Computational Geometry 48 (2015) 589–605

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

On soft predicates in subdivision motion planning [☆]

Cong Wang [a], Yi-Jen Chiang [a,*], Chee Yap [b]

[a] Department of Computer Science and Engineering, Polytechnic School of Engineering, New York University, Brooklyn, NY, USA
[b] Department of Computer Science, New York University, New York, NY, USA

A R T I C L E   I N F O

Article history:
Available online 14 April 2015

Keywords:
Exact algorithms
Subdivision algorithms
Motion planning
Soft predicates
Resolution-exact algorithms

A B S T R A C T

We propose to design new algorithms for motion planning problems using the well-known Domain Subdivision paradigm, coupled with "soft" predicates. Unlike the traditional exact predicates in computational geometry, our primitives are only exact in the limit. We introduce the notion of **resolution-exact algorithms** in motion planning: such an algorithm has an "accuracy" constant $K > 1$, and takes an arbitrary input "resolution" parameter $\varepsilon > 0$ such that: if there is a path with clearance $K\varepsilon$, it will output a path with clearance ε/K; if there are no paths with clearance ε/K, it reports "NO PATH". Besides the focus on soft predicates, our framework also admits a variety of global search strategies including forms of the A* search and probabilistic search.

Our algorithms are theoretically sound, practical, easy to implement, without implementation gaps, and have adaptive complexity. Our deterministic and probabilistic strategies avoid the Halting Problem of current probabilistically complete algorithms. We develop the first provably resolution-exact algorithms for motion-planning problems in $\mathrm{SE}(2) = \mathbb{R}^2 \times S^1$. To validate this approach, we implement our algorithms and the experiments demonstrate the efficiency of our approach, even compared to probabilistic algorithms.

1. Introduction

A central problem of robotics is motion planning [4,20,21,10]. In the early 80's there was strong interest in this problem among computational geometers [15,32]. This period saw the introduction of strong algorithmic techniques with complexity analysis, and the careful investigation of the algebraic configuration space (C-space). In particular, Schwartz and Sharir [31] showed that the method of algebraic cell decomposition is a universal solution for motion planning. We introduced the retraction method in [24,33,34]. In the first survey of algorithmic motion planning [40], we also showed the universality of the retraction method. This method is now commonly known as the road map approach, popularized by Canny [8] who showed that its algebraic complexity is in single exponential time. Typical of algorithms in Computational Geometry, these exact motion planning algorithms assume a computational model in which exact primitives are available in constant time. Implementing these primitives exactly is non-trivial (certainly not constant time), involving computation with algebraic numbers.

In the 1990s, interest shifted back to more practical techniques. Today, the dominant approach is based on sampling, usually combined with randomization. The most well-known representative of the sampling approach is the **probabilistic**

[☆] An extended abstract of this paper appeared in *Proc. ACM Symposium on Computational Geometry (SoCG'13), pp. 349–358, 2013.* This work is supported by NSF Grant CCF-0917093 and DOE Grant DE-SC0004874.
[*] Corresponding author.
E-mail addresses: cwang05@students.poly.edu (C. Wang), chiang@nyu.edu (Y.-J. Chiang), yap@cs.nyu.edu (C. Yap).

http://dx.doi.org/10.1016/j.comgeo.2015.04.002
0925-7721/© 2015 Elsevier B.V. All rights reserved.

roadmap method (PRM) [19]. The idea is to compute a partial road map by random sampling of the C-space. PRM offers a computational framework for a large class of algorithms. Moreover, many variants[1] of the basic framework have been developed (see [21,10]). Most sampling methods take sample points in configuration space, but the recent paper from Halperin's group [29] takes sample (parametrized) subsets of configuration space. In an invited talk at the IROS 2011 Workshop on Progress and Open Problems in Motion Planning.[2] J.C. Latombe stated that the major open problem of such **Sampling Methods** is that they do not know how to terminate when there is no free path. In practice, one would simply time-out the algorithm, but this leads to issues such as "Climber's Dilemma" [16, p. 4] that arose in the work of Bretl (2005). We call this the **halting problem** of PRM, viewed as the ultimate form of what is popularly known as the "Narrow Passage Problem" [10, p. 216]. Latombe's talk suggested promising approaches such as Lazy PRM [3]. The theoretical foundation of PRM is based on two principles: probabilistic completeness, and fast convergence under certain "expansiveness" assumptions [18] about the environment. It is unclear how to check these assumptions on specific environments. For a comprehensive overview of motion planning, see Lavalle [21] and Choset et al. [10].

In this paper, we turn to a third popular approach [46] for motion planning, which we call **Subdivision Methods**. The general idea is to subdivide some bounded domain B_0, typically a subset of $\mathbb{R}^d$. In motion planning, the domain is a subset of configuration space. In its simplest form, the subdivision of B_0 can be represented as a **subdivision tree**, which is a generalization of binary trees ($d = 1$) or quad-trees ($d = 2$). An early reference for this approach is Brooks and Lozano-Perez [5]. Recent subdivision references include [46,2,45,12,26]. Manocha's group has been active and highly successful in producing practical subdivision algorithms for a variety of tasks, not just in motion planning (e.g., [38,36]). Domain subdivisions are sometimes known as "cell decomposition" (e.g., [46]), but we reserve "cell decomposition" for the approaches based on partitioning the configuration space into algebraic "cells" with bounded combinatorial complexity that are directly correlated with the combinatorial features on the obstacles (e.g., [30,40]). In contrast to such cells, the boxes in subdivision approaches are more related to "resolution". Nevertheless, subdivision that takes into account combinatorial complexity may be seen in [45,46]. Such kinds of subdivision algorithms offer tantalizing opportunities for new kinds of complexity analysis. Examples of such analysis may be seen in [28,35,7].

¶1. Contributions of this paper Although subdivision algorithms have been widely used by practitioners, their theoretical foundations have so far been lacking. This paper begins this task.

The notion of "resolution completeness" is widely used in the motion planning literature [10] but rarely analyzed (Section 5 discusses some issues). Our first contribution is to introduce the concept of **resolution-exact** (or **ε-exact**) **planners**. Such planners accept an input **resolution parameter** $\varepsilon > 0$. The planner has an **accuracy constant** $K > 1$, independent of the input, such that if there is a path of clearance $K\varepsilon$, it will output a path with clearance ε/K; if there is no path of clearance ε/K, it will output "NO PATH". As this paper shows, our definition allows us to devise planners that avoid the halting problem of PRM. Moreover, Section 5 notes that the usual concept of "resolution completeness" does not automatically solve the halting problem. But in what sense have we "solved" the halting problem? To be sure, we are *not* solving the halting problem for exact motion planning—this would require exact computation, something we wish to avoid in robotics. Instead, ε-exactness weakens the requirements for the "NO PATH" output. But is this just a trick to solve the halting problem by fiat? No, we argue that our weakening is not only justifiable, but desirable: good engineers know the limits of accuracy in their sensors, actuators, robot dimensions, etc. Path planning that depends on accuracy beyond these limits is not realistic, even dangerous. Note that when we output "NO PATH", we guarantee that there exists no path with clearance $K\varepsilon$ (this is the contrapositive of the statement just mentioned above: "if there is a path of clearance $K\varepsilon$, it will output a path with clearance ε/K"); no similar guarantees can come from PRM. With this information, users can choose ε based on engineering limits so that when we declare "NO PATH", no further search is warranted.

Our second contribution is the introduction of **soft primitives** for designing resolution-exact planners. Briefly, soft primitives are suitable numerical approximations of exact (hard) primitives. Such primitives are perhaps nascent in previous literature. But by making this idea explicit, we open up many new possibilities, as well as lay the groundwork for a systematic investigation of such algorithms. Such primitives are relatively easy to implement correctly (i.e., there are no "implementation gaps" in such algorithms).

Third, we design new planners based on soft predicates. These algorithms are the first explicit examples of resolution-exact planners. Our algorithms can use various search strategies, including probabilistic ones. Halting is guaranteed even in our probabilistic planners.

Our final contribution is the development and implementation of the first resolution-exact algorithms for rigid robots with configuration space $\mathtt{SE}(2) = \mathbb{R}^2 \times S^1$. Our experiments demonstrate their effectiveness.

2. On numerical computational geometry

Computational Geometry (CG) has traditionally concentrated on **Exact Methods**. The attractive features of exact algorithms are well-known. The drawback of such methods is exposed when we start to implement the algorithms. The inability

[1] A partial list includes Expansive-Spaces Tree planner (EST), Rapidly-exploring Random Tree planner (RRT), Sampling-Based Roadmap of Trees planner (SRT).

[2] http://www.cse.unr.edu/robotics/tc-apc/ws-iros2011. Sept. 30, 2011, San Francisco.

of Exact Methods to have wider impact on robotics and fields of Computational Sciences and Engineering (CS&E) despite the fact that geometric reasoning is central in these fields calls for a re-examination of our assumptions. We argue that Subdivision Algorithms, when[3] combined with soft primitives, offer a pathway for CG'ers to design new algorithms based on numerical approximations that are both practical and theoretically sound. Our soft primitives do not entail error analysis in the style of numerical analysis; rather, we rely on interval methods [22]. Algorithms in "Numerical CG" in this sense are distinctly different from the usual exact algorithms (see [43] for a general discussion).

One limitation of numerical primitives is that they are only complete in the limit. They also cannot detect degeneracies unless we use zero bounds [42]. Luckily for us, this is not an issue for resolution-exact planners and many other applications. But there are some problems (e.g., subdivision methods for Voronoi diagrams [39]) for which it is a challenge to handle degeneracies using only soft predicates. On the other hand, numerical methods have the advantage of greater generality, being applicable to non-algebraic problems where exact solutions are generally unknown (see a rare exception in [9]). Numerical CG will open up completely new areas for CG'ers.

The conventional wisdom of roboticists (see Choset et al. [10, p. 202]) is that Subdivision Methods are effective only up to "medium" degrees of freedom (DOFs) while Sampling Methods can be effective for much higher DOFs. This remark is borne out by currently implemented planners. But we do not see any inherent reason for this gap. Use of randomness is not a reason—as we will see, it is easy to deploy random search strategies in Subdivision Methods. We believe that the current reach of Subdivision Methods in motion planning can be greatly extended with better (perhaps randomized) search strategies. More generally, does the supposed limitation of subdivision extend beyond motion planning? Pessimistic views of subdivision often assume that the size of subdivision trees is exponential in the inverse resolution $1/\varepsilon$. This only shows that adaptivity in subdivision is critical. Some recent examples [28,6,35] suggest that adaptive approaches can guarantee optimal tree sizes, even in the worst case sense. Also the exploitation of Newton-type techniques in subdivision is very promising (e.g., [27]). All these point to many new opportunities for algorithmic development in Numerical CG.

3. Subdivision motion planning

In this section, we illustrate our approach with a basic motion planning problem. Fix a rigid robot $R_0 \subseteq \mathbb{R}^d$ and an obstacle set $\Omega \subseteq \mathbb{R}^d$. Both R_0 and Ω are closed sets. Initially we assume R_0 is a d-dimensional ball of radius $r_0 > 0$.

