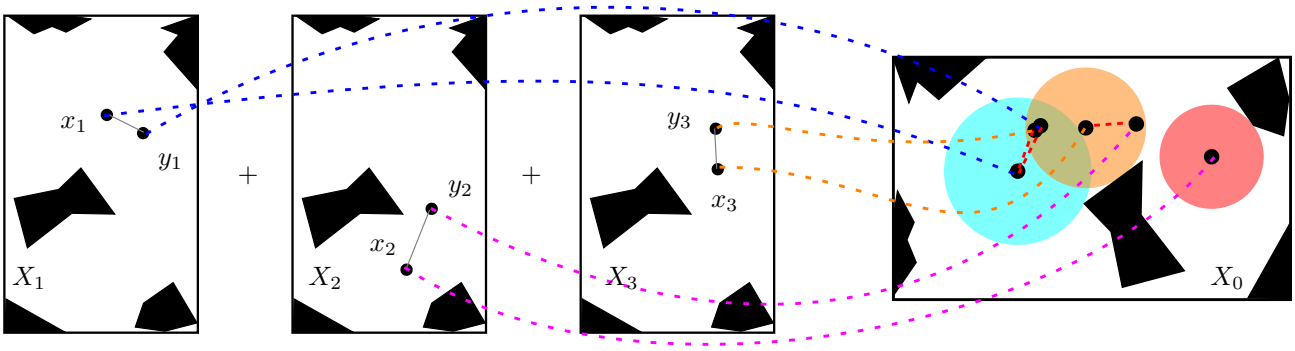

Fig. 14. ***Shared projection***: all robots collision check in the *same* D-dimensional projection (far right). Pointers from configuration space node projections to their collision-checking projection counterparts are depicted with blue/magenta/orange dotted lines, respectively. $\mathbf{x} = x_1 \times x_2 \times x_3$ and $\mathbf{y} = y_1 \times y_2 \times y_3$ are two configurations in X. x_1 certifies y_3 and x_3 certifies y_2.

In practice, the implementation of this method requires that each node in the motion-planning graph stores R certificate pointers (i.e., instead of the single pointer required by the basic method). Storing one pointer per robot enables a new node to be certified by a combination of different old nodes and/or to calculate its own partial certificates as needed.

7.2. Shared Projection

Assuming that robots are identical, then $X_i = X_0$ for all i and so $X = (X_0)^R$. In this case it is possible for all robots to perform collision checking in the same copy of X_0 (see Figure 14). We call this idea *Shared Projection*.

This method causes X_0 to become populated with certificates R times more quickly due to the fact that R nodes are checked and possibly added during each iteration instead of 1. Thus, it requires even fewer (standard) collision checks than *partial certificate*. On the other hand, it requires an *additional* point proximity data structure (e.g., a kd-tree) to store certificate points in X_0.

In practice, the runtime of this variation can be improved by seeding collision certificate nearest-neighbor queries in X_0 based on the nearest-neighbor as determined by normal (graph algorithm) nearest-neighbor queries in X. For example, when searching a kd-tree in X_0 for the collision status of y_1 we begin the search at the location of x_1 instead of at the root of the tree, where $\mathbf{x}$ is the point that has (already) been returned by the motion-planning kd-tree search. Such a seeding is also likely to be possible with other tree-based point-proximity data structures.

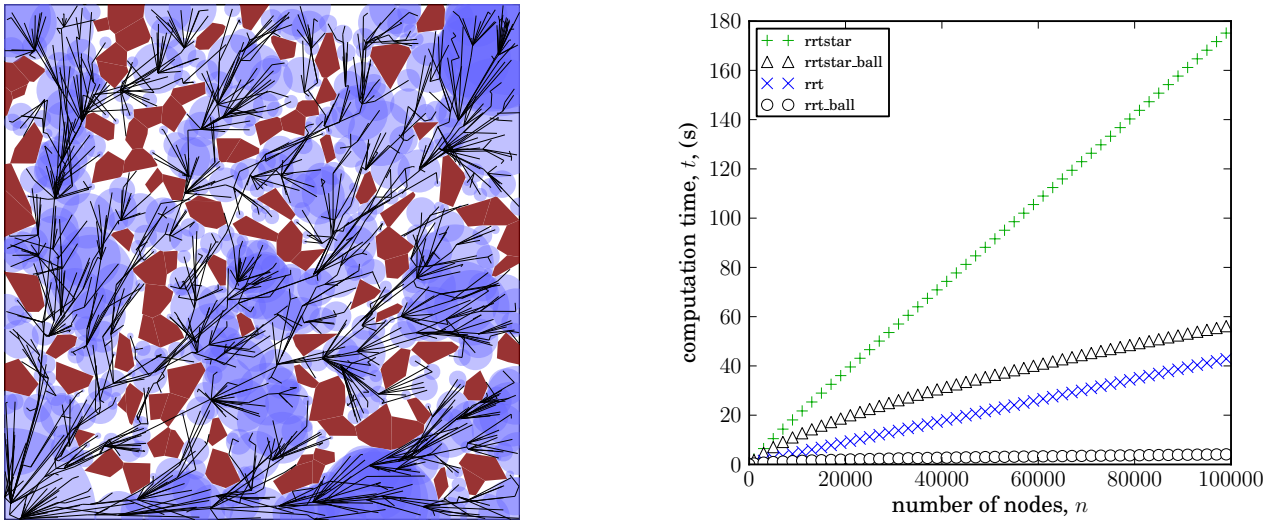

Fig. 15. (Left) The tree search tree and corresponding certificate set when $n = 2,000$ for RRT*. Tree edges are black and certificates are purple. (Right) Runtimes for the four combinations of RRT and RRT* with and without the use of collision checking certificates; methods using certificates have "ball" appended in the legend. Each datapoint represents the mean value over 30 trials.

8. Experiments

8.1. RRT and RRT with Certificates*

In this section we perform experiments in a simulated environment to evaluate the performance of the basic certificate method. In particular, both the RRT and RRT* algorithms are used both with and without our collision certificate method.