Suppose we want to compute a motion from an initial configuration α to some final configuration β. One of the best exact solutions when R_0 is a ball is based on roadmaps (i.e., retraction approach). Historically, the case $d = 2$ was the first exact roadmap algorithm [24]. For polygonal Ω, the roadmap is efficiently computed as the Voronoi diagram of line segments [41,13]. For $d = 3$, it is clear that a similar exact solution is possible. But here we see the limitations of exact solutions: there is no known exact algorithm for the Voronoi diagram of polyhedral obstacles [17,39]. The **configuration space** or C_{space} is $\mathbb{R}^d$ when R_0 is a ball. In general, we write $C_{space}(R_0)$ for the configuration of a robot R_0. Let $\alpha, \beta \in C_{space}$. The **footprint** of R_0 at α is the set $R_0[\alpha]$ comprising those points in $\mathbb{R}^d$ occupied by R_0 in configuration α (where R_0 is centered at α). We say α is **free** if $R_0[\alpha] \cap \Omega$ is empty; it is **semi-free** if it is not free but $R_0[\alpha]$ does not intersect the interior of Ω. Thus α is semi-free if $R_0[\alpha]$ is just touching Ω without penetrating it. Finally α is **stuck** if it is neither free nor semi-free. Thus, every configuration is classified as free, stuck or semi-free. We extend this classification to any set $B \subseteq C_{space}$: we say B is **free** (resp., **stuck**) if every $\alpha \in B$ is free (resp., stuck). Otherwise, B is **mixed** (i.e., contains at least one semi-free configuration). We thus defined the (exact) **classification predicate** $C : 2^{C_{space}} \to \{\texttt{FREE}, \texttt{STUCK}, \texttt{MIXED}\}$. This classification goes back to the beginning of subdivision motion planning in Brooks and Perez [5]. Our goal in soft primitive design is to avoid this exact predicate.

Let $C_{free} = C_{free}(R_0, \Omega) \subseteq C_{space}$ denote the set of free configurations. A **motion** from α to β is a continuous map $\mu : [0, 1] \to C_{space}$ with $\mu(0) = \alpha$ and $\mu(1) = \beta$. We call μ a **free motion** or more simply, a **path**, if its range $\{\mu(t) : t \in [0, 1]\}$ is contained in C_{free}. For sets $A, B \subseteq \mathbb{R}^d$, define their **separation** to be $\text{Sep}(A, B) := \inf\{\|a - b\| : a \in A, b \in B\}$. The **clearance** of a configuration $\gamma \in C_{space}$ is the separation between $R_0[\gamma]$ and Ω. The **clearance** of a path μ is the minimum clearance of $\mu(t)$ for $t \in [0, 1]$.

¶2. Subdivision trees Our main data structure is a subdivision tree $\mathcal{T}$ rooted at a box $B_0 \subseteq \mathbb{R}^d$. The nodes of $\mathcal{T}$ are subboxes of B_0, where boxes are closed subsets of full dimension d, and each internal node B is split into 2^i $(i = 1, \ldots, d)$ congruent subboxes which form the children of B. We remark that boxes B are axes-parallel and not assumed to be square, with **width** $w(B)$ and **length** $\ell(B)$ defined to be the lengths of the shortest and longest side (resp.). For convergence, we must assume that the **aspect ratio** $\ell(B)/w(B) \geq 1$ is bounded. Any box that can be obtained as a descendant of B_0 in a subdivision tree is said to be **aligned**. Let $m(B)$ denote the **midpoint** and **radius** $r(B)$ be the distance from $m(B)$ to any corner of B. For any real number $s > 0$, let $s \cdot B$ (or sB) denote the congruent box centered at $m(B)$ with radius $s \cdot r(B)$. Two boxes B, B' are **adjacent** if $B \cap B'$ is a **facet** F of B or of B', where facets refer to faces of co-dimension 1. Also, let $D_m(r)$ denote the **closed ball** centered at m with radius r.

[3] Subdivision Algorithms could also be combined with hard primitives. But to exploit the full power of Subdivision Methods we must consider soft primitives.

To allow domains of arbitrarily complex geometry, the input to our algorithm is an initial subdivision tree $\mathcal{T}_0$ whose leaves are arbitrarily marked ON or OFF. The set of ON-leaves forms a **subdivision** of the **region-of-interest** $\mathtt{ROI}(\mathcal{T})$ of the tree. Subsequently, $\mathcal{T}$ can be **expanded** at any ON-leaf B, by **splitting** B into 2^i $(1 \le i \le d)$ congruent subboxes who become the children of B.

¶3. A subdivision FindPath algorithm Our algorithm is given $\varepsilon > 0$ and an initial $\mathcal{T}_0$ rooted at B_0. The algorithm is parametrized by two subroutines: a classification predicate $C(B)$ for boxes, and a subroutine $\mathtt{Split}(B, \varepsilon)$ which returns a subdivision of B into 2^i (for some $i = 0, \dots, d$) congruent subboxes; the split subroutine is said to *fail* if $w(B) \le \varepsilon$ (in this case $i = 0$). Recall that we assume the aspect ratio $\ell(B)/w(B)$ to be bounded. We use $\mathcal{T}$ to search for a path in $B_0 \cap C_{free}$ as follows. Let $V(\mathcal{T})$ denote the set of free leaves in $\mathtt{ROI}(\mathcal{T})$. We define an undirected graph $G(\mathcal{T})$ with a vertex set $V(\mathcal{T})$ and edges connecting pairs of adjacent free boxes. We maintain the connected components of $G(\mathcal{T})$ using the well-known **Union-Find** data structure on $V(\mathcal{T})$: given $B, B' \in V(\mathcal{T})$, $Find(B)$ returns the index of the component containing B, and $Union(B, B')$ merges the components of B and of B'.

We associate with $\mathcal{T}$ a priority queue $Q = Q_{\mathcal{T}}$ to store all the mixed leaves B with width $w(B) > \varepsilon$. Let $\mathcal{T}.getNext()$ remove a box in Q of highest "priority". This priority is discussed below. We denote by $Box_{\mathcal{T}}(\alpha)$ (resp. $Box_{\mathcal{T}}(\beta)$) the leaf box in $\mathcal{T}$ containing α (resp. β). Let B be $Box_{\mathcal{T}}(\alpha)$ or $Box_{\mathcal{T}}(\beta)$ or a leaf box returned by $\mathcal{T}.getNext()$. We will expand B as follows: first call $\mathtt{Split}(B, \varepsilon)$. If $\mathtt{Split}(B, \varepsilon)$ fails, we return *fail* (note that it never fails if B is a box returned by $\mathcal{T}.getNext()$). Otherwise, each of the subboxes B' returned by $\mathtt{Split}(B, \varepsilon)$ is made a child of B. We label B' with the predicate $C(B')$. If $C(B') = \mathtt{FREE}$, we insert B' into $V(\mathcal{T})$ and into the union-find structure, and for each $B'' \in V(\mathcal{T})$ adjacent to B', we add an edge (B', B'') to the graph $G(\mathcal{T})$ and call $Union(B', B'')$. Finally, if $C(B') = \mathtt{MIXED}$ and $w(B') > \varepsilon$, we insert B' into Q. Thus, mixed boxes of width $\le \varepsilon$ are discarded (effectively regarded as STUCK). Now we are ready to present a simple but useful subdivision algorithm:

```
FindPath:
Input: Configurations $\alpha, \beta$, tolerance $\varepsilon > 0$, box $B_0 \in \mathbb{R}^d$.
Output: Path from $\alpha$ to $\beta$ in $Free(R_0, \Omega) \cap B_0$.
        Initialize a subdivision tree $\mathcal{T}$ with only a root $B_0$.
1.      While ($Box_{\mathcal{T}}(\alpha) \neq \mathtt{FREE}$)
            If (Expand $Box_{\mathcal{T}}(\alpha)$ fails) Return("No Path").
2.      While ($Box_{\mathcal{T}}(\beta) \neq \mathtt{FREE}$)
            If (Expand $Box_{\mathcal{T}}(\beta)$ fails) Return("No Path").
3.      While ($Find(Box_{\mathcal{T}}(\alpha)) \neq Find(Box_{\mathcal{T}}(\beta))$)
            If $Q_{\mathcal{T}}$ is empty, Return("No Path")
(*)         $B \leftarrow \mathcal{T}.getNext()$
            Expand $B$
4.      Compute a channel $P$ from $Box_{\mathcal{T}}(\alpha)$ to $Box_{\mathcal{T}}(\beta)$.
        Generate a path $\overline{P}$ from $P$ and Return($\overline{P}$)
```

In Step 4, the **channel** P is a sequence $(B_1, \dots, B_m)$ of boxes where B_i, B_{i+1} are adjacent. We also call P an **F-channel** since the B_i's are all free. We easily convert an F-channel into a path (or trajectory) which is a parametrized path $\overline{P} : [0, 1] \to C_{free}$ from α to β. It is also easy to produce $\overline{P}$ that satisfies reasonable constraints such as smoothness. This ability to generate a path is a benefit of subdivision methods over pure algebraic methods. Freeness is essential for our use of the extremely efficient Union-Find data structure. The use of Union-Find was proposed in [21].

In contrast to F-channels, Zhu–Latombe [46] uses **M-channels** (comprised of FREE or MIXED leaf boxes). Their idea is to attempt to find an F-channel along the "shortest" M-channel, by expanding all the MIXED boxes in the channel. Subsequent researchers (Barbehenn–Hutchinson [2] and Zhang–Manocha–Kim [45]) continued this approach. Barbehenn and Hutchinson [2,1] introduced the highly efficient Dijkstra or the related A* search. The challenge in their approach is how to efficiently update the A*-structure after expansions along the M-channel.

The routine $\mathcal{T}.getNext()$ in Step (*) is not fully specified; the correctness of our planner also does not depend on $\mathcal{T}.getNext()$. Nevertheless, it is critical for performance. There are many possible strategies for implementing *getNext*(). For instance, *getNext*() may return a random box in the queue, or use the BFS strategy. We can implement a Dijkstra-like or A* strategy by letting *getNext*() return a mixed leaf that is adjacent to the connected component of $Box_{\mathcal{T}}(\alpha)$. By alternating between two or more of these strategies, we get hybrid strategies. Another idea is to use some entropy criteria. Recent work on shortest-path algorithms in GIS road systems offers many other heuristics.

Our FindPath algorithm is not our claim to novelty. Nevertheless, it has interesting features, including great potential for adaptivity through its *getNext*() strategy. In contrast, non-adaptive uniform grid approaches (e.g., [21, p. 185]) are widely used. Although grids are superficially similar to subdivisions, grids use point-based operations while our theory is based on box (interval) operations (see Section 4). Uniform grid translates into breadth-first search strategy for *getNext*(), but we can do much better.

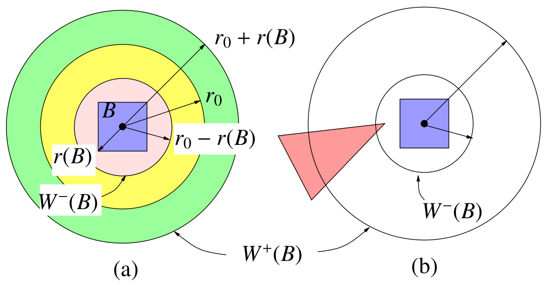


Fig. 1. (a) Domains $W^+(B)$ and $W^-(B)$. (b) Condition (S1) holds. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

4. Let us design soft predicates!

The preceding FindPath is based on the exact predicate $C(B)$. Our main interest in the Subdivision Method lies in its ability to replace $C(B)$ by some "soft" version $\widetilde{C}(B)$ which is easy to compute and correct in the limit. We now formalize this.

¶4. Soft predicates Let $\widetilde{C}(B)$ be a box predicate that returns a value in $\{\texttt{FREE}, \texttt{STUCK}, \texttt{MIXED}\}$. We call $\widetilde{C}$ a **soft version** of C if two conditions hold:

(**A1**) It is **conservative**, i.e., $\widetilde{C}(B) \neq \texttt{MIXED}$ implies $\widetilde{C}(B) = C(B)$.
(**A2**) It is **convergent**, i.e., if $\{B_i : i = 1, 2, \ldots, \infty\}$ converges to a configuration γ, then $\widetilde{C}(B_i) = C(\gamma)$ for large enough i.