The simulated environment consists of a unit-square workspace with 150 randomly placed convex polygonal obstacles (see Figure 15, Left). A holonomic point robot starts at the bottom left corner and is tasked with reaching a goal region, a square with side length 0.1 units, at the top right corner. In this simple case the workspace and configuration space are equivalent $X = \mathcal{W}$. Point-proximity (i.e., nearest neighbor and k-nearest neighbor) queries are performed using a kd-tree Bentley (1975), while set distance queries are performed with a segment-Voronoi hierarchy Aurenhammer (1991).

30 trials are performed per each combination of planning algorithm with and without our collision checking modifications. All four algorithm combinations are run with the same initial random seed (i.e., so that all four use the sample sequence ω). Experiments are run on a 1.73 GHz Intel Core i7 computer with 4GB of RAM, running Linux. However, we note that the software implementation is single-threaded.

Figure 15 (Left) illustrates the set of collision-free balls and corresponding tree resulting after 2,000 sample points have been sampled by RRT*. Note that 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 (white). This graphically illustrates the convergence result from Section 5, i.e., collision checking becomes less likely as the number of samples increases.

Figure 15 (Right) depicts the wall-clock measured runtime vs. tree size for each combination of RRT and RRT* with and without the use of collision checking certificates. The proposed approach achieves greater time savings as the number of nodes in the graph increases. The RRT* time for 10^4 and 10^5 vertices is reduced by 40% and 70%, respectively, vs. the baseline implementation of RRT*. Similarly, the RRT time for 10^4 and 10^5 vertices is reduced by 70% and 90%, respectively, vs. the baseline implementation of RRT.

Using certificates increases the rate at which RRT* converges to the globally optimal solution due to the fact that performing less collision checking decreases average iteration time. Figure 16 (Left) shows the cost of the best path as a function of wall time. In general, the modified RRT* finds paths of lower cost significantly faster than the baseline RRT*.

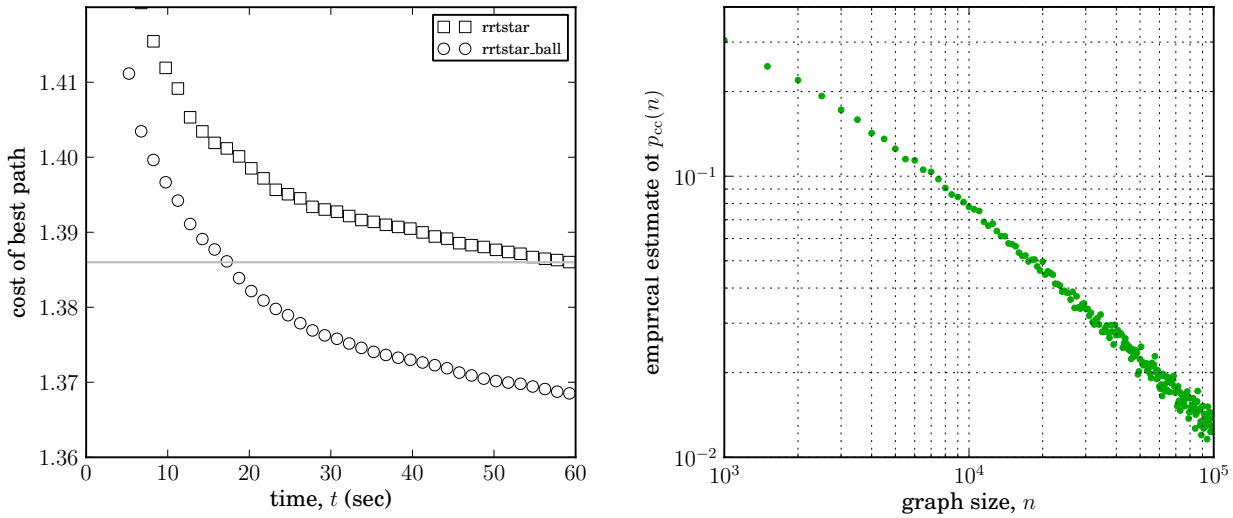

Fig. 16. (Left) RRT* best-path cost vs. time, with and without the proposed method. Points represent the mean over 30 trials. Using certificates yields significantly faster convergence. (Right) The experimentally observed chance of performing an explicit collision query vs. graph size. Note that only 1% of nodes require an explicit check when graph size is 100,000.

Figure 16 (Right) shows the mean 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. For example, only 1,000 nodes out of 100,000 (i.e., 1%) require an explicit check, on average, when the certificate method is used.

Figure 17 shows the single iteration runtime as a function of graph sizes (for all four algorithm combinations) vs. two different obstacle configurations. In particular, environments with 500 and 1,000 randomly generated obstacles. The 500 obstacles environment yields shorter iteration times, in general. However, when certificates are used, the difference between iteration times in either environment decreases. This is due to the fact that the certificate covering of the free space becomes complete, in the limit, and provides additional evidence of the utility of using certificates.

If planning is performed in a low-dimensional space with polytope obstacles, then an exact collision distance computation can be performed using a Voronoi diagram obstacle index (doing so is no more expensive than collision checking, in general). However, generating a Voronoi diagram in higher dimensional geometric spaces is prohibitively complicated. We now evaluate the utility of using certificates in conjunction with a version of RRT that has been designed for these more difficult cases. In particular, a kd-tree is used for point-proximity searches, and an axis-aligned bounding box tree is used to compute set distance queries and to perform collision checking (note that a lower bound on the actual collision distance is returned). Obstacles are randomly generated simplices.

Figure 18 (Left) illustrates the runtime as a function of graph size for the latter implementation of RRT, both with and without certificates. Figure 18 (Right) shows the observed frequency of performing an explicit collision check for a unit workspace in 2 to 5 dimensions when certificates are used.

Using RRT with certificates (i.e., certificates defined by a lower bound on collision distance, as described above) works better than standard RRT in 2 and 3 dimensions, roughly the same in 4 dimensions, and performs worse in 5 dimensions. In even higher dimensions there is a significant degradation in performance. Performance issues in higher-dimensional configuration spaces are further discussed in Section 9.2.

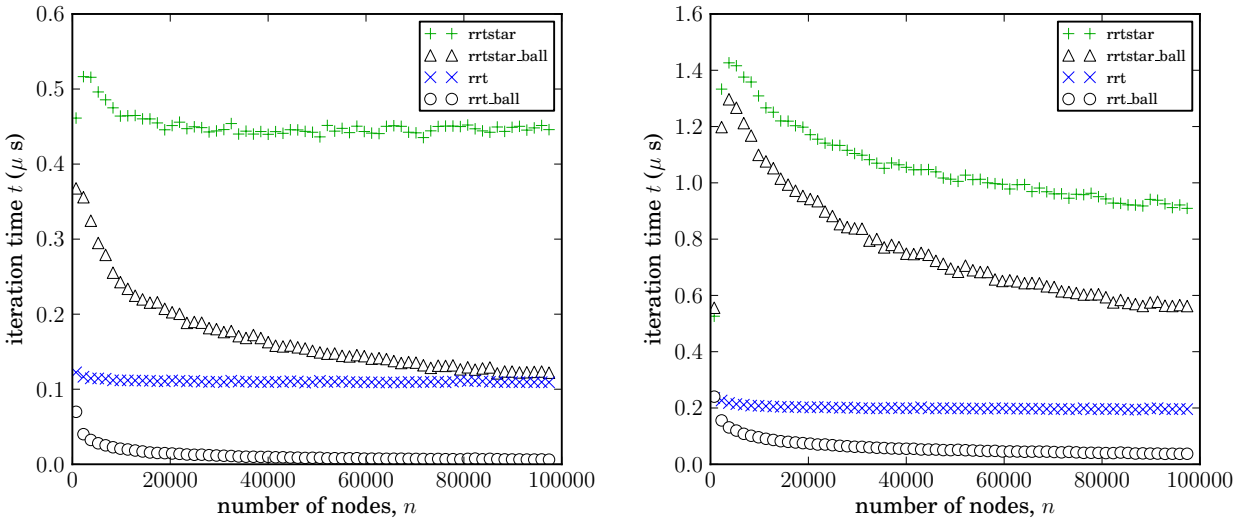

Fig. 17. The computation time of a single iteration vs. algorithm (plot style), mean over 30 runs, in a configuration with 500 and 1,000 obstacles (left and right, respectively). Methods using certificates have "ball" appended in the legend.

8.2. Workspace Certificates

In order to demonstrate the utility of workspace certificates and their ability to accelerate sampling-based planning algorithms we performed experiments on the piano mover problem in $\mathcal{W} \subset \mathbb{R}^3$ and $X \subset \mathbb{R}^D \times SO(3)$, as illustrated in Figure 9. The moving object is an L-shaped volume, decomposed into two convex collision volumes. The planning algorithm is PRM* Karaman and Frazzoli (2011), collision checking is done with FCL Pan et al. (2012), and proximity queries are done with brute force on a GPU. The cost of a path segment between two configurations (x_a, R_a) and (x_b, R_b) is taken to be $\|x_b - x_a\| + c_w \log\left(R_a^\top R_b\right)$, where c_w is a user defined parameter that affects the cost tradeoff between translational and rotational movement (we use $c_w = 1$). This cost function was chosen because it works well in practice. That said, the choice of cost function used by PRM* does not affect the results presented in this sub-section, as the latter are only concerned with certificate coverage and graph size vs. time. 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 $n_{\text{vertices}} \times m_{\text{faces}}$ vertex-face constraint Sturm polynomials (i.e., where n_{vertices} is the number of vertices on the polytopal robot and m_{faces} is the number of faces on the polytopal certificate), 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 19 shows the wall-clock total normalized runtime (runtime divided by number of points) with and without the use of a workspace certificate cache. We see that 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. That is, by 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

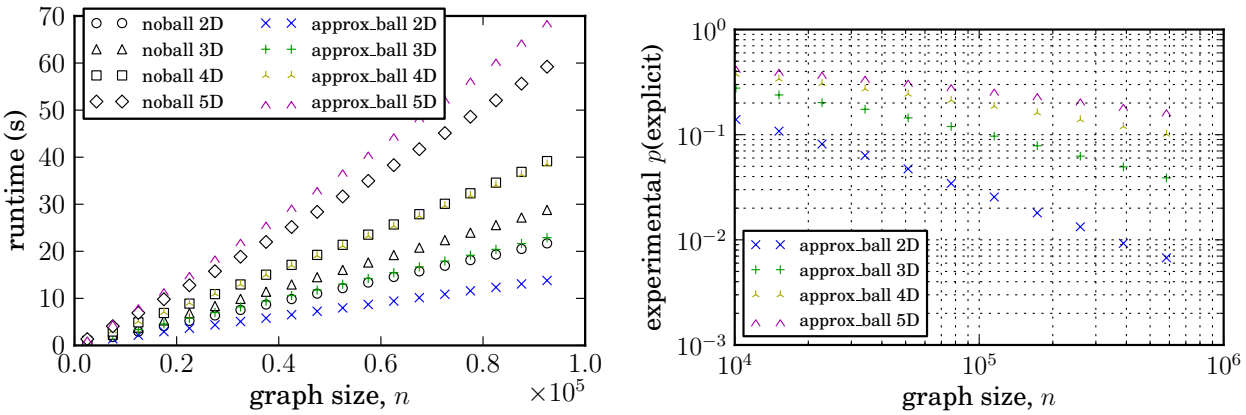

Fig. 18. Mean wall-clock runtime (Left) and experimental p_{explicit} (Right), over 20 runs, for RRT in a unit workspace $X = [0, 1]^D$ for several D. Methods using certificates based on an approximate distance bound are labeled as "approx ball" in the legend, while those using no certificates are labeled as "noball".