We need a quantitative measure of the convergence rate. Let $0 \le \sigma \le 1$ and $\mathcal{B}$ be any class of boxes. A soft version $\widetilde{C}$ of C is said to be σ**-effective** (or have **effectivity factor** σ) for $\mathcal{B}$ if $C(B) = \texttt{FREE}$ implies $\widetilde{C}(\sigma B) = \texttt{FREE}$ for all $B \in \mathcal{B}$ (recall that σB is the congruent box centered at $m(B)$ with radius $\sigma \cdot r(B)$). One might imagine a stronger condition that $C(B) \neq \texttt{MIXED}$ implies $\widetilde{C}(\sigma B) \neq \texttt{MIXED}$ for all $B \in \mathcal{B}$, but our current definition suffices for our main Theorem A. For example, we will prove that our soft predicates below are effective for any class $\mathcal{B}$ of boxes with bounded aspect ratio.

We now design soft predicates $\widetilde{C}$ assuming $\Omega \subseteq \mathbb{R}^d$ is a polyhedral set, and the boundary of Ω is partitioned into a simplicial complex comprising relatively open cells of each dimension. For simplicity, assume $d = 2$. These cells are called **features** of Ω. The features of dimensions 0 and 1 are called **corners** and **edges** (resp.). Each box B is associated with three sets: its **outer domain** $W^+(B)$, **inner domain** $W^-(B)$, and **feature set** $\phi(B)$. When the robot $R_0 \subseteq \mathbb{R}^2$ is a ball of radius r_0, $W^+(B) \subseteq \mathbb{R}^2$ and $W^-(B) \subseteq \mathbb{R}^2$ are defined as the disks $D_{m(B)}(r_0 + r(B))$ and $D_{m(B)}(r_0 - r(B))$, respectively. See Fig. 1(a). If $r_0 < r(B)$, then $W^-(B)$ is empty. Also, $\phi(B)$ comprises the features of Ω that intersect $W^+(B)$. We call B **simple** if one of the following conditions holds:

(**S0**) Its feature set $\phi(B)$ is empty. Equivalently, no feature of Ω intersects its outer domain $W^+(B)$.
(**S1**) Some feature of Ω intersects its inner domain $W^-(B)$. Thus (S1) holds in Fig. 1(b) because of the red triangle obstacle.

The soft predicate $\widetilde{C}$ can now be defined: for our purposes, we only need to define $\widetilde{C}(B)$ for aligned boxes B. Thus we can use induction by depth. If B is non-simple, declare $\widetilde{C}(B) = \texttt{MIXED}$. Else if (S1) holds, declare $\widetilde{C}(B) = \texttt{STUCK}$. Otherwise, (S0) holds and clearly B is either free or stuck, and we define $\widetilde{C}(B) = C(B)$ accordingly.

We now come to computing $\widetilde{C}(B)$, but only in the context where B is a leaf of a subdivision tree. Observe if B' is a child of B, then $W^+(B')$ is contained in $W^+(B)$. This implies the following **distributional approach** of computing $\phi(B)$ is valid: when we expand B, we can distribute the features in $\phi(B)$ to each of its children. Note that a feature can be given to more than one child, or to no child (when it intersects no $W^+(B')$). Moreover, we can check the conditions (S1) and (S0) during this distribution. Finally, if (S0) holds, we determine $\widetilde{C}(B)$ as follows: $\widetilde{C}(B) = \texttt{FREE}$ (resp. $\texttt{STUCK}$) iff $m(B)$ is outside (resp. inside) the obstacle Ω. To decide between these two cases, note that by a linear search of the non-empty set $\phi(B.\texttt{parent})$, we can find the feature f in $\phi(B.\texttt{parent})$ that is closest to $m(B)$. We have 2 possibilities: (1) f is an edge. Assume that edges are oriented so that we can decide using a orientation test whether $m(B)$ is inside or outside Ω in the neighborhood of f. (2) f is a corner. We call f a **convex** (resp., **concave**) **corner** if, for any sufficiently small ball D centered at f, the set $D \cap \Omega$ is a convex (resp., concave) set. Every corner is either convex or concave. Moreover, f is convex iff $m(B)$ is outside Ω (iff B is free).

Suppose Ω is given as the union of a set of polygons that may overlap (this situation arises in Section 7). Moreover, $\phi(B)$ is defined to comprise features in these (possibly overlapping) polygons. We extend the above FREE/STUCK test for (S0) as follows: again linearly search $\phi(B.\texttt{parent})$, and for *each* obstacle polygon S appearing in $\phi(B.\texttt{parent})$, find the feature $f \subseteq \partial S$ that is closest to $m(B)$. Then $m(B)$ is outside Ω (and B is free) iff $m(B)$ is outside *all* such polygons S.

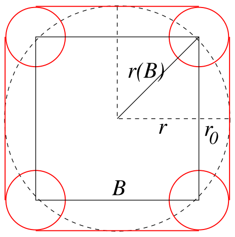


Fig. 2. Effectivity factor $1/\sqrt{2}$.

Lemma 1. *The predicate $\widetilde{C}$ is a soft version of C for the ball robot $R_0 \subseteq \mathbb{R}^2$. When boxes are squares, $\widetilde{C}$ has an effectivity factor $\sigma = 1/\sqrt{2}$. More generally, if our boxes have aspect ratio at most $\alpha \geq 1$, then the effectivity factor is $\sigma = 1/\sqrt{1+\alpha^2}$.*

Proof. To see the effectivity factor for square boxes, suppose that $C(B) = \texttt{FREE}$ for some square B. Referring to Fig. 2, we see that the region bounded by the outer four red segments and the outer four red circular arcs does not intersect the obstacle set Ω. Clearly, the dotted circle also does not intersect Ω. Note that this dotted circle is centered at $m(B)$ with radius $r(B) = r + r_0$, and it is the boundary of the outer domain $W^+(\sigma B)$ of box σB whose radius is r, where $r = r(B)/\sqrt{2}$. This means that $\sigma = 1/\sqrt{2}$ and we have $\widetilde{C}(\sigma B) = \texttt{FREE}$. Therefore, $\widetilde{C}$ has an effectivity factor $\sigma = 1/\sqrt{2}$. In general, let B be a rectangular box with dimensions $2w \times 2\alpha w$. If B is free, there is a disc centered at $m(B)$ of radius $w + r_0$ that does not intersect Ω. This disc is the outer domain of σB where $\sigma = 1/\sqrt{1+\alpha^2}$. $\square$

The proof easily generalizes to ball robots in every dimension. We can now use the soft predicate $\widetilde{C}$ instead of the exact predicate C to get a resolution-exact algorithm. This will be proved below. Of course, doing this for the disc robot is no great achievement since the exact algorithm is actually quite practical too. But it lays the groundwork for generalization to more complicated robots for which exact methods are no longer viable.

¶5. Implementability We claim that our algorithm is easy to implement correctly. We have designed our predicates so that they are reduced to comparison of "distances" between sets. In particular, a feature f is in $\phi(B)$ iff

$$\mathrm{Sep}(m(B), f) \leq r(B) + r_0 \tag{1}$$

where $\mathrm{Sep}(A, B)$ is the separation between sets A and B. Notice that (1) is a comparison of two exact (!) expressions. There are implicit square roots in these expressions, so an exact implementation would be expensive. But we are not obliged to implement soft predicates exactly—this cannot be said for hard predicates. We provide a simple implementation method: for any numerical expression x, let $\square(x)$ or $\square x$ denote any closed interval $[a, b]$ that contains x. If the interval has width at most 2^{-p}, we also write $\square_p(x)$. *Assume that for any expression x and any given p, we can compute some $\square_p(x)$.* This can be achieved with any software bigFloat package (e.g., GMP [14], MPFR [23]). We define the "lax comparison" $\preceq$ on intervals whereby $[a, b] \preceq [a', b']$ holds iff $a \leq b'$. Note that the "strict comparison" would be $b \leq a'$. We implement the test (1) using this lax comparison:

$$\square_p(\mathrm{Sep}(m(B), f)) \preceq \square_p(r(B) + r_0) \tag{2}$$

where $p = -\lg r(B)$. Let $\widehat{C}(B)$ be the "implemented" version of $\widetilde{C}(B)$.

Lemma 2. *$\widehat{C}(B)$ is a soft predicate for $C(B)$.*

Proof. Recall that $\phi(B)$ is the set of features belonging to the box B. Suppose $\widehat{\phi}(B)$ is the set of features that belong to B when we use the lax comparison $\preceq$. The key observation is that $x \leq y$ implies $\square_p x \preceq \square_p y$. This shows that $\phi(B) \subseteq \widehat{\phi}(B)$. The lax comparison (2) implies that

$$\mathrm{Sep}(m(B), f) - r(B) \leq (r(B) + r_0) + r(B)$$

or, $\mathrm{Sep}(m(B), f) \leq 3 \cdot r(B) + r_0$. This shows that the extra features in $\widehat{\phi}(B)$ must intersect the disc $W^+(3B)$ (the outer domain of $3B$). If a sequence of boxes B_i converges to a point q as $i \to \infty$, we see that $\widehat{\phi}(B_i) \to \phi(q)$. This implies that the approximate classification $\widehat{C}(B_i)$ also converges to $C(q)$. $\square$

¶6. Improvements We can improve the convergence of our soft predicates. In practice, and typical of subdivision approaches, such improvements can be quite significant (e.g., see [39]). Let us define the set $\phi(B)$ slightly differently, by recognizing two regimes for boxes. In the "small B regime", i.e., $r(B) < r_0$, we compute $\phi(B)$ as before. In the "large B regime", i.e., $r(B) \geq r_0$, we can define $\phi(B)$ to comprise those features that intersect the box αB where $\alpha = 1 + \sqrt{2} r_0 / r(B)$. Checking if a feature intersects αB is simple. This new definition should generally result in smaller sizes for $\phi(B)$. For a simple implementation, condition (S1) could be omitted; its role is to provide an early stuck decision.

5. Resolution exactness

We have designed some non-trivial algorithms under our scheme. We now clarify what sort of algorithms these are. Informally, our algorithms are "resolution complete". There are slightly variant definitions, but a typical (e.g., [37]) definition says "a planner is resolution complete if it finds a path if one exists provided the resolution parameters are selected small enough". This definition does not[4] say what happens if there is no path. Some formulations appear to assume that the resolution is not given but the planner has to search for it. Of course, this search would not terminate if there is no path. Our algorithms in Section 4 (and in Section 6 as well) have an explicit input $\varepsilon > 0$, called the **resolution parameter**. It is essential that ε be different from 0. To use this parameter, we recall the concept of "clearance". Here is an attempt to define resolution completeness with a converse: (i) *if there is a path with clearance ε, the planner will find a free path*, and (ii) *if there is no path with clearance ε, it will report "NO PATH"*. Taken together, this pair of statements cannot be the correct, as it implies that, with sufficient resolution, we can detect the case where the clearance is exactly ε, a feat that only Exact Methods can achieve (in which case we might as well design algorithms with $\varepsilon = 0$). What is missing in current discussions of resolution completeness is the concept of an **accuracy constant** $K > 1$. We say that a planner is **resolution-exact** if there exists an (accuracy) constant $K > 1$ that is independent of the input (but may depend on the algorithm) such that:

- If there is a path with clearance $K\varepsilon$, it outputs a path with clearance ε/K.
- If there is no path with clearance ε/K, it reports "NO PATH".