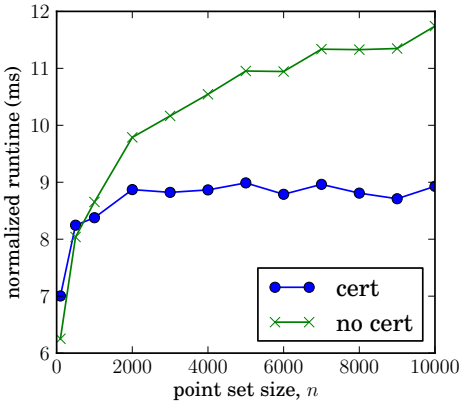

Fig. 19. Using workspace certificates accelerates continuous collision checking by up to 40%, resulting in a 40% total runtime reduction

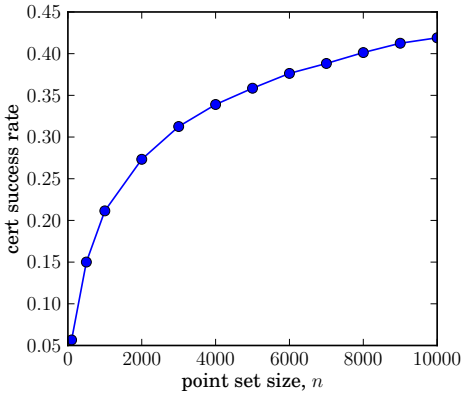

Fig. 20. The certificate success rate grows with the size of the point set, as the inter-configuration distance is shrinking.

the certificate increases. Figure 20 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 21 illustrates the that runtime of the cache-enabled implementation is only marginally improved over the baseline implementation when obstacles are tightly packed in this piano mover example, and when using conservatively dilated certificates. We can see in Figure 22 that the conservatism leads to a very low cache hit-rate in this case.

In this case, the higher coverage afforded by certificates generated from successive pruning (Algorithm 5) provides a more appropriate collision cache. Figure 23 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

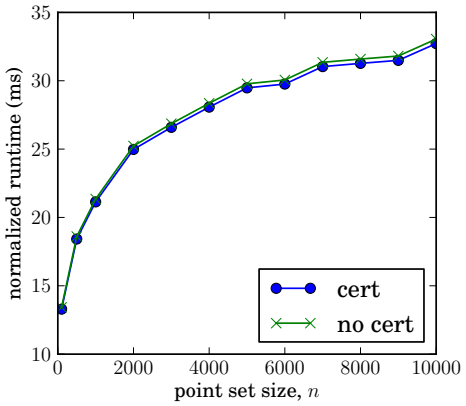

Fig. 21. When obstacles are close together ($< 1\times$ the length of the moving object) the benefit of the collision cache is greatly reduced.

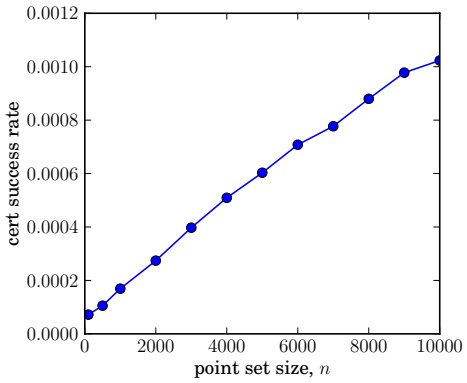

Fig. 22. The likelihood of the path to neighboring configurations lying within a sample's certificate volume remains small for even relatively large point set sizes.

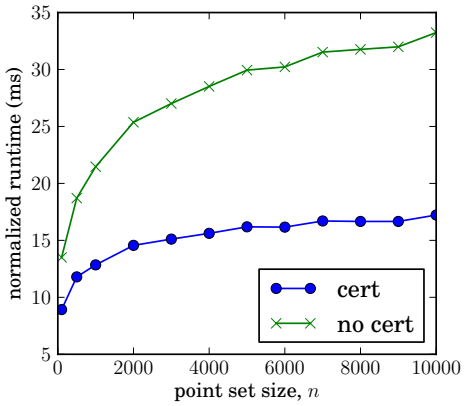

Fig. 23. Caching of certificates generated by successive pruning yields significantly improved runtime when obstacles are close together with respect to the size of the collision volume.

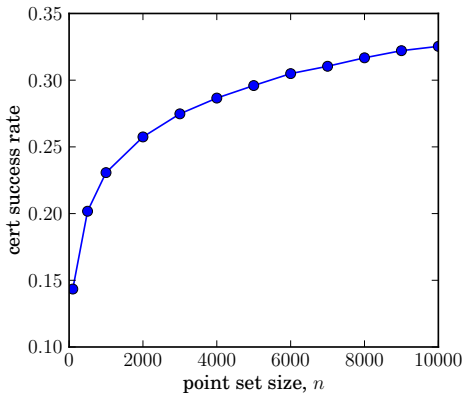

Fig. 24. Certificates generated by successive pruning provide a cache with a higher cache rate when obstacles are close together.

experiment. The trade-off between the two certificate generating methods is evident in Figure 24. The increased cost of generating the certificates is offset by the higher cache hit rate, reaching 30% after 10,000 samples.

8.3. Spaces with Symmetries

We now perform a number of experiments evaluating the effectiveness of using *Partial Certificate* and *Shared Projection* vs. the basic method (which we will refer to as *Basic Certificate*) for centralized multi-robot planning.

Experiments are performed with both RRT and RRT* and with teams containing $R = 1$ to 5 robots. The workspace $\mathcal{W} \subset \mathbb{R}^2$ used for all experiments appears in Figure 25. Robots 1, 2, 3, 4, and 5 start at the clock-wise positions of $9:00$, $6:00$, $3:00$, $12:00$, and $7:30$, respectively, and each robot has the goal of reaching the opposite side of the workspace. Robots are discs with radius 0.5 and holonomic, therefore the configuration space is $X = \prod_{i=1}^{R} X_i$. When an experiment is run with a team size $R < 5$ then robots $i > R$ are removed from the workspace.

Figures 26 and 27 show the average proportion of points that require a collision check (over 20 trials) for different team sizes (1 to 5 robots) vs. iteration number (1 to 10^5). Note that fractional values are possible for *Partial Certificate*

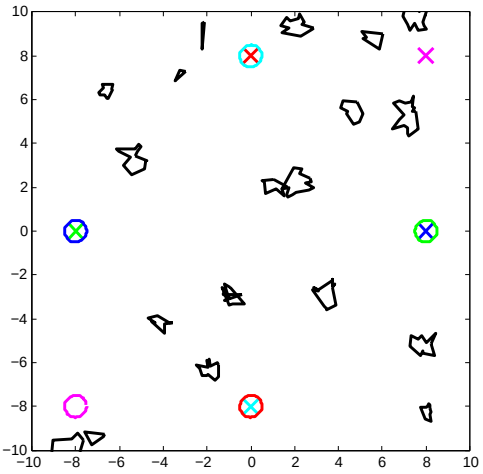

Fig. 25. The randomly generated workspace that is used for all centralized multi-robot team experiments. Robot starting locations and goals appear as circles and crosses, respectively. Note that a particular robot's starting location and goal have the same color. Obstacle appear black and have been randomly generated.

and *Shared Projection* when only some of the robots in a configuration require a check. Figure 26 shows results with RRT, while 27 shows results with RRT*.

Both *Partial Certificate* and *Shared Projection* require fewer collision checks at a particular iteration than the basic method.

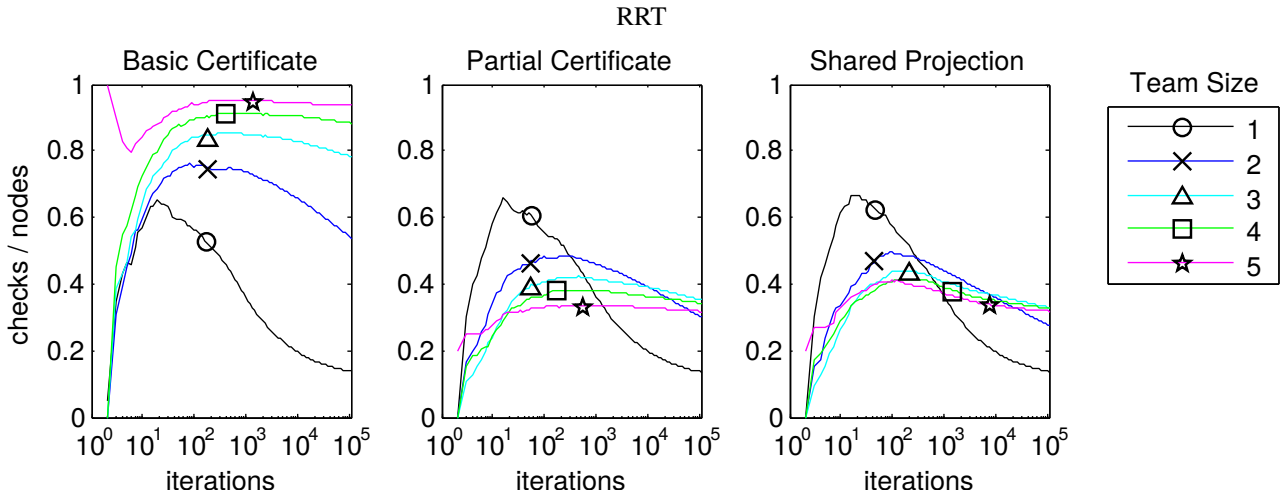

Fig. 26. Proportion of nodes requiring a collision check vs. team size (mean value over 20 trials), lower values are better. The RRT algorithm is used. It is important to note the difference vs. the left-most sub-figure.

9. Discussion

9.1. Removing the Collision Checking Bottleneck

Both analysis and experiments show that using collision certificates significantly reduces the proportion of time spent on collision checking. In particular, the expected time spent on collision checking per iteration approaches zero, in the limit, as the number of iterations approaches infinity (Proven in Theorem 1, and observed experimentally). Given that collision checking is widely believed to be the main computational bottleneck in sampling based motion planning, this result has the

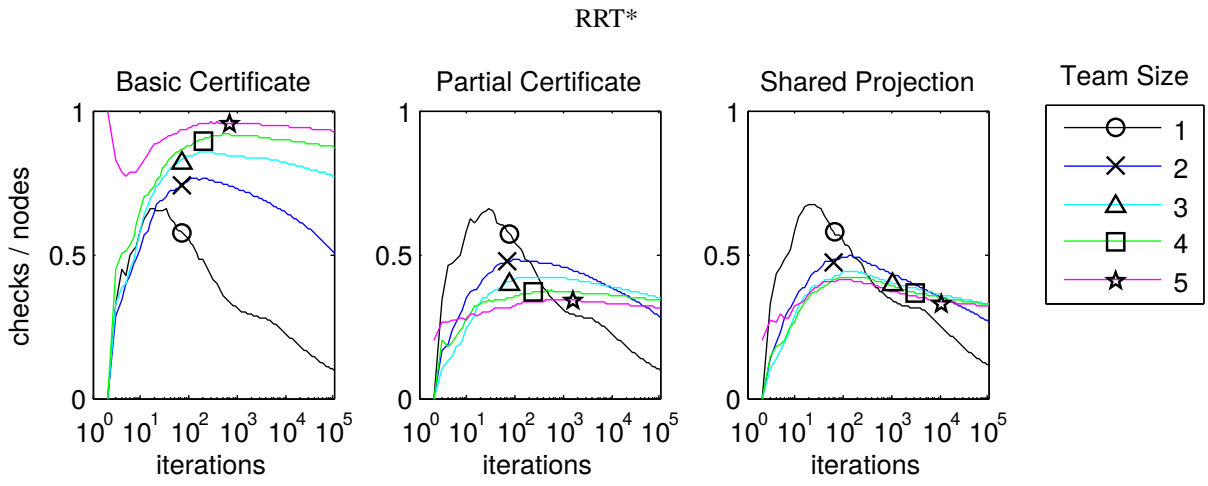

Fig. 27. Proportion of nodes requiring a collision check vs. team size (mean value over 20 trials), lower values are better. The RRT* algorithm is used. It is important to note the difference vs. the left-most sub-figure.

potential to improve the performance of nearly all sampling based motion planning algorithms. In general, those satisfying Assumptions (A1) and (A2) in environments that satisfy Assumption (A3).