What if the maximum clearance of free paths lies strictly in the range $(\varepsilon/K, K\varepsilon]$? According to this definition, the planner is free to report a path or "NO PATH". In our Theorem A below, we prove that this cannot be avoided! *This indeterminacy is the necessary price to pay for resolution-exactness*. In our view, this price is not a serious one because the user has the option to decrease the ε parameter as desired. Of course, if we decrease ε to ε/K, the indeterminacy will reappear for input instances that only have paths with clearance in the range $(\varepsilon/K^2, \varepsilon]$. But as argued in Section ¶1 there is no infinite regress if we know some hard engineering limits of how much clearance a path should have.

The result of Theorem A below concerns our algorithm Exact FindPath in Section ¶3 in the 2D case, assuming that all boxes are squares and we use the exact classifier predicate $C(B)$. Recall that in our Exact FindPath algorithm, we subdivide a box only if its width $w(B)$ is *larger than* the input resolution parameter $\varepsilon > 0$. So the smallest boxes in the subdivision tree $\mathcal{T}$ have width t with $\varepsilon/2 < t \le \varepsilon$. Now consider the "full expansion" of the subdivision tree $\mathcal{T}$ whose leaves are of the smallest size possible. Recall from Section ¶3 that a channel is a sequence $(B_1, \dots, B_m)$ where B_i, B_{i+1} are adjacent. We are interested in a free channel where $\alpha \in B_1$ and $\beta \in B_m$.

Lemma 3. *If there exists a motion μ with clearance $\delta = \sqrt{2}\varepsilon$, then our* Exact FindPath *algorithm outputs a path with clearance $\varepsilon/4$.*

Proof. Consider the "full expansion" of $\mathcal{T}$ as mentioned above, where the leaves have a width t with $\varepsilon/2 < t \le \varepsilon$. Consider the subset $\mathcal{A}$ of such leaves that cover μ. We claim that each leaf box in $\mathcal{A}$ is *free*: let p be a point in μ and B_ℓ be the leaf box where p lies; since the diagonal of B_ℓ is $\sqrt{2}t \le \sqrt{2}\varepsilon = \delta$, B_ℓ lies entirely within the "clearance region" of p and thus B_ℓ is free. Therefore, $\mathcal{A}$ consists of free leaf boxes of width t that covers μ; in other words, $\mathcal{A}$ is a **free channel** Π that covers μ.

Since there exists a free channel Π connecting α and β, our Exact FindPath algorithm will find *some* free channel Π' connecting α and β (Π' is *not necessarily* Π, but at least Π exists as a candidate to be found by our algorithm). This can be justified as follows: consider the subdivision tree $\mathcal{T}$ *produced by our algorithm*. It produces a subdivision of $\mathtt{ROI}(\mathcal{T})$, and for each free box B in $\mathcal{A}$, there is a corresponding free leaf B^* in $\mathcal{T}$ that contains B. These free leaves B^*, after pruning redundancies, yield a free channel Π^* that covers Π. By definition of the correctness of any path finding algorithms, a free channel Π' connecting α and β will be found iff there exists a free channel Π^* connecting α and β.

Note that Π' consists of free aligned boxes connecting from $B(\alpha)$, the free (aligned) box containing α, to $B(\beta)$, the free (aligned) box containing β. Since each free box in Π' has width at least t, we can construct a rectilinear path P, from the box center a of $B(\alpha)$ to the box center b of $B(\beta)$, through the free boxes in Π' where each point of P is away from the box boundary by a distance at least $t/2$ (see Fig. 3 for an example), and thus P has clearance $t/2 > \varepsilon/4$.

Our final reported path P_f is given by $P_f = \overline{\alpha a} \cup P \cup \overline{b\beta}$. It remains to show that $\overline{\alpha a}$ has clearance $\varepsilon/4$ (and similarly for $\overline{b\beta}$ by the same argument). The key point is to use the fact that α belongs to μ and thus has a clearance $\delta = \sqrt{2}\varepsilon$. We consider the following two cases.

Case (1): The width of $B(\alpha)$ is t. Then for any point $q \in \overline{\alpha a}$, $d(\alpha, q)$ is at most half of the diagonal of $B(\alpha)$, i.e., $d(\alpha, q) \le \sqrt{2}t/2 \le \sqrt{2}\varepsilon/2 = \delta/2$. However, α has clearance δ, and thus $q \in \overline{\alpha a}$ has clearance $\delta - d(\alpha, q) \ge \delta/2 > \varepsilon/4$.

Case (2): The width of $B(\alpha)$ is at least $2t$. We refer to Fig. 4, where the boundaries of the inner box and of $B(\alpha)$ are apart by a distance $t/2$. Clearly, any point of $\overline{\alpha a}$ lying inside the inner box has clearance at least $t/2 > \varepsilon/4$. Now consider the portion of $\overline{\alpha a}$ outside the inner box. Without loss of generality, suppose such portion lies in the green shaded rectangle

[4] In Computer Science, "completeness" concepts typically have some "if-and-only-if" connotation. Otherwise, they might be qualified as "partial completeness". E.g., "partial correctness" of programs, or "partial decidability" of problems, etc.

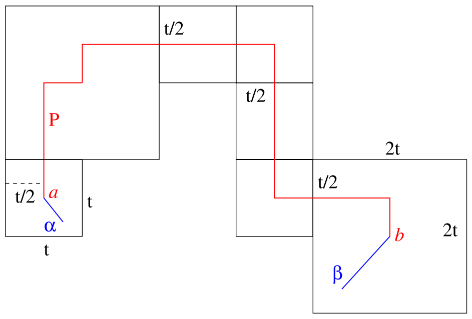


Fig. 3. Path P from a to b with clearance $t/2 > \varepsilon/4$. A canonical path P^* consists of $\overline{\alpha a}, \overline{b\beta}$ and essential path P, with essential clearance $t/2$.

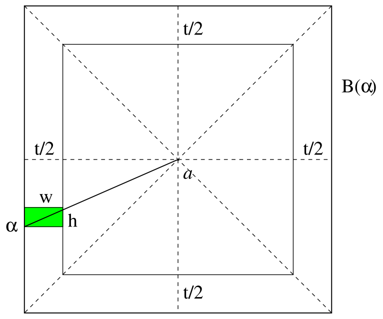


Fig. 4. Segment $\overline{\alpha a}$ has clearance $\varepsilon/4$.

and the slope of $\overline{\alpha a}$ is in the range $[0, 1]$ (for other cases the slopes are in the ranges $(1, \infty)$, $[-1, 0)$, and $(-\infty, -1)$ and symmetric arguments apply). Note that $w = t/2$ and $h \le w$ (since the slope of $\overline{\alpha a}$ is in $[0, 1]$), the diagonal of the green shaded rectangle is at most $\sqrt{2}t/2 \le \sqrt{2}\varepsilon/2 = \delta/2$, i.e., any point $q \in \overline{\alpha a}$ lying in the green shaded rectangle has $d(\alpha, q) \le \delta/2$. Since α has clearance δ, such q has clearance $\delta - d(\alpha, q) \ge \delta/2 > \varepsilon/4$. Therefore, every point of $\overline{\alpha a}$ has clearance $\varepsilon/4$. □

We define an **essential path** to be a path from the center a of a free box $B(\alpha)$ containing α to the center b of a free box $B(\beta)$ containing β (e.g., path P in Fig. 3). A **canonical path** P^* consists of line segments $\overline{\alpha a}, \overline{b\beta}$, and an essential path P from a to b. Note that the major task in motion planning is to find an essential path P, while making P canonical by adding $\overline{\alpha a}$ and $\overline{b\beta}$ is straightforward. We define the **essential clearance** of a canonical path to be the clearance of its essential path (see Fig. 3).

Lemma 4. *If there is no free canonical path with essential clearance $\varepsilon/4$, then our* Exact FindPath *algorithm reports "NO PATH".*

Proof. We prove the contrapositive: When our Exact FindPath algorithm finds a path, there exists a free canonical path with essential clearance $\varepsilon/4$. Indeed, when our algorithm finds a free path, it finds a set of free aligned boxes connecting from $B(\alpha)$ to $B(\beta)$. Since each such free box has width at least t, we can construct an essential path, which is a rectilinear path P where each point of P is away from the box boundary by a distance at least $t/2$ (see Fig. 3). Clearly $\overline{\alpha a} \cup P \cup \overline{b\beta}$ is a free canonical path with essential clearance at least $t/2 = \varepsilon/4$. □

Putting together Lemmas 3 and 4, we have the following results for 2D, assuming that all boxes are squares and we use the exact classifier predicate $C(B)$.

Theorem A *(Hard predicate). Let $K_0, k_0 \ge 1$ and consider our planner* Exact FindPath.

(i) *For $K_0 = \sqrt{2}$, if there is a path with clearance $K_0\varepsilon$, then our planner outputs a path with clearance $\varepsilon/4$.*
(ii) *For $k_0 = 4$, if there is no free canonical path of essential clearance ε/k_0, then our planner reports "NO PATH".*

The results in (i) and (ii) are tight in the following sense:

(i′) *If $K_0 < \sqrt{2}$, there are obstacle inputs Ω admitting paths with clearance $K_0\varepsilon$, but our planner reports "NO PATH".*
(ii′) *If $k_0 < 4$, there are obstacle inputs Ω admitting no paths of clearance ε/k_0 but our planner outputs a path.*

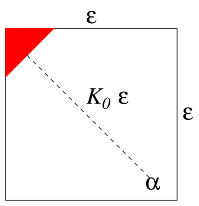


Fig. 5. Proof of Theorem A (i′).

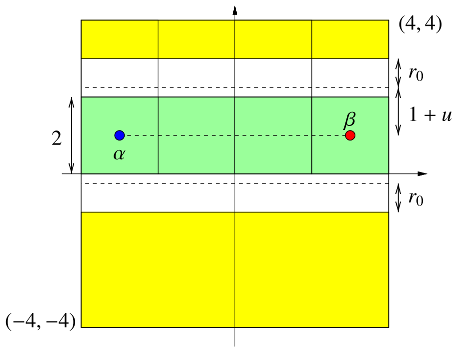


Fig. 6. Proof of Theorem A (ii′). (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

Proof. (i) and (ii) are Lemmas 3 and 4 respectively.

(i′). Consider any $K_0 < \sqrt{2}$. We can have an obstacle input Ω such that it admits a path with clearance $K_0\varepsilon$, where α lies in the aligned box $B = B(\alpha)$ of our subdivision tree, with width $w(B) = \varepsilon$, but the robot center cannot be placed in the red shaded triangle region (see Fig. 5). Note that the diagonal of B is $\sqrt{2}\varepsilon$ and α can still have clearance $K_0\varepsilon$. However, B is a mixed box with $w(B) = \varepsilon$ and thus the expansion of B fails. Therefore, our planner reports "NO PATH".

(ii′). Suppose $k_0 < 4$. Let $\delta := 4 - k_0$. We now construct an input for our algorithm. Let

$$B_0 = [-4, 4] \times [-4, 4], \quad r_0 < 1.9, \quad \varepsilon = 4 - (\delta/2),$$

$$\alpha = (-3, 1), \quad \beta = (3, 1),$$

and Ω is the union of two half spaces, $\{(x, y) \in \mathbb{R}^2 : y \le -u - r_0\}$ and $\{(x, y) \in \mathbb{R}^2 : y \ge 2 + u + r_0\}$ for some small $u \in (0, 2 - r_0)$ to be determined. See Fig. 6, where Ω is shown in yellow.