9.2. Limitations in Higher-Dimensional Configuration Spaces

In our experiments evaluating configuration-space-based certificates (Section 8.1) we observe decreasing performance vs. configuration space dimensionality D. We believe that decreasing performance in higher dimensions is a direct result of the curse of dimensionality. In particular, the fact that $\lim_{D\to\infty} \frac{\mathscr{L}(\mathcal{B}_\ell)}{\mathscr{L}(X_{\ell'})} = 0$, where $\mathscr{L}(\mathcal{B}_\ell)$ is the hypervolume contained in a ball of radius ℓ, and $\mathscr{L}(X_{\ell'})$ is the hypervolume contained in a state space with diameter $\ell' > \ell$. Consequently, a certificate of a particular radius ℓ will cover proportionally less of the state space as dimensionality D increases; and thus have less chance of preventing future collision checks (although it is still expected to prevent an increasing number of collision checks vs. iteration and runtime).

This illustrates one reason why workspace certificates (see Sections 6 and 8.2) are especially desirable for problems with high-dimensional configuration spaces. That is, regardless of the dimensionality of the configuration space, the workspace is limited to 3 dimensions for "real-world" robotics problems.

9.3. Certificates in Configuration Space vs. Workspace

Certificates can be used in either the configuration space or the workspace. The version that should be used in any particular case will likely be determined by the relative ease of calculating the certificate in either space, as well as practical concerns such as the obstacle representation that chosen (or already implemented in a preexisting motion-planning code-base). Configuration space certificates require that a bound on the minimum distance between a point and the obstacle set can be calculated inexpensively—an assumption that may not hold in all configuration spaces or in all practical implementations. Workspace certificates require many (constant times) more operations per certificate because the robot is represented as a volume instead of a point, but are more generally applicable in practice because obstacle volumes tend to be easily representable in the workspace, and also have dimensionality fixed at 3 or less for real-world problems. We advise that the configuration space method be used when possible, and the workspace method when practical (e.g., for problems with high-dimensional configuration spaces).

9.4. Considerations for Configuration Space Certificate

The proposed approach is very general but there are some important implementation details to be aware of when using it. First, we require that points explicitly determined to be in-collision are kept in order to characterize the obstacle set (i.e., in addition to those that are collision-free). While the ratio of expected explicit collision checks vs. the total number of samples goes to zero as the total number of samples approaches infinity, the number of points required to truly characterize the obstacle set can be quite large, especially in high dimensions. That said, our method 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.

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.

9.5. Considerations for Workspace Certificate

Experimental results demonstrate 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. 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 is merged into the continuous collision checking procedure.

There is significant room for improvement in parameterizing the collision volumes over the methods that we have presented here. 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 procedures 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 (see Section 6 of the new manuscript).

9.6. Benefits vs. Different Types of Algorithms

Certificate methods may benefit multi-query algorithms (e.g. k-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 sparse environments, single query algorithms may find a solution before the end of the start-up phase (i.e., when long-term benefits of reduced collision checks have not yet offset the extra per-sample overhead of using certificates). Multi-query and optimal planners require more collision checks because they attempt many more nearest neighbor connections per node; thus present more opportunities for savings. The start-up phase is less of a concern for multi-query algorithms due to the fact that their use assumes graph construction, and thus collision checking, happens off-line.

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. Since the the source configuration is statically collision checked during the sampling phase, this initial certificate can be computed prior to any continuous collision checking. 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 a certificate.

One important negative theoretical result is that collision certificates will not necessarily be useful with versions of PRM that essentially create an r-disc-graph due to the fact that, in such algorithms, a non-zero proportion of new trajectories will pass through more than two certificates — and thus require standard collision checks.

9.7. Certificates in Spaces with Symmetries

Certificates provide additional benefits in spaces with symmetries, e.g., a centralized multi-robot team, where each robot operates within the same environment. Storing team certificates as the Cartesian product of individual robot certificates allows separate certification per each robot; thus, new points may only require a *fractional* new certificate to certify the lower-dimensional projections (i.e., robots) of a point that do not fall into a certificate. When all robots are identical, all certification can be performed in a single lower-dimensional projection of the space (e.g., the certificate of one robot can be used for all robots). Experiments show that both of these ideas provide significant reduction in the number of "real" collision checks that must be performed—even vs. the basic certificate method.

9.8. Certificates vs. Different Spaces

At any particular iteration, the expected benefits of using certificates are proportional to the relative amount of space that is covered by certificates, where the latter is a function of both obstacle clutter and space dimensionality. Thus, the benefits of our method tend to increase vs. iteration number. The marginal benefit of adding a new certificate is proportional to how much additional space it certifies. The marginal benefits of using certificates tend to increase relatively quickly vs. iteration number in spaces that are relatively free of obstacles (where certificates tend to be centered relatively far from obstacles and have large radii as a result), and in lower dimension spaces (where certificates of a particular radius tend to cover more space). In high dimensional spaces with many obstacles it may take many iterations before the marginal benefits become substantial; however, they will also likely provide a much greater reduction in computation time once they do (due to the fact that standard collision checking algorithms also suffer from decreased performance in high-dimensional spaces, see Section 5.2 for more details).

Our best advice to practitioners is that certificate use is likely to have similar effects in environments with similar characteristics (clutter, narrow passage size, etc.). Therefore, one can test how effective the method in particular types of environments before decided to use it (or not) in practice.

9.9. Possible Extensions to General Trajectories

Some of the subroutines in the current paper leverage the fact that, in many path planning problems, the feasible trajectory between two points is a straight line. In general, this is not the case for robotic systems subject to, e.g., differential constraints. Extensions to more general systems is relatively straightforward, assuming that it is possible (and tractable) to test if a trajectory is bounded by a certificate and/or where the trajectory intersects the certificate (or even a bound on the latter).

10. Conclusions

We propose a novel certificate based approach to collision checking in sampling-based algorithms. In summary, whenever a point must be collision checked using standard methods, we save a certificate that defines a subset of space around that point that is also collision free. New points that are sampled from within a certificate can skip "normal" collision checking because they are guaranteed to be collision free. Similarly, the subset of any trajectory that passes through a certificate is also collision free. In practice, checking the certification status of new points and trajectories can be combined with nearest neighbor queries that are already used as subroutines in sampling-based motion planning algorithms.

The certificate method 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. Although complexity analysis has always suggested that this should be the case, the high complexity of collision checking has prevented it from being realized in practice in most previous work.

Acknowledgments

This work was partially supported by the Office of Naval Research, MURI grant #N00014-09-1-1051, the Army Research Office, MURI grant #W911NF-11-1-0046, and the Air Force Office of Scientific Research, grant #FA-8650-07-2-3744.

The final version of this paper was completed while the second author was supported by the Control Science Center of Excellence at the Air Force Research Laboratory (AFRL) and "in residence" at AFRL.

A. Canny Interpolation

This section continues using the additional notation introduced in Section 6. For sampling-based motion planning, a requisite component is a "*local planning method*" or alternatively a "*steering function*" that, given two configurations $x_a, x_b \in X$, generates a trajectory σ between them. $\sigma : [0,1] \to X$, where $\sigma(0) = x_a$, and $\sigma(1) = x_b$. We use s as the interpolation parameter along σ, i.e., $s \in [0,1]$.

Recall that $F_{i,\sigma}(s)$ is the transform from the local (workspace) coordinate system of link i to that of its (workspace) parent as parameterized by $s \in [0,1]$. Thus, given the chain $J = \{1, \ldots, i\}$ of links through the armature from the global coordinate frame 0 to frame i, it is possible to construct a transformation from link i to the global coordinate frame as $\prod_{j \in J} F_{j,\sigma}(s)$. We now derive particular forms of $F_{i,\sigma}(s)$ that can be used to apply Canny's interpolation method to workspace certificates used in conjunction with sampling-based motion planning.

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 $d : X \times X \to \mathbb{R}$

$$\lim_{d(x_0,x_1)\to 0} {}^{J(\sigma)}\!/_{J^*} = 1,$$

where $J^* = \min_{\sigma \in \Sigma} J(\sigma)$ and Σ denotes the set of all possible paths between configurations x_0 and 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 (1986) 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 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 $\mathbf{T}_0$ 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_1.

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:

$$\mathbf{T}(s) = \mathbf{T}_0 + s\,(\mathbf{T}_1 - \mathbf{T}_0)\,, \qquad 0 \le s \le 1. \tag{2}$$

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 using the inverse (log) map from $SO(3)$ to $so(3)$:

$$\mathbf{v} = \frac{1}{\Delta\theta}\omega,$$

where $\omega = \log(\Delta R_{0,1})$ and $\Delta\theta = \|\omega\|$. Note that the magnitude of the direct rotation is at most π. If the magnitude of the rotation is exactly $\Delta\theta = \pi$, then the log map is undefined. There are two axis-angle rotations which satisfy Rodrigues' formula (i.e., the reverse transform from $so(3)$ to $SO(3)$). 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:

$$\mathbf{v}' = \frac{\mathbf{e}_0 \times \mathbf{v}}{\|\mathbf{e}_0 \times \mathbf{v}\|},$$

where $\theta' = \sin^{-1}\|\mathbf{e}_0 \times \mathbf{v}\|$, and $R' = \exp(\theta'\mathbf{v}')$, and $\mathbf{e}_0$ is the unit vector along the x axis. Within this intermediate frame, rotation about the vector ω is expressed as a rotation about the x axis. At an intermediate angle θ such that $0 \le \theta \le \Delta\theta$, 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:

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

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

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

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

$$\gamma = \tan\Delta\theta/2, \tag{3}$$

$$\theta(s) = 2 * \tan^{-1}(s\gamma). \tag{4}$$

Then we have $\tan(\theta(s)/2) = \tan\left(\tan^{-1}(s\gamma)\right) = s\gamma$. Note that this choice means that orientations are interpolated linearly along a central projection of the unit sphere in quaternion space.

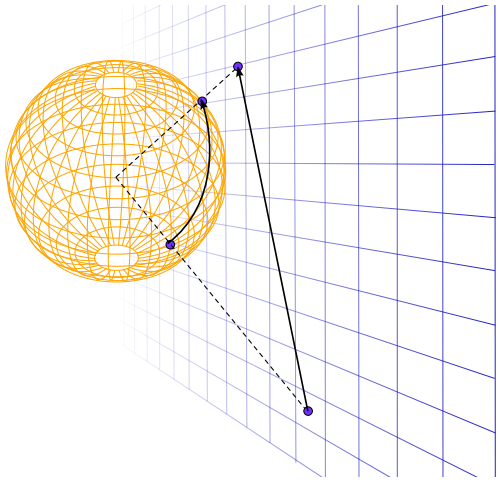

Fig. 28. Centroidal projection of a path between two points in S^2 to the projected points on $SO(2)$. The Canny interpolation between orientations is a similar projection in quaternion space (S^3). That is, this figure is a lower dimensional analogy to the method for projecting S^3 to $SO(3)$ that is used in this section to handle the rotational components of Canny interpolation.

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

$$\left(1+\gamma_{i,\sigma}^2 s^2\right) R_x(\theta) = \begin{bmatrix} 1+\gamma_{i,\sigma}^2 s^2 & 0 & 0 \\ 0 & -\gamma_{i,\sigma}^2 s^2 & -2\gamma_{i,\sigma} s \\ 0 & 2\gamma_{i,\sigma} s & -\gamma_{i,\sigma}^2 s^2 \end{bmatrix}, \tag{5}$$

$$\left(1+\gamma_{i,\sigma}^2 s^2\right) R_{i,\sigma}(s) = \left(1+\gamma_{i,\sigma}^2 s^2\right) R_x(s) \Delta R_{i,\sigma}. \tag{6}$$

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. This yields interference predicates which are univariate polynomials for the orientation of frame F_i along the path σ.

B. Additional Derivations for Constrained Motions in Articulated Robot

In this section we present a few additional details related to the application of Canny's method to articulated robots. 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

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

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.

References

Arslan, O. and Tsiotras, P. (2013). Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2421–2428. IEEE.