Using an exact classification predicate, we will subdivide until we obtain a "linear" channel of boxes from $B(\alpha)$ to $B(\beta)$ (shown in green in Fig. 6). Note that each box in this channel has width 2 and the straightline path from α to β has clearance $1 + u$. So our algorithm will output the straightline path from α to β. Note that this path has clearance $1 + u$ (which is in fact the largest clearance possible, but the algorithm does not actually know this clearance). We shall choose u to fulfill

$$1 + u < \frac{\varepsilon}{k_0} = \frac{4 - (\delta/2)}{4 - \delta}, \tag{3}$$

i.e., the largest clearance of any paths, $1 + u$, is less than ε/k_0, so that Ω admits no paths of clearance ε/k_0. Here (3) is true iff $u < (1/2)(\delta/k_0)$. Note that the ratio δ/k_0 could be large, but recall that $u \in (0, 2 - r_0)$ (where $r_0 < 1.9$) from our construction. So we can pick $u = \min\{(1/3)(\delta/k_0), 1.9 - r_0\}$ to fulfill both conditions. □

Theorem A implies an accuracy factor $K = 4$, but it is clear that K can be reduced by adjusting our algorithm to use the resolution parameter ε in a more equitable way.

The general form of this result is perhaps no surprise, but the accuracy constants might not be what we initially expect, since we are talking about an "exact algorithm". There are several sources for loss of accuracy: first, subdivision boxes are "aligned" with the integer grid in the sense that their coordinates are dyadic numbers. Second, the width of our smallest boxes, the ε-MIXED boxes, lies between $\varepsilon/2$ and ε. The third is the use of soft predicates. In particular, what is the accuracy of our prototype algorithm in Section ¶3 when using the soft predicates of Section ¶4? Recall from Lemma 1 that when boxes are squares, our soft predicate $\widetilde{C}$ has an effectivity factor $\sigma = 1/\sqrt{2}$. In our algorithm, we can replace our input resolution parameter with $\bar{\varepsilon} = \sigma\varepsilon$, i.e., we split boxes until the smallest box width is between $\bar{\varepsilon}/2$ and $\bar{\varepsilon}$ (between $\sigma\varepsilon/2$ and $\sigma\varepsilon$).

Lemma 5. *If there exists a motion μ with clearance $\delta = \sqrt{2}\varepsilon$, then our algorithm using soft predicate $\widetilde{C}$ outputs a path with clearance $\sigma\varepsilon/4$.*

Proof. This is a "soft version" of Lemma 3. Consider the "full expansion" of our subdivision tree $\mathcal{T}$; now the smallest boxes have width σt (instead of t). Look at the subset $\mathcal{A}$ of such leaf boxes that cover μ. For each such leaf box B_ℓ, let B_ℓ/σ be the box centered at $m(B_\ell)$ with width t. We claim that B_ℓ/σ is free: let p be a point on μ that lies in B_ℓ; clearly p also lies in B_ℓ/σ. Since the diagonal of B_ℓ/σ is $\sqrt{2}t \le \sqrt{2}\varepsilon = \delta$, B_ℓ/σ lies entirely within the "clearance region" of p and thus B_ℓ/σ is free. Therefore, we have $C(B_\ell/\sigma) = \mathtt{FREE}$. By the effectivity factor σ for $\widetilde{C}$, $C(B_\ell/\sigma) = \mathtt{FREE}$ implies $\widetilde{C}(B_\ell) = \widetilde{C}(\sigma(B_\ell/\sigma)) = \mathtt{FREE}$. Therefore, we can use $\widetilde{C}$ to classify each B_ℓ to be free, and thus to classify $\mathcal{A}$ as a **free channel** covering μ. This is the same as the free channel $\mathcal{A}$ covering μ in the proof of Lemma 3, but now each channel box has width σt rather than t. The rest of the proof of Lemma 3 carries over, with the reported path having a clearance $\sigma\varepsilon/4$ rather than $\varepsilon/4$. □

Lemma 6. *If there is no free canonical path with essential clearance $\sigma\varepsilon/4$, then our algorithm using soft predicate $\widetilde{C}$ reports "no path".*

Proof. This is a "soft version" of Lemma 4. Again we prove the contrapositive: When our algorithm finds a path, there exists a free canonical path with essential clearance $\sigma\varepsilon/4$. The proof of Lemma 4 carries over, but now each free aligned box has width σt rather than t, and thus the essential clearance is at least $\sigma t/2 = \sigma\varepsilon/4$. □

We re-state Lemmas 5 and 6 together in the following.

Theorem B *(Soft predicate). With the same assumptions as Theorem A, but with the exact predicate $C(B)$ replaced by a soft predicate $\widetilde{C}(B)$ with effectivity factor σ, we have:*

(i) *For $K_0 = \sqrt{2}$, if there is a path with clearance $K_0\varepsilon$, then our planner outputs a path of clearance $\sigma\varepsilon/4$.*
(ii) *For $k_0 = 4$, if there is no free canonical path with essential clearance $\sigma\varepsilon/k_0$, then we report "NO PATH".*

This implies that the accuracy factor K now becomes $4/\sigma$. In general, we have:

Corollary. *If the Exact version of our planner has an accuracy factor of K, then the Soft version of our planner using a soft predicate with effectivity factor σ has an accuracy factor of K/σ.*

6. Rotational degree of freedom

In this section we develop resolution-exact algorithms for the case where robot $R_1 \subseteq \mathbb{R}^2$ has a simple shape: R_1 is a triangle that is contained in a circumscribing disc R_0 of radius r_0. Now, $C_{space} = SE(2) = \mathbb{R}^2 \times S^1$. Each box $B \subseteq C_{space}$ is decomposed as $R \times \Theta$ where $R \subseteq \mathbb{R}^2$ is a rectangle and $\Theta \subseteq S^1$ is an angular range. We also write $m(R), r(R), w(R)$ to denote the previously defined $m(B), r(B), w(B)$. Two boxes $B = R \times \Theta$ and $B' = R' \times \Theta'$ are **adjacent** iff R and R' are adjacent, and Θ and Θ' are adjacent in the circular geometry of S^1.

¶7. *ε-smallness* We discuss the issue of **splitting** $B = R \times \Theta$: we can obviously simply split B into 8 congruent children. However there are two issues. First of all, we may want to avoid splitting the angular range when B is in the "large regime": as long as $w(R) \ge r_0$, we can approximate R_1 by the disc R_0 and ignore the rotational degree of freedom. So B is split into 4 children (based on splitting R but not Θ). When B is in the "small regime", i.e., $w(R) < r_0$, we begin to split the angular range. But here, we want to treat Θ differently from R. To understand this, recall that we previously do not split a box R when $w(R) \le \varepsilon$. Let us say that R is ε-**small** if $w(R) \le \varepsilon$. We need a similar criterion for Θ: say Θ is ε-**small** if $|\Theta| \le \varepsilon/r_0$. This assumes that angles are in radians, and Θ is represented as an interval $[\theta_1, \theta_2] \subseteq [0, 2\pi]$; also $|\Theta|$ is defined as $\theta_2 - \theta_1$. Finally, we say that $B = R \times \Theta$ is ε-**small** if both R and Θ are ε-small. We now define our procedure $\mathtt{Split}(B, \varepsilon)$ as follows: to split B, we split R and Θ separately. These are not split if they are already ε-small. Thus, splitting B will result in 2^i children for $i = 0, 1, 2, 3$. The following justifies our definition of ε-smallness:

Lemma 7. *Assume $0 < \varepsilon \le \pi/2$. If $B = R \times \Theta$ is ε-small and R is a square, then the Hausdorff distance between the footprints of R_1 at any two configurations in B is at most $(1 + \sqrt{2})\varepsilon$.*

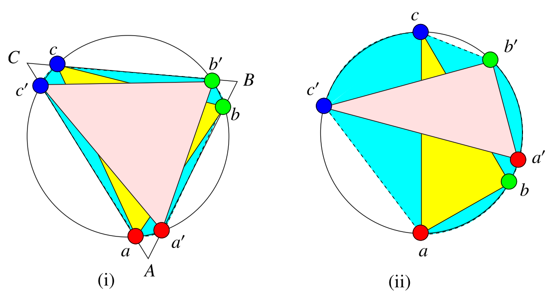


Fig. 7. Shaded areas represent round triangles: (i) $aa'bb'cc'$ with 3 straight edges, (ii) $ab'cc'$ with 2 straight edges. In (i), the round triangle $aa'bb'cc'$ is $T \cap D$ where T is the triangle (A, B, C) and D is the (white) disk.

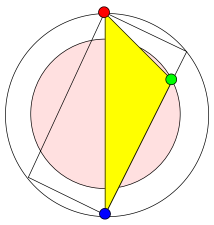


Fig. 8. Enclosing circle of enclosing rectangle for obtuse triangle: their rotation. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

Proof. This result uses the fact that if we rotate R_1 by θ about the center of R_0, then the vertices of R_1 move by at most $2r_0 \sin(\theta/2) \le r_0\theta \le \varepsilon$ since $\sin\theta \le \theta$ for θ in the said range. Also, the translational distance between any two configurations in B is at most $\sqrt{2}\varepsilon$. $\square$

¶8. Soft predicate for rotation We now design a soft version $\widetilde{C}$ of C. The strategy follows the case of disc robot: we define the feature set $\phi(B)$ associated with a box $B = R \times \Theta$ as comprising those features of Ω that intersect the set $W^+(B)$ where $W^+(B)$ is a "round triangle" associated with B. We call RT a **round triangle** if it is given as the intersection of a disc D with a triangular region T (see Fig. 7).

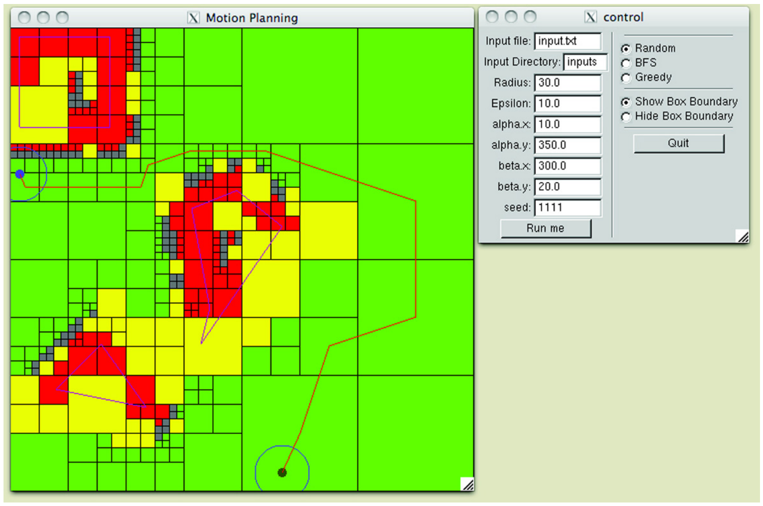
For any real number s, we denote the s-**expansion** of various shapes $S \subseteq \mathbb{R}^2$ by $(S)^s$. If $S = D(m, r)$ is a disc, $(D)^s := D(m, r + s)$. If S a convex polygon P, then $(P)^s$ is the polygon obtained by shifting each defining line of its edges in an outward normal direction by a distance of s. Typically, P is a triangle or a box. Finally, if S is a round triangle $RT = D \cap T$, then $(RT)^s = (D)^s \cap (T)^s$. Note that $(RT)^s$ depends on the representation D and T. Usually we have $s \ge 0$; if $s < 0$, then RT is shrunk and $(RT)^s$ may be the empty set.

Consider a configuration $(m, \theta) \in C_{space}$; the footprint $R_1[m, \theta]$ is a triangle in $D_m(r_0)$. Let $RT(m, \Theta)$ be the convex hull of the union of these footprints as θ ranges over Θ. Note that $RT(m, \Theta)$ is a round triangle. In Fig. 7, we show $RT(m, \Theta)$ for two choices of R_1. We define the **outer domain** $W^+(B)$ to be the $r(B)$-expansion of $RT(m(B), \Theta)$. As before, the **feature set** $\phi(B)$ is defined as those features of Ω that intersect $W^+(B)$. Finally, we define $\widetilde{C}(B)$ using $\phi(B)$ as before. Computing $\widetilde{C}(B)$ in the context of expanding a subdivision tree is also similar.