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

Basch, J., Comba, J. a., Guibas, L. J., Hershberger, J., Silverstein, C. D., and Zhang, L. (1999). Kinetic data structures: Animating proofs through time. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry*, SCG '99, pages 427–428, New York, NY, USA. ACM.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517.

Bialkowski, J. (2013). *Optimizations for Sampling-Based Motion Planning Algorithms*. PhD thesis, Massachusetts Institute of Technology.

Bialkowski, J., Otte, M., and Frazzoli, E. (2013a). Fast collision checking: From single robots to multi-robot teams. http://arxiv.org/abs/1305.2299.

Bialkowski, J., Otte, M., and Frazzoli, E. (2013b). Fast collision checking: From single robots to multi-robot teams. In *IEEE International Conference on Robotics and Automation: Crossing the Reality Gap - From Single to Multi- to Many Robot Systems*.

Bialkowski, J., Otte, M., and Frazzoli, E. (2013c). Free-configuration biased sampling for motion planning. In *International Conference on Intelligent Robots and Systems*, Tokyo.

Bialkowski, J., Otte, M., Karaman, S., and E., F. (2013d). Efficient collision checking in sampling-based motion planning. In *Algorithmic Foundations of Robotics X*, volume 69 of *Springer Tracts in Advanced Robotics*, page TBD. Springer Berlin / Heidelberg.

Bohlin, R. and Kavraki, L. E. (2000). Path planning using lazy PRM. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE.

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

Brock, O. and Kavraki, L. E. (2001). Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*,

volume 2, pages 1469–1474.

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

Deits, R. L. H. and Tedrake, R. (2014). Computing large convex regions of obstacle-free space through semidefinite programming. In *Workshop on the Algorithmic Foundations of Robotics*.

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

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

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

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

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

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

Hou, Q., Sun, X., Zhou, K., Lauterbach, C., and Manocha, D. (2011). Memory-scalable gpu spatial hierarchy construction. *Visualization and Computer Graphics, IEEE Transactions on*, 17(4):466 –474.

Hsu, D., Kavraki, L. E., Latombe, J.-C., Motwani, R., and Sorkin, S. (1998). On finding narrow passages with probabilistic roadmap planners. In Agarwal, P., Kavraki, L. E., and Mason, M. T., editors, *Robotics: The Algorithmic Perspective (WAFR '98)*, pages 141–154. A.K. Peters/CRC Press, Wellesley, MA.

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

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

Karavelas, M. and Yvinec, M. (2003). The voronoi diagram of planar convex objects. In Di Battista, G. and Zwick, U., editors, *Algorithms - ESA 2003*, volume 2832 of *Lecture Notes in Computer Science*, pages 337–348. Springer Berlin / Heidelberg.

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

Kavraki, L. E., Švestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580.

Kim, B. and Rossignac, J. (2003). 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. ACM.

Kirkpatrick, D., Snoeyink, J., and Speckmann, B. (2000). 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. ACM.

Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. (2009). Fast bvh construction on gpus. 28(2):375–384.

Lauterbach, C., Mo, Q., and Manocha, D. (2010). gProximity: hierarchical GPU-based operations for collision and distance queries. 29(2):419–428.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge university press.

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

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

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

Mirtich, B. (1998). V-clip: Fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208.

Otte, M. and Frazzoli, E. (2015). RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning.

Int. Journal of Robotics Research, to appear.

Pan, J., Chitta, S., and Manocha, D. (2012). FCL: A general purpose library for collision and proximity queries. In *IEEE Int. Conference on Robotics and Automation*, Minneapolis, Minnesota.

Pan, J. and Manocha, D. (2011). Gpu-based parallel collision detection for real-time motion planning. In Hsu, D., Isler, V., Latombe, J.-C., and Lin, M., editors, *Algorithmic Foundations of Robotics IX*, volume 68 of *Springer Tracts in Advanced Robotics*, pages 211–228. Springer Berlin / Heidelberg.

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

Ponamgi, M., Manocha, D., and Lin, M. (1997). Incremental algorithms for collision detection between polygonal models. *Visualization and Computer Graphics, IEEE Transactions on*, 3(1):51–64.

Redon, S., Kheddar, A., and Coquillart, S. (2000). 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.

Redon, S., Kheddar, A., and Coquillart, S. (2002). Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–287.

Redon, S., Kim, Y. J., Lin, M. C., and Manocha, D. (2004). 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. Eurographics Association.

Redon, S., Lin, M. C., Manocha, D., and Kim, Y. J. (2005). Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering*, 5(2):126–137.

Samet, H. (2006). *Foundations of multidimensional and metric data structures*. Morgan Kaufmann.

Schwarzer, F., Saha, M., and Latombe, J.-C. (2004). Exact collision checking of robot paths. In Boissonnat, J.-D., Burdick, J., Goldberg, K., and Hutchinson, S., editors, *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics*, pages 25–42. Springer Berlin / Heidelberg.

Sharifzadeh, M. and Shahabi, C. (2010). Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *Proc. VLDB Endow.*, 3(1-2):1231–1242.

Tang, M., Manocha, D., and Tong, R. (2010). Mccd: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models*, 72(2):7 – 23.

Tokuta, A. (1991). 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.

Yang, L. and LaValle, S. M. (2000). A framework for planning feedback motion strategies based on a random neighborhood graph. In *In Proc. Intl. Conf. on Robotics and Automation*, pages 544–549.

Yang, L. and LaValle, S. M. (2002). An improved random neighborhood graph approach. In *In Proc. Intl. Conf. on Robotics and Automation*, pages 254–259.

Yang, L. and LaValle, S. M. (2004). The sampling-based neighborhood graph: A framework for planning and executing feedback motion strategies. *IEEE Transactions on Robotics and Automation*, 20(3):419–432.

Yang, Y. and Brock, O. (2013). dapting the sampling distribution in PRM planners based on an approximated medial axis. In *In Proc. Intl. Conf. on Robotics and Automation*, pages 4405–4410.

Zhang, X., Redon, S., Lee, M., and Kim, Y. J. (2007). Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Trans. Graph.*, 26(3).