Lemma 8. *$\widetilde{C}$ is a soft version of C for the robot R_1. Also $\widetilde{C}$ is effective for the class of squares.*

¶9. Improvements We can improve by providing some heuristic for quick detection of stuck boxes, in analogy to Property (S1) for a disc robot. For any box B, we can define an **inner domain** $W^-(B)$ such that if any feature intersects $W^-(B)$, then B is stuck. Indeed $W^-(B)$ can be defined to be a suitable triangle: in Fig. 7(i), $W^-(B)$ is the triangle bounded by the lines $\overline{ab'}$, $\overline{bc'}$ and $\overline{ca'}$.

Our definition of R_0 as the circumscribing circle for R_1 can lead to extremely large radius r_0 when R_1 is a very thin obtuse triangle. We describe an alternative: when R_1 is obtuse, we will define R_0 as the smallest disc containing R_1. Choose the robot origin to be the center of this new R_0. Thus, the longest side of R_1 will be a diameter of R_0, and one vertex of R_1 will be in the interior of R_0. This is illustrated in Fig. 8 where the red and blue vertices of R_1 define a diameter of the circle C, but the green vertex lies on a concentric inner circle C'. The interior of C' is pink in this figure. If we slightly rotate the robot R_1 counter-clockwise about the center of C, the boundary of the area swept by R_1 will include a small arc of C'. The convex hull of this swept area will comprise of 3 arcs, two arcs from C and one arc from C'. We can again construct a soft predicate based on such a convex hull, but this variation has not been implemented.

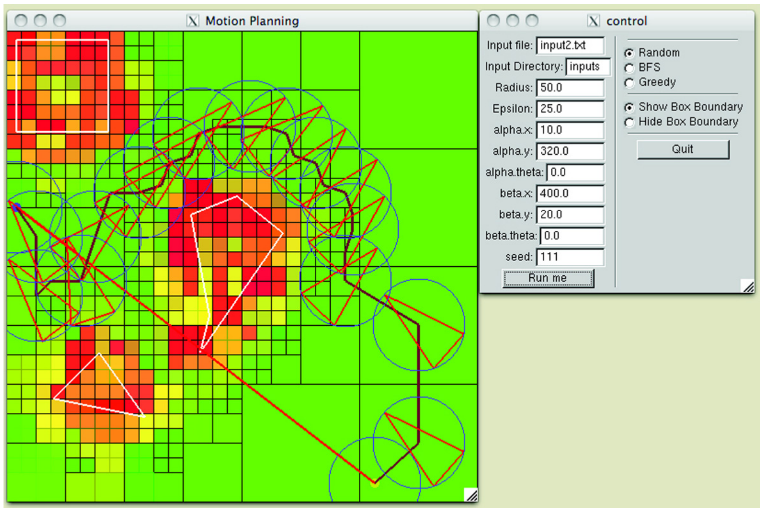


Input file: input.txt
Input Directory: inputs
Radius: 30.0
Epsilon: 10.0
alpha.x: 10.0
alpha.y: 350.0
beta.x: 300.0
beta.y: 20.0
seed: 1111

[✓ Random] BFS Greedy

[✓ Show Box Boundary] Hide Box Boundary

Fig. 9. Subdivision for disc robot. Color scheme: Green = `FREE`, Red = `STUCK`, Yellow = ε-large `MIXED`, Grey = ε-small `MIXED`. (For interpretation of the references to color in this caption, the reader is referred to the web version of this article.)



Input file: input2.txt
Input Directory: inputs
Radius: 50.0
Epsilon: 25.0
alpha.x: 10.0
alpha.y: 320.0
alpha.theta: 0.0
beta.x: 400.0
beta.y: 20.0
beta.theta: 0.0
seed: 111

[✓ Random] BFS Greedy

[✓ Show Box Boundary] Hide Box Boundary

Fig. 10. Subdivision for triangular robot: translational boxes show blended colors. (For interpretation of the references to color in this caption, the reader is referred to the web version of this article.)

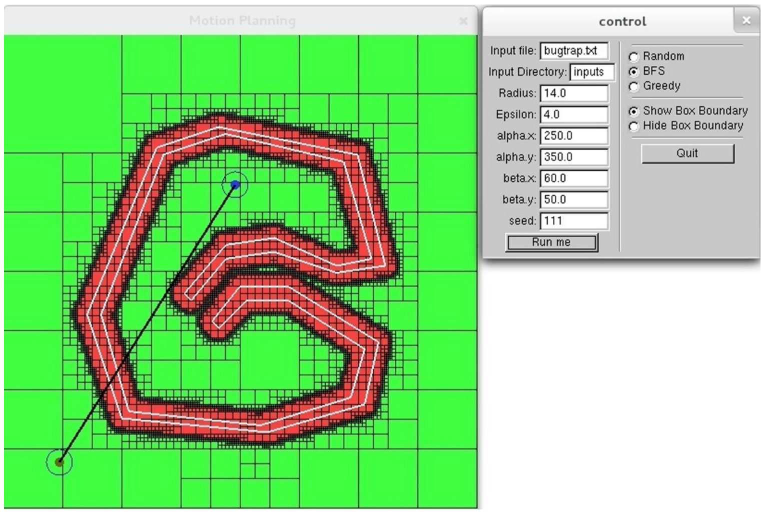
7. Experimental results

We have implemented in C++ the planner for disc and triangle robots described in this paper. Our code, data and experiments are freely distributed with the `Core Library`[5] and is available on our project web page. The platform for the experiments was a Linux Fedora 16 OS with a 3.4 GHz Intel Quad Core CPU, and 16 GB RAM. Our current implementation does not apply the technique of "lax comparison" in Section ¶5. Instead, we use machine arithmetic. This is because in our examples, the subdivision boxes are large enough that machine arithmetic suffices. In the future, we plan to provide error estimates to justify this expedient.

Figs. 9 and 10 show the GUI interface of our implementation of the disc and triangle robots, respectively. Since $C_{space} = \mathbb{R}^2$ for a disc, it is straightforward to visualize the box classification in a subdivision, as illustrated in Fig. 9. For a triangle robot, each box $B \subseteq C_{space} = \mathbb{R}^2 \times S^1$ has the form $B = B_t \times B_r$ where B_t is the translational component. The color of B is projected onto B_t. We display a blended color of all boxes that project to B_t.

We implemented the following three search strategies: *Breadth First Search (BFS)*, *Random (RAN)*, and *Greedy Best First (GBF)*. In *BFS* and *Random*, we follow the original scheme described in Section ¶3, where a union-find data structure is used to determine if the leaf boxes $Box_{\mathcal{T}}(\alpha)$ and $Box_{\mathcal{T}}(\beta)$ belong to the same connected component of the adjacency

[5] http://cs.nyu.edu/exact/core/download/core/.

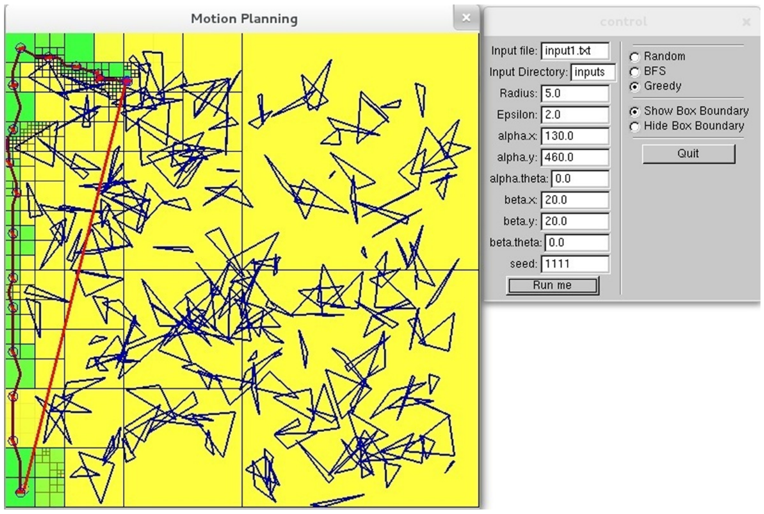


Input file: bugtrap.txt
Input Directory: inputs
Radius: 14.0
Epsilon: 4.0
alpha.x: 250.0
alpha.y: 350.0
beta.x: 60.0
beta.y: 50.0
seed: 111

Random [✓ BFS] Greedy

[✓ Show Box Boundary] Hide Box Boundary

Fig. 11. Bugtrap.



Input file: input1.txt
Input Directory: inputs
Radius: 5.0
Epsilon: 2.0
alpha.x: 130.0
alpha.y: 460.0
alpha.theta: 0.0
beta.x: 20.0
beta.y: 20.0
beta.theta: 0.0
seed: 1111

Random BFS [✓ Greedy]

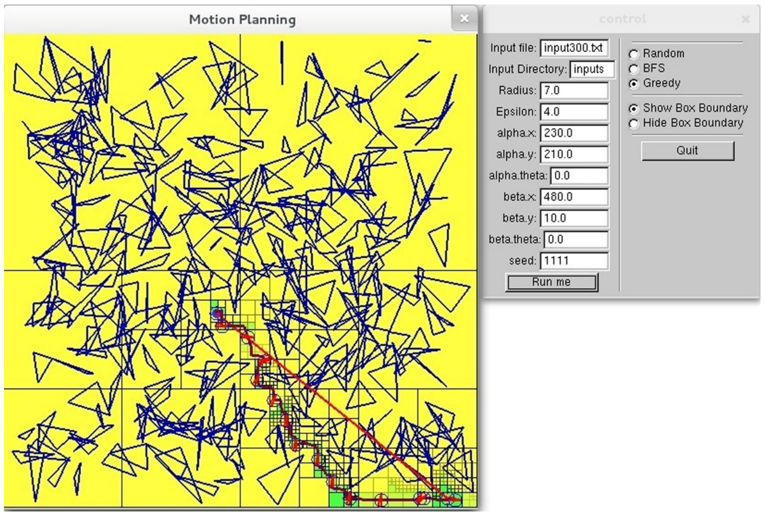
[✓ Show Box Boundary] Hide Box Boundary

Fig. 12. Input200: 200 random triangles.

graph comprising the FREE-leaves in $\mathcal{T}$. The MIXED-leaves of width $> \varepsilon$ are stored in a priority queue Q as candidates for expansion. The only difference between these two strategies is that *BFS* picks a box in Q with the *maximum size* to expand, while *Random* picks a *random* box in Q to expand.

In *GBF*, we do *not* use a union-find data structure; rather, we maintain a set $\mathcal{S}_\alpha$ of leaves of $\mathcal{T}$ which are FREE and connected to $Box_{\mathcal{T}}(\alpha)$. Boxes in $\mathcal{S}_\alpha$ are said to be "marked". A path is detected as soon as $Box_{\mathcal{T}}(\beta)$ is marked. Initially, $Box_{\mathcal{T}}(\alpha)$ is the only marked box. The priority queue Q contains all the FREE- or MIXED-leaves of $\mathcal{T}$ that are on the "fringe" of $\mathcal{S}_\alpha$ (a "fringe box" is unmarked but adjacent to some marked box). The priority of a box B in the queue is the "distance" from B to β. In case of a disc robot, this "distance" is the Euclidean distance from $m(B)$ to $\beta \in \mathbb{R}^2$. In case of a triangle robot, let $B = B_t \times A$ where $B_t \subseteq \mathbb{R}^2$ is the translational component of B, and A an angular range. Similarly, $\beta = (\beta_t, \theta)$ where $\beta_t \in \mathbb{R}^2$. Then "distance" is the Euclidean distance between $m(B_t)$ and β_t. Moreover, Q is a min-priority queue, so that box with the smallest distance to β has highest priority. We terminate with "NO PATH" if Q is empty; otherwise let B be a box extracted from Q for expansion. There are two cases: (1) If B is free, we mark B (i.e., add it to the set $\mathcal{S}_\alpha$) and for each neighbor B' of B, we push B' into Q if it is unmarked and is either FREE or MIXED. (2) If B is mixed, we first check its width. If the width is $\leq \varepsilon$, we discard B. Otherwise we expand B, and for each child B' of B, we push B' into Q if it is adjacent to a marked box and is either FREE or MIXED.

We use four input files: bugtrap, input150, input200, input300. Each file represents the environment as a set of polygons (not necessarily disjoint) within a 512×512 bounding box. The file inputNNN (where NNN = 150, 200 or 300) contains NNN randomly generated triangles. Three of these environments are shown in Figs. 11, 12, 13. The polygons edges are



Input file: input300.txt
Input Directory: inputs
Radius: 7.0
Epsilon: 4.0
alpha.x: 230.0
alpha.y: 210.0
alpha.theta: 0.0
beta.x: 480.0
beta.y: 10.0
beta.theta: 0.0
seed: 1111

Random [✓ Greedy] BFS

[✓ Show Box Boundary] Hide Box Boundary

Fig. 13. Input300: 300 random triangles.

shown in white (Fig. 11) or in blue (Figs. 12, 13). The paths found by our GBF search strategy are also shown ("NO PATH" in Fig. 11). We can see that the triangles may overlap, which can be handled by our approach easily.

In the following we present Table 1. The top shows the statistics of running our planners for disc and triangle robots on each input. The starting and ending configurations α and β are shown as (x, y) for disc robot and (x, y, θ) for triangle robot. The disc robot is specified as disc(r, Z) where r is the robot radius and $Z \in \{BFS, GBF, RAN\}$ indicates the search strategy. Similarly, the triangle robot is specified by tri(r, Z). Whenever the randomized strategy $Z = RAN$ is used, the statistics is the average of 5 runs; these are reproducibly encoded in Makefile targets in `Core Library`. Of the 3 strategies, we see that *GBF* is consistently the fastest. We have columns reporting the number of free, stuck, and mixed boxes. There were two kinds of mixed boxes: those of width $> \varepsilon$ and the rest. Note that when the number of mixed boxes of width $> \varepsilon$ is zero (last column), this implies 'NO PATH'. Excluding the cases of "Expanding $Box_{\mathcal{T}}(\alpha)$ fails" and "Expanding $Box_{\mathcal{T}}(\beta)$ fails" (trivial cases of 'NO PATH'), the converse is true only for the *BFS* or *Random* search strategy.[6] We explicitly mark the entries in the last column with an asterisk (*) to indicate 'NO PATH'.

We also directly compared our triangle robot with the *GBF* strategy (the instances of the top-table entries in bold, also shown in Figs. 11–13) with PRM and RRT, and show the results in the bottom table. For PRM, we ran the benchmark package OOPSMP [25], and the running times are shown as preprocessing, query, and total times in the bottom table. For RRT, we ran OOPSMP (denoted "RRT-OOP" in the table), the code by Prof. Jyh-Ming Lien (denoted "RRT-JML" in the table) of the robotics group in George Mason University, and the MSL library from Prof. Steven Lavalle's group in University of Illinois (UIUC). MSL did not seem to work well for our datasets of bounding boxes 512×512 (all its examples are of small bounding boxes 100×100) so we do not include its results in the table.

We remark that our only parameter is $\varepsilon > 0$. For PRM, OOPSMP requires user-chosen parameters like number of sample points, budgeted times for preprocessing and query, with default values 5000 points, 5 s and 5 s. Ideally, one would like to have an automatic process to find an optimal number of sample points that can result in a free path (if one exists). Unfortunately, the PRM of OOPSMP is poorly designed for such experiments, since each run needs user interactions to specify the parameters and other settings, and hence we were not able to make the process automatic. We did experiment with one instance, on bugtrap with triangle robot of radius 22 (i.e., the instance with No. 7, bugtrap (22), in the two tables): we first tried with 1000 sample points (33 ms), no path found, then increased for another 1000 samples (another 34 ms) to get 2000 samples, still no path. We then repeated the incremental process: another 1000 samples were added (another 34 ms) to get 3000 samples total, no path; finally, after adding 1000 more samples (another 36 ms, 4000 samples total), a free path was found (via query) in 3 ms. Overall, the incremental sampling up to 4000 points took 137 ms overall, plus the final query time of 3 ms, thus a total of 140 ms (plus much longer user interaction time!). For all other runs of PRM, we just used the default values (5000 samples), and increased the number of samples if 5000 was not enough to find a path. For RRT, the RRT of OOPSMP (RRT-OOP) does not allow the user to fine-tune any parameters, while RRT-JML provides additional flexibility to adjust parameters such as step size and goal bias, for which we tried for each instance to find reasonable values to use; the resulting number of samples are also shown in the table.

From the top portion of the bottom table, we see that our running times are competitive with PRM and RRT: among the 8 instances listed, we are the fastest in 4 of them, and the second fastest in another 3 of them; for the remaining instance

[6] We do *not* include the instances of such trivial cases of 'NO PATH' in our tables here.

Table 1

In the top table, the effect of increasing ε is seen in the first three rows. The 'NO PATH' instances are marked with "(*)" in the last column. In the bottom table, each row is an instance of the row in the top table with the corresponding line number (the "No." entry). Among the running times of various methods, the winner is shown in bold, and our results marked with "†" are the second fastest. The 'NO PATH' instances are marked with "(*)" in the first column.

No.	Obstacle (input)	Robot (radius)	α (x, y, θ)	β (x, y, θ)	eps	Time (ms)	Free	Stuck	Mixed $\le \varepsilon$	Mixed $> \varepsilon$
1	bugtrap	disc(14, GBF)	200, 350	60, 50	1	16	3867	2076	3403	462
2		disc(14, GBF)	200, 350	60, 50	2	10	1779	943	1750	275
3		disc(14, GBF)	200, 350	60, 50	4	5	854	460	801	151 (*)
4		**disc(15, GBF)**	200, 350	60, 50	0.5	**30**	8036	4193	6887	913
5		**disc(25, GBF)**	200, 350	60, 50	1	**18**	3260	1709	2984	415 (*)
6		**tri(14, GBF)**	200, 350, 0	60, 50, 0	5	**245**	19 394	26	31 121	220
7		**tri(22, GBF)**	200, 350, 0	60, 50, 0	5	**482**	39 482	1556	57 546	3880
8		**tri(50, GBF)**	200, 350, 0	60, 50, 0	5	**746**	55 864	4153	79 411	9025 (*)
9	input150	disc(7, RAN)	200, 270	20, 20	2	428	6892	10 082	8027	1955
10		**tri(7, GBF)**	200, 270, 0	20, 20, 0	5	**10**	945	0	1334	360
11		tri(7, BFS)	200, 270, 0	20, 20, 0	5	1349	152 841	366	0	608 432
12		tri(7, RAN)	200, 270, 0	20, 20, 0	5	315	32 179	1028	101 322	32 477
13		**tri(7, GBF)**	325, 425, 0	20, 20, 0	5	**216**	10 233	129	19 382	1389
14	input200	disc(5, BFS)	130, 460	20, 20	2	16	2590	4891	0	5636
15		**disc(5, GBF)**	130, 460	20, 20	2	**15**	283	99	230	131
16		**tri(5, GBF)**	130, 460, 0	20, 20, 0	2	**89**	16 866	160	0	29 602
17		tri(5, BFS)	130, 460, 0	20, 20, 0	2	1742	182 866	1036	0	747 445
18		tri(5, RAN)	130, 460, 0	20, 20, 0	2	3940	331 830	7044	0	1 408 722
19	input300	disc(7, BFS)	230, 210	480, 10	4	23	3785	11 284	7465	0 (*)
20		disc(7, BFS)	230, 210	480, 10	1	35	6439	15 339	0	11 052
21		**disc(3, GBF)**	230, 210	480, 10	1	**29**	3986	1760	3529	1001
22		**tri(7, GBF)**	230, 210, 0	480, 10, 0	4	**32**	5212	0	0	11 686
23		tri(7, BFS)	230, 210, 0	480, 10, 0	4	2101	110 005	667	0	899 054
24		tri(7, RAN)	230, 210, 0	480, 10, 0	4	7470	371 119	8539	0	2 694 907
25		**tri(7, GBF)**	10, 500, 0	480, 10, 0	4	**478**	25 893	106	42 274	2245

No.	Obstacle input file (robot radius)	Tri(GBF) Time (ms)	PRM				RRT-JML				RRT-OOP Time (ms)
			No. of samples	Prep. (ms)	Query (ms)	**Total** (ms)	No. of samples	Step size	Goal bias	**Time** (ms)	
6	bugtrap (14)	**245**	5000	256	3	259	37 009	0.05	0.5	922	277
7	bugtrap (22)	482 †	5000	270	3	**273**	156 276	0.01	0.1	24 092	796
8 (*)	bugtrap (50)	**746**	125 000	32 368	7	32 375	500 000	0.01	0.1	134 912	60 000
10	input150 (7)	**10**	5000	176	2	178	4309	0.05	0.5	270	17
13	input150 (7)	216	5000	176	2	178	7635	0.05	0.5	375	**62**
16	input200 (5)	**89**	5000	203	5	208	18 757	0.05	0.5	1624	151
22	input300 (7)	32 †	5000	145	0.3	145.3	3538	0.05	0.5	398	**14**
25	input300 (7)	478 †	5000	145	0.3	**145.3**	57 619	0.05	0.5	2390	945
	Obstacle input file (robot radius)	Disc(GBF) Time (ms)	PRM				RRT-JML				RRT-OOP Time (ms)
			No. of samples	Prep. (ms)	Query (ms)	**Total** (ms)	No. of samples	Step size	Goal bias	**Time** (ms)	
4	bugtrap (15)	**30**	25 000	2135	9	2144	N/A				3979
5 (*)	bugtrap (25)	**18**	125 000	22 594	3	22 597	N/A				60 000
15	input200 (5)	**15**	5000	23 438	7	23 445	N/A				2335
21	input300 (3)	**29**	5000	765	2	767	N/A				1129

we are the third. In some cases we are much faster than all others. In particular, for the instance of 'NO PATH' (No. 8), our method stopped and reported 'NO PATH' easily in 746 ms, while for PRM we gave up after trying 125 000 samples (32 375 ms), for RRT-JML we gave up after trying the run of 500 000 samples (1 34 912 ms), and for RRT-OOP we gave up after using up the maximum allowed run-time of 60 000 ms. Clearly our method is much more advantageous.

Finally, we compared our disc robot with PRM and RRT-OOP (we do not have the RRT-JML code for disc robot). For both PRM and RRT-OOP, the robot must be a polygon; we approximated the disc robot by a same-radius regular 20-gon. The results are shown in the bottom portion of the bottom table. Note that for the row of No. 4, PRM used 25 000 samples since no path was found for 5000 samples. For the row of No. 5, which is an instance of 'NO PATH', for PRM we gave up after using 125 000 samples, and for RRT-OOP we gave up after using up the maximum allowed run-time of 60 000 ms. From the table, we can see that clearly our method is significantly faster than all others in all instances.

8. Conclusions

The motion planning literature has a bipolar nature—many algorithms are theoretically sound but unimplementable, others are practical but lack theoretical foundations or proper implementation. The dominant approach based on randomization offer some theoretical guarantees but they have issues: there are no guarantees in case of NO PATH, and "expansiveness" assumptions [18] are non-verifiable. This paper takes up the classic subdivision paradigm to develop a theoretically sound alternative. To aid the development of such algorithms, we introduce soft predicates and demonstrated their use in subdivision planners. We introduced the concept of resolution-exact planners, and designed the first examples of such algorithms. We also show the inherent indeterminacy of resolution-exactness. Finally, our implementations validate the claims that our theory is practical; the experiments demonstrate that our approach is competitive with PRM in speed, despite our much stronger guarantees.

According to Zhang et al. [45], implementations of exact motion planning algorithms are only known for simple planer robots (like ladders or discs) and up to 3 degrees of freedom. Thus it is important to pay attention to implementability. We propose to give up exactness for the weaker notion of resolution-exactness. Little is lost by this step, since exact algorithms are ill-matched to the inherent inaccuracies of physical systems. But we have much to gain: Subdivision algorithms are more holistic, integrating the concerns of topological correctness with geometric accuracy into one algorithm.

The techniques of this paper can be extended to robots with complex geometry (e.g., the "gear" robot [45]). We could decompose the complex robot geometry into a union of (possibly overlapping) triangles. If we now have soft predicates for each of the triangle robots, we could compose them into a soft predicate for the complex robot. This remarkable decomposition property of soft predicates has no analogue in exact algorithms. A subtlety is that the triangle robots are not free to choose its origin; this freedom was exploited in Section 6 above. This extension will be described in a followup work.

Several open problems are raised by this research. (1) Clearly, a more general theory of subdivision planners can be developed; see our companion paper [44] where many of the ideas here are generalized. (2) We can extend our work to subdivision of $\mathtt{SE}(3) = \mathbb{R}^3 \times \mathtt{SO}(3)$, and believe this too can be competitive with PRM. Note that no general exact algorithms have been implemented for $\mathtt{SE}(3)$. (3) Note that we have not tried to compute the connected components of STUCK boxes. Doing this can lead to fast termination in the case of "NO PATH". However, maintaining this information runs into interesting issues of computational topology. Edelsbrunner and Delfinado's work on computing the Betti number of a 3-complex offers some clues here [11]. (4) General investigation of various search strategies, including probabilistic ones is needed.

We plan to explore other variants of our search strategies with an eye to simplicity, implementability, and correctness. Our approach can be extended to more demanding motion planning problems such as kinodynamic problems or those with differential constraints.

Acknowledgements

We thank Prof. Jyh-Ming Lien and his student Zhonghua Xi for sharing their RRT code with us, which was used to conduct some of the experiments in this paper.

References

[1] M. Barbehenn, S. Hutchinson, Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees, IEEE Trans. Robot. Autom. 11 (2) (1995).
[2] M. Barbehenn, S. Hutchinson, Toward an exact incremental geometric robot motion planner, in: Proc. Intelligent Robots and Systems 95, vol. 3, IEEE/RSJ Intl. Conf., 5–9 Aug 1995, Pittsburgh, PA, USA, 1995, pp. 39–44.
[3] R. Bohlin, L. Kavraki, A randomized algorithm for robot path planning based on lazy evaluation, in: P. Pardalos, S. Rajasekaran, J. Rolim (Eds.), Handbook on Randomized Computing, Kluwer Academic Publishers, 2001, pp. 221–249.
[4] M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Perez, M. Mason, Robot Motion: Planning and Control, MIT Press, 1982.
[5] R.A. Brooks, T. Lozano-Perez, A subdivision algorithm in configuration space for findpath with rotation, in: Proc. 8th Intl. Joint Conf. on Artificial Intelligence, vol. 2, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1983, pp. 799–806.
[6] M. Burr, F. Krahmer, SqFreeEVAL: an (almost) optimal real-root isolation algorithm, J. Symb. Comput. 47 (2) (2012) 153–166.
[7] M. Burr, F. Krahmer, C. Yap, Continuous amortization: a non-probabilistic adaptive analysis technique, Electron. Colloq. Comput. Complex. TR09 (December 2009) 136.
[8] J. Canny, Computing roadmaps of general semi-algebraic sets, Comput. J. 36 (5) (1993) 504–514.
[9] E.-C. Chang, S.W. Choi, D. Kwon, H. Park, C. Yap, Shortest paths for disc obstacles is computable, in: 21st ACM Symp. on Comp. Geom., June 5–8, Pisa, Italy, 2005, pp. 116–125.
[10] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations, MIT Press, Boston, 2005.
[11] C. Delfinado, H. Edelsbrunner, An incremental algorithm for Betti numbers of simplicial complexes on the 3-sphere, Comput. Aided Geom. Des. 12 (1995) 771–784.
[12] B. Donald, P. Xavier, Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamics bounds, Algorithmica 14 (1995) 443–479.
[13] S.J. Fortune, A sweepline algorithm for Voronoi diagrams, Algorithmica 2 (1987) 153–174.
[14] GNU MP Homepage, Since 1991, GNU MP (=GMP) is a free C++ library for arbitrary precision arithmetic on integers, rationals and floating point numbers, URL http://gmplib.org.
[15] D. Halperin, L. Kavraki, J.-C. Latombe, Robotics, in: J.E. Goodman, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, CRC Press LLC, 1997, pp. 755–778, Chap. 41.
[16] K. Hauser, Motion planning for legged and humanoid robots, PhD thesis, Department of Computer Science, Stanford University, 2008.

[17] M. Hemmer, O. Setter, D. Halperin, Constructing the exact Voronoi diagram of arbitrary lines in three-dimensional space, in: Algorithms—ESA 2010, in: Lecture Notes in Computer Science, vol. 6346, Springer, Berlin/Heidelberg, 2010, pp. 398–409.
[18] D. Hsu, J.-C. Latombe, H. Kurniawati, On the probabilistic foundations of probabilistic roadmap planning, Int. J. Robot. Res. 25 (7) (2006) 627–643.
[19] L. Kavraki, P. Švestka, C. Latombe, M. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 12 (4) (1996) 566–580.
[20] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, 1991.
[21] S.M. LaValle, Planning Algorithms, Cambridge University Press, Cambridge, 2006.
[22] R.E. Moore, Interval Analysis, Prentice Hall, Englewood Cliffs, NJ, 1966.
[23] MPFR Homepage, Since 2000, MPFR is a C++-library for multi-precision floating-point computation with exact rounding modes, URL http://www.mpfr.org/.
[24] C. Ó'Dúnlaing, C.K. Yap, A "retraction" method for planning the motion of a disc, J. Algorithms 6 (1985) 104–111. Also, Chap. 6 in: Sharir, Schwartz, Hopcroft (Eds.), Planning, Geometry, and Complexity, Ablex Pub. Corp., Norwood, NJ, 1987.
[25] E. Plaku, K. Bekris, L. Kavraki, OOPS for motion planning: an online open-source programming system, in: IEEE Intl. Conf. Robotics and Automation, 2007, pp. 3711–3716.
[26] J.H. Reif, H. Wang, Nonuniform discretization for kinodynamic motion planning and its applications, SIAM J. Comput. 30 (2000) 161–190.
[27] M. Sagraloff, When Newton meets Descartes: a simple and fast algorithm to isolate the real roots of a polynomial, in: Proc. ISSAC 2012, 2012, pp. 297–304.
[28] M. Sagraloff, C.K. Yap, A simple but exact and efficient algorithm for complex root isolation, in: I.Z. Emiris (Ed.), 36th Int'l Symp. Symbolic and Alge. Comp., June 8–11, San Jose, California, 2011, pp. 353–360.
[29] O. Salzman, M. Hemmer, B. Raveh, D. Halperin, Motion planning via manifold samples, in: Proc. European Symp. Algorithms (ESA), 2011.
[30] J.T. Schwartz, M. Sharir, On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers, Commun. Pure Appl. Math. 36 (1983) 345–398.
[31] J.T. Schwartz, M. Sharir, On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds, Adv. Appl. Math. 4 (1983) 298–351.
[32] J.T. Schwartz, M. Sharir, J. Hopcroft (Eds.), Planning, Geometry and Complexity of Robot Motion, Ablex Series in Artificial Intelligence, Ablex Publishing Corp., Norwood, New Jersey, 1987.
[33] M. Sharir, C. O'D'únlaing, C. Yap, Generalized Voronoi diagrams for moving a ladder I: topological analysis, Commun. Pure Appl. Math. XXXIX (1986) 423–483. Also: NYU-Courant Institute, Robotics Lab., No. 32, Oct 1984.
[34] M. Sharir, C. O'D'únlaing, C. Yap, Generalized Voronoi diagrams for moving a ladder II: efficient computation of the diagram, Algorithmica 2 (1987) 27–59. Also: NYU-Courant Institute, Robotics Lab., No. 33, Oct 1984.
[35] V. Sharma, C. Yap, Near optimal tree size bounds on a simple real root isolation algorithm, in: 37th Int'l Symp. Symbolic and Alge. Comp., ISSAC'12, July 22–25, 2012, Grenoble, France, 2012, pp. 319–326.
[36] G. Varadhan, S. Krishnan, T. Sriram, D. Manocha, Topology preserving surface extraction using adaptive subdivision, in: Proc. Symp. on Geometry Processing, SGP'04, 2004, pp. 235–244.
[37] G. Varadhan, D. Manocha, Star-shaped roadmaps—a deterministic sampling approach for complete motion planning, in: Robotics: Science and Systems, 2005, pp. 25–32.
[38] G. Varadhan, D. Manocha, Accurate Minkowski sum approximation of polyhedral models, Graph. Models 68 (4) (2006) 343–355.
[39] C. Yap, V. Sharma, J.-M. Lien, Towards exact numerical Voronoi diagrams, in: 9th Proc. Int'l. Symp. of Voronoi Diagrams in Science and Engineering, ISVD, Rutgers University, NJ, June 27–29, 2012, IEEE, 2012, pp. 2–16, Invited talk.
[40] C.K. Yap, Algorithmic motion planning, in: J. Schwartz, C. Yap (Eds.), Advances in Robotics, in: Algorithmic and Geometric Issues, vol. 1, Lawrence Erlbaum Associates, 1987, pp. 95–143.
[41] C.K. Yap, An $O(n \log n)$ algorithm for the Voronoi diagram for a set of simple curve segments, Discrete Comput. Geom. 2 (1987) 365–394. Also: NYU-Courant Institute, Robotics Lab., No. 43, May 1985.
[42] C.K. Yap, Robust geometric computation, in: J.E. Goodman, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, 2nd edition, Chapman & Hall/CRC, Boca Raton, FL, 2004, pp. 927–952, Chap. 41.
[43] C.K. Yap, In praise of numerical computation, in: S. Albers, H. Alt, S. Näher (Eds.), Efficient Algorithms, in: Lect. Notes in C.S., vol. 5760, Springer-Verlag, 2009, pp. 308–407.
[44] C.K. Yap, Soft subdivision search in motion planning, in: Proceedings, Robotics Challenge and Vision Workshop, RCV 2013, 2013, Best Paper Award, sponsored by Computing Community Consortium (CCC); in: Robotics Science and Systems Conference, RSS 2013, Berlin, Germany, June 27, 2013, 2013, The full paper is available from http://cs.nyu.edu/exact/papers/.
[45] L. Zhang, Y.J. Kim, D. Manocha, Efficient cell labelling and path non-existence computation using C-obstacle query, Int. J. Robot. Res. 27 (11–12) (2008).
[46] D. Zhu, J.-C. Latombe, New heuristic algorithms for efficient hierarchical path planning, IEEE Trans. Robot. Autom. 7 (1991) 9–20.